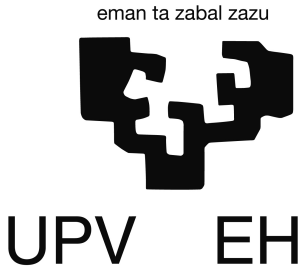


PhD Thesis

A FRAMEWORK FOR TRAFFIC ENGINEERING IN SOFTWARE-DEFINED NETWORKS WITH ADVANCE RESERVATION CAPABILITIES

ALAITZ MENDIOLA ALCARAZ

Supervised by Eduardo Jacob Taquet and Jasone Astorga Burgo

University of the Basque Country UPV/EHU
Department of Communications Engineering

2017

Dedicated to the loving memory of Fernando and Marian.

Abstract

The appearance of Software-Defined Networking (SDN) has cleared the pace for the next generation of networks. In a nutshell, SDN proposes the separation of the forwarding and control planes, where the control of the network is performed by means of a logically centralized control plane that resides in an external element and programs the forwarding plane using open interfaces. With this new paradigm in mind, network operators worldwide have started to evolve their network architectures, in order to reduce their CAPEX and OPEX expenditures and facilitate the introduction of new services more efficiently.

Among the benefits introduced by SDN, network operators are interested in its capability to control the network taking into account its overall state. This capability of SDN can revolutionize Traffic Engineering (TE), one of the most challenging topics in communication networks, where the logically centralized control plane enables the utilization of novel TE strategies and the fine granularity available to forward the packets makes possible the use of alternative load balancing approaches.

Moreover, Research and Education Networks (REN) that offer high availability and high quality services to the research and academic community have started to include SDN in their network evolution plans, attracted as well by the impact of SDN in TE. In RENs, TE plays a key role, where the nature of the data being transmitted through these kinds of networks requires fast failure recovery capabilities and an efficient utilization of the network resources to cope with the growing data traffic.

More precisely, one of the services typically provided to the research community is Bandwidth on Demand (BoD), which allows to establish end-to-end connectivity services during a specific period of time with a guaranteed bandwidth. The provisioning of the BoD service requires the utilization of advance reservation mechanisms, typically used in grid computing and optical networks. In addition, these kind of services are usually provided in a multi-domain fashion, that is, involving multiple network domains.

Notwithstanding, the introduction of SDN, makes possible to rethink how TE should be supported and how the resource and timing constraints to support advance reservations should be handled in such an environment. On the one hand, an in-depth analysis of current solutions for TE based on SDN has been conducted, in order to identify the features that a TE framework for SDN should provide. On the other hand, current advance reservation systems, and more precisely, the impact that the data structures utilized to handle resource and timing constraints have in the performance of such systems has been evaluated.

As a consequence of these analysis, this PhD thesis introduces DynPaC, a novel framework for TE in SDN that by means of a modular and technology agnostic approach, aims to satisfy the TE requirements of network operators independent of the scope in which it is utilized and the use case. It is worth remarking that DynPaC has been designed taking into account the requirements proposed by the network operations team of the pan-European REN GÉANT, which provides connectivity services to 50 million users across the globe.

In that regard, the DynPaC framework supports advance reservations and is compliant with the Network Services Framework (NSF) employed to provide the multi-domain BoD service. Moreover, it improves the current BoD service provided by GÉANT by adding automatic topology discovery and resilience capabilities. In addition, its modular approach and its capability to introduce novel TE mechanisms has allowed to incorporate two mechanisms that make possible the improvement of the network resource utilization, the flow relocation mechanism and the flow disaggregation mechanism.

The advance reservation capabilities of DynPaC rely on a set of data structures and algorithms specifically designed to leverage the logically centralized control plane and the high programmability available in SDN. The solution is based on the generation of *network snapshots*, a data structure that represents a period of time during which the resource consumption of a given set of concurrent service reservations remains stable. These network snapshots are arranged using a hierarchical data structure called Network Snapshots Tree (NSTree), which enables the representation of wider time intervals where a subset of common resources

are available by aggregating multiple network snapshots into auxiliary nodes.

This approach presents some benefits. On the one hand, the utilization of network snapshots allows to find different but equally valid paths for the given resource constraints in the different network snapshots in which the service exists. Furthermore, this also makes possible the introduction of the aforementioned resource utilization optimization techniques named flow relocation and flow disaggregation, which allow to accept new service demands that otherwise would not be accepted.

On the other hand, by aggregating the network snapshots, the time required to accept or reject a service gets reduced. In advance reservations it is necessary to check whether there are enough resources to satisfy the demand during the entire life span of the reservation. In this particular case, this requires analyzing all the affected network snapshots. By arranging the network snapshots in a hierarchical data structure, the running time of the solution is improved, achieving a logarithmic running time instead of a linear one.

In order to prove the suitability of the solution for the REN environment, it has been conducted a functional validation where the connectivity requirements proposed by GÉANT's network operations team has been successfully tested, so as the multi-domain capabilities of the solution. Moreover, the performance of the solution has been evaluated using different topologies, including the topology of one of the most renowned RENs in the United States, the Energy Sciences Network. The experiments conclude that the DynPaC framework can satisfy the requirements of the RENs, and that the data structures and algorithms used to handle the advance reservations are able to accept or reject a service reservation request within the order of the milliseconds.

Finally, the solution presented in this PhD thesis has been implemented using one of the most important SDN controllers in the market, the open source ONOS controller. It is worth remarking that the DynPaC framework will be deployed in the pilot phase of the new SDN-based BoD service of GÉANT by the end of 2017. Furthermore, the Open Networking Laboratory is planning to include some of the features included in the DynPaC framework in future releases of the ONOS controller.

Laburpena

Software bidez Definitutako Sareen (SDN, Software-Defined Network) agerpena funtsezkoa izango da sareen hurrengo belaunaldiaren garapenerako. Laburki esanda, SDNek datu eta kontrol planoak bereiztea proposatzen dute, sarearen kontrola logikoki zentralizatuta dagoen kontrol plano baten bidez burutzen delarik. Logikoki zentralizatutako kontrol plano hau saretik kanpo dagoen elementu batean kokatzen da eta sareko elementuen datu planoa programatzen du interfaze irekien bitartez. Sare paradigma berri hau kontutan izanda, sareen operadoreek beraien sareetako arkitekturak eboluzionatzen hasi dira, hasierako kostuak zein operazio kostuak murrizteko asmoarekin eta sareko zerbitzu berrien sarrera era eraginkorrago batean ahalbidetzeko asmoarekin.

SDNek ekarritako onura guztien artean, sareko operadoreentzat interesgarriena zera da, SDNek sarea kontrolatzeko daukaten gaitasuna, sare horien egoera orokorra kontutan izanda. SDNen gaitasun honek erabateko iraultza ekar dezake Trafikoaren Ingeniaritzaren (TE, Traffic Engineering) arlora, komunikazio-sareen munduan erronka garrantzitsuak planteatzen dituen arloa hain zuzen ere. Izan ere, SDNen logikoki zentralizatutako kontrol planoak TEko estrategia berrien erabilpena ahalbidetzen du eta datu planoan, berriz, trafiko fluxuak bereizmen handiarekin banatzeko gaitasunak, sareko zama orekatzeko teknika berritzaileak erabiltzeko aukera ematen du.

Zentzu horretan, Sare Akademikoak (REN, Research and Education Network), zientzia erkidegoari eskuragarritasun eta kalitate altuko zerbitzuak eskaintzen dizkiotenak, beraien sareen eboluziorako plangintzetan SDNak sartzen hasi dira, SDNek TEn eduki dezaketen eraginak erakarrita. RENetan TEk funtsezko papera jokatzen du, sare mota hauen bitartez bidalitako datuen izaerak hutsegiteei aurre egiteko tekniken inplementazioa eta sareko baliabideen erabilpen eraginkorra eskatzen baititu, azken hau bidalitako datu bolumenaren etengabeko handiagotzea onartu ahal izateko.

RENek zientzia erkidegoari emandako ohiko zerbitzu bat Eskatu Ahalako Banda-Zabalera (BoD, Bandwidth on Demand) deritzona da. Zerbitzu honen bitartez denbora epe zehatz baterako banda zabalera bermatua daukaten muturretik muturrerako konektibitate zerbitzuak ezartzea posiblea da. BoD zerbitzua emateko aldez aurretik erreserbak egiteko mekanismoak beharrezkoak dira, sare optikoetan eta grid konputazioan sarritan erabili ohi direnak. Gainera, zerbitzu mota hauek domeinu anitzetako testuinguruetan erabilgarriak izan ohi dira, hau da, zerbitzuek sare domeinu bat baino gehiagoren bitartez ezartzen diren testuinguruetan.

Hala ere, SDNen eta ohiko sareen arteko desberdintasunak kontutan izanda, sare berri hauetan aldez aurretiko erreserbak onartzen dituzten zerbitzuak emateko, berriro ere hausnartu behar da TEa zelan emango den eta sare baliabideak zelan kudeatuko diren denboraren arabera. Horretarako, gaur egun SDNetarako existitzen diren TE proposamenen azterketa sakona burutu da, SDNetarako TE mekanismo egoki batek eman behar dituen funtzionaltasunak identifikatzeko xedearekin. Beste alde batetik, gaur egungo aldez aurretiko erreserba sistemak aztertu dira ere, denbora eta sareko baliabideen eskakizunak kudeatzeko erabilitako datu egiturek daukaten eragina azpimarratuz, datu egitura hauek erreserba sistemen errendimenduan daukaten eragina ulertzeko xedearekin.

Azterketa hauen ondorio bezala, doktorego-tesi honek DynPaC framework-a aurkezten du, SDNetarako TE framework berri bat dena. SDN teknologiekiko independentea den diseinu modular baten bitartez, DynPaC framework-aren helburua sareko operadoreen TE eskakizunak asetzea da, edozein sare mota eta erabilpen kasurako. Aipatzekoa da DynPaC 50 milioi erabiltzaile baino gehiago dituen GÉANT RENaren eragiketarako taldeak proposatutako betebeharrak kontutan hartuta diseinatu dela.

Zentzu horretan, DynPaCek aldez aurreko zerbitzuak ematen ditu eta domeinu anitzetako BoD zerbitzua emateko erabili ohi den Network Services Framework-arekin (NSF) bat dator. Are gehiago, DynPaCek gaur egun GÉANT-ek ematen duen BoD zerbitzua hobetzen du, sareko topologia automatikoki detektatzeko eta hutsegiteak kudeatzeko mekanismoak ematen baititu. Gainera, DynPaCen diseinu modularrari

eta TE mekanismo berriak sartzeko aurkezten duen gaitasunari esker, sareko baliabideen erabilpena hobetzen duten bi mekanismo gehitzea posiblea izan da: fluxuen bir-kokapenerako mekanismoa eta fluxuen desagregaziorakoa.

Aldez aurreko erreserbak egiteko gaitasunari dagokionez, DynPaC, SDNen logikoki zentralizatutako kontrol planoa eta programagarritasun altua ustiatzeko bereziki diseinatutako datu egitura eta algoritmo talde batean oinarritzen da. Proposatutako ebazpena sarearen uneko egoeren sorkuntzan oinarritzen da, network snapshots bezala ezagutzen direnak. Network snapshot hauek aldi berean existitzen diren zerbitzu erreserben multzo zehatz baten ondorioz, sareko baliabideen erabilpena egonkor mantentzen duten denbora tarteak adierazten dituzten datu egiturak dira. Network snapshot hauek datu egitura hierarkiko baten bidez antolatzen dira, sarearen uneko egoeren zuhaitza deritzona. Egitura honek network snapshot bat baino gehiago bateratzen ditu, sareko baliabideen azpimultzo bat eskuragarri mantenduko duten denbora tarte luzeagoak adieraztea ahalbidetuz.

Hurbilketa honek zenbait abantaila ematen ditu. Alde batetik, network snapshot-en erabilpenari esker zerbitzu erreserba bati dagozkion network snapshot desberdinetan bide aukera desberdin, baina era berean baliagarriak, aurkitzea posible da. Are gehiago, hurbilketa honek lehen aipatutako optimizazio tekniken aplikazioa errazten du ere, fluxuen bir-kokapenerako mekanismoak eta fluxuen desagregaziorako mekanismoak alegia. Teknika hauei esker, beste modu batean ukatuko liratekeen zerbitzuen erreserbak onartzea posible da.

Beste alde batetik, network snapshot-ak nodo laguntzaileetan bateratuz, zerbitzu erreserba bat onartu daitekeen zehazteko beharrezkoa den denbora murriztu daiteke. Aldez aurreko erreserbak onartzen dituzten sistemetan, zerbitzu eskaera batek iraungo duen denbora tarte osoan eskaera asetzeko behar beste baliabide eskuragarri egongo direla egiaztatu behar da. Honek, aldaketak izango dituzten network snapshot guztiak aztertzea suposatuko luke. Network snapshot-ak datu egitura hierarkiko batean antolatuz, zerbitzu eskariak onartzeko prozesuak behar duen konputazio denbora murrizten da, portaera lineal batetik portaera logaritmiko batera pasatuz.

Proposatutako ebazpena RENentzako bideragarria dela frogatzeko helburuarekin, balioztatze funtzional bat burutu da. Balioztatze funtzional honen bitartez doktorego-tesi honetan proposatutako ebazpenak GÉANTeko sare-eragiketarako taldeak adierazitako betebeharrak asetzen dituela egiaztatu da, baita domeinu anitzetako testuinguruetan zerbitzua emateko gai dela ere. Bestalde, ebazpenaren errendimenduaren ebaluazio bat burutu da ere. Honetarako sare topologia desberdinak erabili dira, hauen artean, ESNet REN amerikarraren sareko topologia ere. Burututako saiakuntzek DynPaCek RENen eskakizunak asetzen dituela frogatzen dute, baita aldez aurreko erreserben kudeaketarako erabilitako datu egiturek eta algoritmoek zerbitzu eskarien onarpenerako prozesua milisegundoen inguruko denboran burutu dezaketela ere.

Azkenik, tesi honetan aurkeztutako ebazpena merkatuko SDNen kontrolatzaile garrantzitsuenetariko bat erabiliz inplementatu da, kode irekiko ONOS kontrolatzailea hain zuzen ere. Gainera, azpimarratu beharrekoa da DynPaC GÉANTek emango duen BoD zerbitzu berriaren pilotuan erabiliko dela, SDNetan oinarrituko dena, eta 2017 urtean zehar martxan ipiniko dena. Are gehiago, Open Network Laboratory delakoa, ONOS kontrolatzailearen garapenaren erantzule dena, DynPaCeko funtzionaltasun batzuk kontrolatzaile honen hurrengo bertsioetan sartzea pentsatzen dago.

Resumen

La aparición de las Redes Definidas por Software (SDN, Software-Defined Network) será clave para la próxima generación de redes. En resumen, las SDNs proponen la separación del plano de datos y el plano de control, donde el control de la red es llevado a cabo mediante un plano de control lógicamente centralizado que se sitúa en un elemento externo al equipo de red, y el cual programa el plano de datos mediante interfaces abiertas. Teniendo en mente este novedoso paradigma de red, los operadores de red han comenzado a evolucionar sus arquitecturas de red, con el propósito de reducir tanto los costes iniciales como los costes operacionales y facilitar la introducción de nuevos servicios de red de manera más eficiente.

Entre los beneficios proporcionados por las SDNs, los operadores de red están especialmente interesados en su capacidad de controlar la red teniendo en cuenta el estado global de la misma. Esta capacidad de las SDN puede suponer una auténtica revolución en la Ingeniería de Tráfico (TE, Traffic Engineering), uno de los temas más desafiantes en el mundo de las redes de comunicaciones, donde el plano de control lógicamente centralizado posibilita la utilización de estrategias novedosas de TE y donde la alta granularidad disponible en el plano de datos hace posible la utilización de técnicas alternativas para el balanceo de la carga en la red.

En ese sentido, las Redes Académicas (REN, Research and Education Network), las cuales ofrecen servicios de alta disponibilidad y alta calidad a la comunidad científica, han comenzado a incluir las SDNs en sus planes de evolución de red, atraídos por el impacto que las SDNs puede suponer en la TE. En las RENs, la TE juega un papel fundamental, donde la naturaleza de los datos transmitidos a través de este tipo de redes requiere la implementación de técnicas de recuperación frente a errores y una eficiente utilización de los recursos de red, para así asimilar el continuo crecimiento del volumen de datos transmitidos.

Uno de los servicios típicamente proporcionados a la comunidad científica por las RENs es el Ancho de Banda bajo Demanda (BoD, Bandwidth on Demand), el cual permite establecer servicios de

conectividad extremo a extremo durante un periodo de tiempo específico y con un ancho de banda garantizado. La provisión del servicio BoD requiere la utilización de mecanismos de reserva avanzada, comúnmente utilizados en redes ópticas y en la computación grid. Además, este tipo de servicios suelen incluir soporte para múltiples dominios, en los que el servicio se establece a lo largo de múltiples dominios de red.

No obstante, dadas las diferencias de las SDNs con respecto a las redes tradicionales, es necesario preguntarse cómo se ha de soportar la TE y cómo se han de gestionar los recursos de red en función del tiempo, para proporcionar servicios de reserva avanzada en este tipo de redes. Para ello, se ha llevado a cabo un exhaustivo análisis de las soluciones existentes de TE en SDNs, a fin de identificar las funcionalidades que una solución para la TE en SDNs debería proporcionar. Por otra parte, también se han estudiado los sistemas de reserva avanzada actuales, haciendo especial hincapié en el impacto que las estructuras de datos empleadas para gestionar los requerimientos de tiempo y recursos de red, a fin de comprender el impacto que las mismas tienen en el rendimiento de tales sistemas.

Como consecuencia de estos análisis, esta tesis doctoral presenta DynPaC, un novedoso framework para la TE en SDNs, el cual, mediante un diseño modular e independiente de la tecnología SDN empleada, pretende satisfacer los requerimientos de TE de los operadores de red, independientemente del tipo de red y el caso de uso en el que se utilice. Cabe destacar que DynPaC ha sido diseñado teniendo en consideración los requerimientos propuestos por el equipo de operación de red de la REN europea GÉANT, la cual proporciona servicios de conectividad a más de 50 millones de usuarios.

En ese sentido, DynPaC soporta la provisión de servicios de reserva avanzada, y es compatible con el Network Services Framework (NSF), utilizado habitualmente para proporcionar el servicio BoD con multi-dominio. Es más, DynPaC mejora el servicio actual de BoD proporcionado por GÉANT al introducir mecanismos para el descubrimiento de topología y para la gestión de los fallos de red. Además, su diseño modular y su capacidad para introducir nuevos mecanismos de TE ha permitido incorporar dos mecanismos que hacen posible la mejora de la utilización

de los recursos de red, el mecanismo para el realojamiento de flujos y el mecanismo para la desagregación de flujos.

En relación al soporte de reservas avanzadas, DynPaC se basa en la utilización de una serie de estructuras de datos y algoritmos específicamente diseñados para aprovechar el plano de control lógicamente centralizado y la alta programabilidad disponible en las SDNs. La solución se basa en la generación de *network snapshots*, una estructura de datos que permite representar periodos de tiempo en los que la utilización de los recursos de red permanece estable como consecuencia de la coexistencia de un conjunto dado de reservas de servicio. Los network snapshots se organizan mediante una estructura de datos jerárquica, denominada Árbol de Instantáneas de Red (NSTree, Network Snapshot Tree), que permite la representación de periodos de tiempo más amplios en los que un subconjunto de los recursos de red están disponibles, como consecuencia de agregar múltiples network snapshots en nodos auxiliares.

Esta aproximación presenta una serie de ventajas. Por una parte, la utilización de network snapshots permite hallar caminos alternativos pero igualmente válidos en los diferentes network snapshots en los que una reserva de servicio está contemplada. Es más, esta aproximación también posibilita la introducción de las técnicas de optimización previamente mencionadas, denominadas realojamiento de flujos y desagregación de flujos, que facilitan la aceptación de nuevas reservas de servicio que de otra manera serían rechazadas.

Por otra parte, al agregar los network snapshots en nodos auxiliares, el tiempo necesario para determinar si una reserva de servicio se puede aceptar se ve reducido. En los sistemas de reserva avanzada es necesario comprobar si hay recursos suficientes para satisfacer la demanda durante todo el periodo de tiempo solicitado. En este caso particular, supondría analizar todos los network snapshots que se ven afectadas. Al organizar los network snapshots en una estructura de datos jerárquica, el tiempo de computo del proceso de admisión de las demandas de servicio se ve reducido, pasando de un comportamiento lineal a un comportamiento logarítmico.

Con el objetivo de demostrar la viabilidad de la solución para las RENs, se ha llevado a cabo una validación funcional en la que se ha comprobado que la solución propuesta en esta tesis doctoral

satisface los requerimientos planteados por el equipo de operación de red de GÉANT, así como su capacidad para proporcionar el servicio en entornos multi-dominio. Adicionalmente, también se ha llevado a cabo una evaluación del rendimiento de la solución, para lo cual se han empleado distintas topologías de red, incluyendo la topología de red de la REN americana ESNet. Los experimentos demuestran que DynPaC satisface los requerimientos de las RENs, y que las estructuras de datos y algoritmos utilizados en la gestión de reservas avanzadas son capaces de realizar el proceso de admisión de las peticiones de servicio en el orden de los milisegundos.

Finalmente, la solución presentada en esta tesis ha sido implementada utilizando uno de los controladores SDN más importantes del mercado, el controlador de código abierto ONOS. Además, cabe destacar que DynPaC será utilizado en el despliegue piloto del nuevo servicio de BoD basado en SDNs de GÉANT a lo largo del 2017. Es más, el Open Networking Laboratory, a cargo del desarrollo del controlador ONOS, está planeando incluir algunas de las funcionalidades de DynPaC en próximos lanzamientos del dicho controlador.

Acknowledgements

Writing this PhD thesis has been, undoubtedly, the most challenging task I have ever faced. It would have been impossible without the help and support of my supervisors, colleagues, family and friends.

Seven years ago I ran into the office of an unknown professor to me at that time, Eduardo, which told me a crazy story about how the future of networking was software-defined. He introduced me to this challenging and amazing research topic, and motivated me to became the engineer I am today. For this, and for all his support throughout this years as my supervisor, I will be forever grateful. Notwithstanding, I have had the good fortune of having not one, but two amazing supervisors. For me, having Jasone on board has been the greatest experience. She has teach me how research must be done, and she has encouraged me to pursue my objectives and not to give up. Jasone, thank you very very much!!

In addition, during these years in the I2T Research group I have had the chance to work with a great group of people, and therefore, I would like to express my gratitude to them as well, specially to Mariví, for her advise and her inestimable help reviewing articles and to Jon, Jokin and Elías for being my friends in battle. But above all, thank you Igor, you saved me a lot of money on psychotherapy, but not so much in beers!. I would also like to thank to my colleagues in the GÉANT project and to Sonja Filiposka and Wolfgang John for their reviews.

Pero sin lugar a dudas, el mayor de mis agradecimientos se lo dedico a mi familia. A mi aitatxu, a mi amatxu y a mi hermano Iker, quienes me han apoyado y aguantado a lo largo de estos años, sin olvidarme de mis tías, tíos, primas y primos. También me gustaría aprovechar estas líneas para recordar a Fernando y a mi prima Marian, quienes por desgracia no estarán presentes para verme convertida en doctora. Azkenengoz, nire lagun minei Maitetxu, Ganore eta Maci, eta nire bizitza osoan zehar nirekin egon diren neskatilei: Nagore C., Nagore M., Itxa, Eidertxi eta Tatxu. Guztioi, MILA ESKER BIHOTZ BIHOTZEZ!

Contents

I INTRODUCTION 1

1 INTRODUCTION 3

1.1 Introduction . . . 3
1.2 Context . . . 4
1.2.1 Research and Education Networks . . . 5
1.2.2 Traffic Engineering in packet networks . . . 8
1.2.3 Software-Defined Networking . . . 17
1.3 Problem statement and motivation . . . 20
1.4 Objectives . . . 21
1.5 Contributions . . . 23
1.6 Content of the document . . . 24

II STATE OF THE ART 27

2 TRAFFIC ENGINEERING FRAMEWORKS IN SOFTWARE-DEFINED NETWORKS 29

2.1 Introduction . . . 30
2.1.1 Impact of the SDN interface type to TE . . . 30
2.1.2 Overview of the most relevant D-CPI Protocols . . . 34
2.2 TE frameworks for network resource utilization optimization 44
2.2.1 Bin *et al.* proposal . . . 45
2.2.2 B4 . . . 48
2.2.3 SWAN . . . 51
2.2.4 OSCARS . . . 54
2.2.5 OPEN . . . 57
2.2.6 OFVN . . . 59
2.2.7 IDEALIST . . . 62
2.3 TE frameworks for congestion minimization . . . 65
2.3.1 Huang *et al.* proposal . . . 66
2.3.2 Li *et al.* proposal . . . 68
2.3.3 BaatDaat . . . 71
2.3.4 MiceTrap . . . 74

2.4 TE frameworks for packet loss minimization 76
2.4.1 QNOX 77
2.4.2 Yoon *et al.* proposal 80
2.4.3 Phemius *et al.* proposal 82
2.5 TE frameworks for QoE maximization 85
2.5.1 Kassler *et al.* proposal 85
2.6 Summary of SDN-based TE frameworks 88
2.7 Conclusions 90
3 ALGORITHMS AND DATA STRUCTURES FOR ADVANCE RESERVATIONS 93
3.1 Introduction 94
3.1.1 Fundamentals of advance reservations 95
3.1.2 Algorithms and data structures 98
3.2 Resource management on a per link basis 100
3.2.1 Schelén *et al.* proposal 100
3.2.2 Guerin and Ariel proposal 105
3.2.3 Burchard proposal 108
3.2.4 Wang and Chen proposal 111
3.3 Alternative resource management approaches 115
3.3.1 Wolf and Steinmetz proposal 115
3.3.2 Andreica *et al.* proposal 120
3.3.3 Schneider and Linnert proposal 124
3.3.4 Balman *et al.* proposal 127
3.3.5 Barshan *et al.* proposal 129
3.4 Comparison of advance reservation solutions 132
3.5 Conclusions 134

III PROPOSAL 137
4 A GENERIC FRAMEWORK FOR TRAFFIC ENGINEERING IN SDN 139
4.1 Modules 139
4.1.1 Path computation element (PCE) 139
4.1.2 Topology abstractor 141
4.1.3 Resilience module 142
4.1.4 Monitoring module 143
4.1.5 Network programmer 143
4.1.6 Service manager 144

4.2 Interface towards external elements . . . 145
4.2.1 Traffic Analyzers . . . 146
4.2.2 Orchestrators . . . 147
5 SUPPORT FOR ADVANCE RESERVATIONS 153
5.1 Reservation model . . . 154
5.1.1 Definition . . . 154
5.1.2 Timing constraints . . . 154
5.2 Path computation algorithm . . . 157
5.2.1 Pre-computation phase . . . 157
5.2.2 On-demand phase . . . 160
5.3 Data structures for advance reservations in SDN . . . 162
5.3.1 Network Snapshot . . . 162
5.3.2 NSTree . . . 164
5.4 Operations supported by the NSTree . . . 179
5.4.1 Reservation of a new service . . . 179
5.4.2 Removal of a service . . . 185
5.4.3 Programmatic removal of a network snapshot . . . 187
6 OPTIMIZATION TECHNIQUES 191
6.1 Flow relocation . . . 191
6.2 Flow disaggregation . . . 195

IV VALIDATION 199
7 FUNCTIONAL VALIDATION 201
7.1 Suitability of the DynPaC framework for TE and optimization techniques . . . 201
7.2 Multi-domain E2E involving OpenFlow domains . . . 205
7.3 Multi-domain E2E involving heterogeneous domains . . . 209
7.4 GÉANT Requirements Tests . . . 213
7.4.1 Requirements . . . 213
7.4.2 Scenario . . . 215
7.4.3 Results . . . 216
7.4.4 Conclusions . . . 222
8 PERFORMANCE EVALUATION 223
8.1 Analysis of the impact of the number of alternative paths in the setup time and the resource utilization . . . 224
8.1.1 Full mesh networks . . . 225

8.1.2 ESNet network 235
8.2 Analysis of the impact of the number of links on the setup time 240
8.3 Analysis of the impact of the requested bandwidth on the setup time and the number of rejected services 242
8.4 Analysis of the impact of the aggregation policy 244

V CONCLUSIONS 247
9 CONCLUSIONS AND FUTURE WORK 249
9.1 Introduction 249
9.2 Contributions 250
9.3 Dissemination of the results 253
9.3.1 Publications in international journals 253
9.3.2 Publications in proceedings of international conferences 254
9.3.3 Oral communications 255
9.3.4 Other publications related to this PhD thesis . . . 256
9.3.5 Participation in projects 258
9.4 Future work and research lines 261

VI APPENDIX 263
A APPENDIX: RESERVATIONS GENERATOR SCRIPT 265

BIBLIOGRAPHY 273

List of Figures

Figure 1.1 GÉANT connectivity map. 5
Figure 1.2 SDN architecture proposed by the SDNRG [80]. . 18
Figure 1.3 SDN architecture proposed by the ONF [81]. . . . 19
Figure 2.1 Categorization of SDN protocols depending on the interface type. 31
Figure 2.2 D-CPI interface in the ONF architecture. 35
Figure 2.3 Main components of the ForCES architecture. . . 36
Figure 2.4 Main components of the OpenFlow switch. 38
Figure 2.5 Structure of the flow tables in an OpenFlow switch. 39
Figure 2.6 Main components of the PCE-based architecture. 43
Figure 2.7 Architecture proposed by Bin *et al.* [128]. 46
Figure 2.8 Google's B4 architecture [51]. 49
Figure 2.9 SWAN architecture [129]. 52
Figure 2.10 OSCARS architecture [131]. 55
Figure 2.11 Architecture of OPEN's control plane [134]. . . . 57
Figure 2.12 OFVN architecture [138]. 60
Figure 2.13 Architecture proposed by the FP7-IDEALIST project [142]. 63
Figure 2.14 Architecture proposed by Huang *et al.* [154]. . . . 67
Figure 2.15 Architecture proposed by Li *et al.*[155]. 69
Figure 2.16 BaatDaat architecture [156]. 72
Figure 2.17 MiceTrap architecture [157]. 74
Figure 2.18 QNOX architecture [165]. 78
Figure 2.19 Architecture proposed by Yoon *et al.* [169]. 81
Figure 2.20 Architecture proposed by Phemius *et al.* [170]. . . 83
Figure 2.21 Architecture proposed by Kassler *et al.* [50]. . . . 86
Figure 3.1 Structure of the binary search tree over time proposed by Schelén *et al.* [194]. 102
Figure 3.2 Structure of the segment tree proposed by Schelén *et al.* [194]. 103

Figure 3.3 Data structure proposed by Guerin and Ariel [196]. 106
Figure 3.4 Array proposed by Burchard *et al.* [180]. 109
Figure 3.5 Modifications to the segment tree introduced by Burchard *et al.* [180]. 110
Figure 3.6 Structure of the Bandwidth Tree [199]. 112
Figure 3.7 Architecture of the ReRA system proposed by Wolf and Steinmetz [202]. 116
Figure 3.8 Data structure for time management proposed by Wolf and Steinmetz [202]. 118
Figure 3.9 Data structure for reservations management proposed by Wolf and Steinmetz [202]. 119
Figure 3.10 List of free blocks proposed by Schneider and Linnert [206]. 125
Figure 3.11 Linked list proposed by Balman *et al.* [207]. . . . 128
Figure 3.12 Architecture proposed by Barshan *et al.* [208]. . . 130
Figure 4.1 Architecture of a generic framework for advance path computation in SDN. 140
Figure 4.2 Integration of the DynPaC framework with a traffic analyzer. 146
Figure 4.3 Integration of the DynPaC framework as NRM for multi-domain BoD service provisioning. 148
Figure 4.4 DynPaC operational modes. 149
Figure 5.1 Service is accepted by the DynPaC framework. . 155
Figure 5.2 Service A is rejected by the DynPaC framework due to a lack of resources. 155
Figure 5.3 Service B is rejected by DynPaC because it is blocked for new requests. 156
Figure 5.4 Service A is rejected by DynPaC because the book-ahead interval is not respected. 156
Figure 5.5 Ingress and egress nodes facing users or other domains in a network. 159
Figure 5.6 Selection of paths of less than n_{MAX} hops. 159
Figure 5.7 Structure to store the pre-computed paths. 161
Figure 5.8 Network resource consumption evolution represented with network snapshosts. 164
Figure 5.9 Design of the NSTree data structure. 165

Figure 5.10 Node generation process case 0: no overlapping. . 170
Figure 5.11 Node generation process case 1: back overlapping. 171
Figure 5.12 Node generation process case 2: partial overlapping.172
Figure 5.13 Node generation process case 3: front overlapping. 172
Figure 5.14 Node generation process case 4: full overlapping. . 173
Figure 5.15 Rearrangement Case A: Two consecutive NS aggregated into the same CA 176
Figure 5.16 Rearrangement Case B: Two consecutive NS children of the root node. 176
Figure 5.17 Rearrangement Case C: Predecessor Network Snapshot aggregated into a Common Ancestor. . 177
Figure 5.18 Rearrangement Case D: New node aggregated into a Common Ancestor. 177
Figure 5.19 Rearrangement Case E: Consecutive nodes aggregated into different Common Ancestors. 178
Figure 5.20 Creation of a CA from a NS with root parent. . . 184
Figure 5.21 Creation of a CA from a NS with CA parent. . . 184
Figure 6.1 Flow relocation. 192
Figure 6.2 Flow disaggregation. 196
Figure 7.1 Implementation of the DynPaC framework with ODP. 203
Figure 7.2 Topology used for the EWSDN 2015 demonstration.203
Figure 7.3 Video of the EWSDN 2015 demonstration. 204
Figure 7.4 Demo scenario for multi-domain OpenFlow based on NSI CONTEST and DynPaC results. 206
Figure 7.5 Modules of a NRM for OpenFlow domains based on DynPaC. 207
Figure 7.6 Video of the Supercomputing 2014 demonstration. 208
Figure 7.7 Implementation of the DynPaC framework with ONOS. 210
Figure 7.8 Scenario for multi-domain BoD service provisioning involving three heterogeneous domains. 211
Figure 7.9 Video of the NetSOFT 2016 demonstration. . . . 213
Figure 7.10 Cambridge laboratory deployment for functional validation. 216

Figure 8.1 Effect of the path diversity in a 5 node full mesh network. 227
Figure 8.2 Effect of the path diversity in a 6 node full mesh network. 228
Figure 8.3 Effect of the path diversity in a 7 node full mesh network. 229
Figure 8.4 Effect of the path diversity in a 8 node full mesh network. 230
Figure 8.5 Overall network resource utilization for a 8 node full mesh topology. 233
Figure 8.6 Path pre-computation time. 233
Figure 8.7 Topology of the ESNet network. 235
Figure 8.8 Effect of the path diversity in the ESNet network. 236
Figure 8.9 Effect of the path diversity in the ESNet network. 237
Figure 8.10 Overall network resource utilization in the ESNet topology. 239
Figure 8.11 Impact of the number of nodes in the setup time. 240
Figure 8.12 Effect of the requested bandwidth. 243
Figure 8.13 Effect of the value of delta. 246

List of Tables

Table 2.1 Summary of the impact of SDN protocols to TE. 34
Table 2.2 Summary of SDN-based TE frameworks 89
Table 3.1 Comparison of advance reservation solutions . . . 133
Table 7.1 Requirements for an SDN-based NRM. 214
Table 7.2 Summary of the functional validation tests. . . . 217
Table 7.3 Test 1 - DynPaC topology abstraction. 217
Table 7.4 Test 2 - Programmatic VLAN circuit establishment.218
Table 7.5 Test 3 - Programmatic VLAN circuit establishment with VLAN translation. 219
Table 7.6 Test 4 - Rate limiting. 220
Table 7.7 Test 5 - Port status notification. 221
Table 7.8 Test 6 - Interface towards NSA. 221
Table 7.9 Test 7 - STP discovery. 222
Table 8.1 Summary of the results obtained for the full mesh topologies. 231
Table 8.2 Summary of the results obtained for the ESNet topology. 238

Acronyms

A-CPI Application-Controller Plane Interface

ABNO Application-Based Network Operations

ALTO Application Layer Traffic Optimization

ANM Adaptive Network Management

aNode Analyzed NSTree Node

ATD Address Translation Database

aTime Analyzed Time Slot

BDDP Broadcast Domain Discovery Protocol

BFD Bidirectional Forwarding Detection

BFS Breadth First Search

BGP Border Gateway Protocol

BGP-LS Border Gateway Protocol - Link State

BoD Bandwidth on Demand

C-SPF Constrained-Shortest Path First

CA Common Ancestor

CE Control Elements

CKE Cognitive Knowledge Element

cPCE Child Path Computation Element

D-CPI Data-Controller Plane Interface

DABP Deadline And Budget Priority based on Deadline Budged Constraint

DARA Dynamic Advance Reservation Algorithm

DC Data Center

DFS Depth First Search

DynPaC Dynamic Path Computation

ECMP Equal-Cost Multi-Path

EON Elastic Optical Network

ERO Explicit Route Objects

ESNet Energy Sciences Network

eT End Time

ETSI European Telecommunications Standards Institute

FE Forwarding Elements

ForCES Forwarding and Control Element Separation

FPGA Field-Programmable Gate Array

GMPLS Generalized Multi-Protocol Label Switching

GTS GÉANT Testbed Services

H-PCE Hierarchical Path Computation Element

I2RS Interface to the Routing System

IETF Internet Engineering Task Force

IoT Internet of Things

LDP Label Distribution Protocol

LFB Logical Forwarding Blocks

LLDP Link Layer Discovery Protocol

LP Linear Programming

LSDB Link Status Database

LSP Label Switched Path

MDP Media Degradation Path

ME Management Entity

MI Management Interface

MPLS Multi-Protocol Label Switching

MPLS-TE Multi-Protocol Label Switching - Traffic Engineering

MPTCP Multi-Path TCP

NA Not Applicable

NaaS Network as a Service

NE Network Element

NETCONF Network Configuration Protocol

NFV Network Functions Virtualization

NOS Network Operating System

NREN National Research and Education Networks

NRM Network Resource Manager

NS Network Snapshot

NSA Network Services Agent

NSF Network Services Framework

NSI-CS Network Services Interface - Connectivity Service

NSI Network Services Interface

NSP Network Service Provider

NSTree Network Snapshots Tree

NVGRE Generic Routing for Encapsulation Network Virtualization

RWA Routing and Wavelength Assignment

PAF Path Assignment Function

PCC Path Computation Client

PCE Path Computation Element

PCEL Physical Control Elements

PCEP Path Computation Element Communication Protocol

PCM Path Computation Module

PFE Physical Forwarding Elements

pNode Predecessor NSTree Node

pPCE Parent Path Computation Element

PPM Path Provision Module

PredNS Predecessor Network Snapshot

PSS Path Setup Subsystem

ODP Open Daylight Project

OF-CONFIG OpenFlow Management and Configuration Protocol

OFVN OpenFlow-based Virtualization-aware Networking

OLS OpenFlow Logical Switch

ONE Open Network Environment

ONF Open Networking Foundation

ONOS Open Network Operating System

OPEN Open Programmable Extensible Networks

OSCARS On-Demand Secure Circuits and Advance Reservation System

OSPF Open Shortest Path First

OVS Open vSwitch

OVSDB Open vSwitch Database Management Protocol

OXM OpenFlow eXtensible Match

QMOF QoS Matching and Optimisation Function

QNOX QoS-aware Network Operating System

QoE Quality of Experience

REN Research and Education Networks

ReRA Resource Reservation in Advance

REST Representational State Transfer

RFC Request For Comments

RMS Resource Management System

RST Remaining Service Time

RSVP Resource Reservation Protocol

RSVP-TE Resource Reservation Protocol - Traffic Engineer

SAR Service Acceptance Ratio

SARA Static Advance Reservation Algorithm

SDN Software-Defined Networking

SDNRG Software-Defined Networking Research Group

SDO Standards Development Organizations

SE Service Element

SLA Service Level Agreement

SNMP Simple Network Management Protocol

SPF Shortest Path First

SPN Second Pointer Node

sT Start Time

STP Service Termination Point

STSD Specified Time Specified Duration

STUD Specified Time Unspecified Duration

SuccNS Successor Network Snapshot

SWAN Software-Driven WAN

TE Traffic Engineering

TED Traffic Engineering Database

ToR Top of Rack

TRL Technology Readiness Level

UTSD Unspecified Time Specified Duration

UTUD Unspecified Time Unspecified Duration

VM Virtual Machine

VoIP Voice over IP

VPN Virtual Private Networks

VXLAN Virtual eXtensible Local Area Network

WDM Wavelength Division Multiplexing

Part I

INTRODUCTION

1 Introduction

> "Computer science research is different from these more traditional disciplines. Philosophically it differs from the physical sciences because it seeks not to discover, explain, or exploit the natural world, but instead to study the properties of machines of human creation."
>
> — Dennis Ritchie

1.1 INTRODUCTION

During the last decade, Software-Defined Networking (SDN) has emerged as a revolutionary networking paradigm, and has gained the attention of both the industry and the academia. Actually, SDN has been included in several reports as one of the most disruptive and interesting technologies in the networking area [1, 2]. Several factors have been decisive for the success of SDN. On the one hand, the vast range of SDN-enabled networking devices available in the market has been primordial. Both classical manufacturers such as Cisco [3], HP [4] or NEC [5] and novel manufacturers such as Corsa [6] are commercializing SDN products. On the other hand, the availability of open-source controllers with OpenFlow support [7], the *de facto* protocol for SDN, has fostered the implementation of SDN applications.

For the moment, the Standards Development Organizations (SDO)s, Network Service Provider (NSP)s and vendors involved in this new

market opportunity are working on a unified definition of SDNs. Most definitions agree on the availability of open programmable interfaces at networking devices, the separation of the control and forwarding planes, and the existence of a logically centralized controller. Nevertheless, most agents involved in the standardization of SDN do agree on some possible applications. In addition to its utilization in Data Center (DC) networks [8], campus networks [9, 10] and as an enabler for Network Functions Virtualization (NFV) [11], SDN appears as a promising candidate to enhance current Traffic Engineering (TE).

TE has always been one of the most challenging topics in communication networks [12]. As stated by the Internet Engineering Task Force (IETF), TE deals with the performance optimization of operational networks, and plays a key role in the provisioning of services with Quality of Service (QoS) [13]. Being aware of the benefits that SDN can bring to TE, telecom companies such as Telefonica and AT&T have started to work on SDN-based solutions [14]. Similarly, and given the relevance that TE has on the provisioning of connectivity services inside the research and academic community, Research and Education Networks (REN) have also started to analyze the applicability of SDN to their transport networks.

In such a context, Section 1.2 introduces RENs and the services typically provided by them, making special emphasis on the need to support advance reservations in order to offer the Bandwidth on Demand (BoD) service. It also presents the fundamentals of TE in packet networks, including the most common TE performance objectives and techniques, its evolution and its current limitations. In addition, the fundamentals of SDN are described, where the architectures proposed by two different SDOs are introduced. Section 1.3 identifies the problem and the motivation behind this PhD thesis, while Section 1.4 presents the objectives and Section 1.5 summarizes the contributions. Finally, Section 1.6 provides a detailed overview of the content of this document.

1.2 CONTEXT

It is indubitable that research and innovation have a huge impact on the development of our society. Aware of this, most countries, to the extent

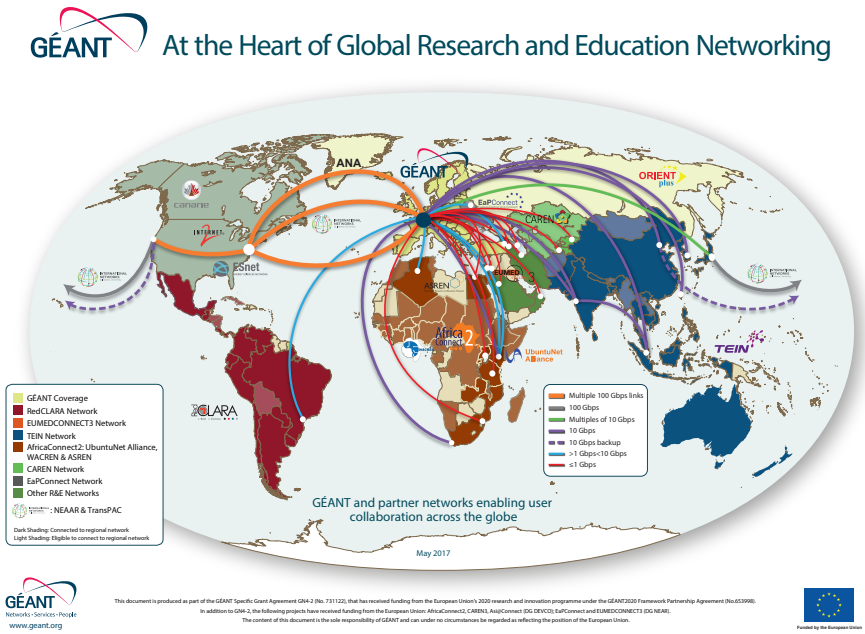


Figure 1.1.: GÉANT connectivity map.

of their possibilities, invest on research and development projects in order to increase their competitiveness.

The magnitude of the research questions trying to be answered today require multidisciplinary teams, which results in hundreds of researchers and innovators distributed across the world working together. Hence, it is necessary to facilitate the communication between the team members, and provide efficient ways to access the data necessary to conduct the experiments.

With the advent of the Internet and the huge advancements in communications since its appearance, collaborative research has become a reality. Furthermore, in order to satisfy the connectivity needs of researchers, most countries have created National Research and Education Networks (NREN).

1.2.1 RESEARCH AND EDUCATION NETWORKS

In an effort to satisfy the connectivity requirements of their research community, most developed countries have deployed their own NREN.

Examples of such networks are Internet2 [15] or the Energy Sciences Network (ESNet) [16] in the United States of America, Canarie [17] in Canada, Kreonet [18] in South Korea, Sinet [19] in Japan or RedIRIS [20] in Spain. Moreover, the European Community considers the interconnection of the European NRENs a high priority subject. As a result, it finances GÉANT [21], the pan-European REN that interconnects the European NRENs and peers them with non-European NRENs, as illustrated in Figure 1.1.

The portfolio of this kind of network usually considers the following connectivity services to satisfy the needs of the academic institutions [22]:

- Provide private high-bandwidth Internet Protocol (IP) connectivity separated from the general-purpose access to the Internet.
- Provide dedicated point-to-point connectivity circuits. For instance, GÉANT offers dedicated Ethernet circuits, dedicated wavelengths and BoD, which will be later explained in this section.
- Facilitate peerings among RENs and with external commercial partners, such as Amazon or Microsoft, for example, to enable rapid access to cloud services.
- Provide single-domain and multi-domain Virtual Private Networks (VPN).
- Enable networking-related research through distributed testbeds.

Notwithstanding, the provisioning of such kind of service usually encompasses high operational costs, especially in those services where a Service Level Agreement (SLA) must be ensured and a certain QoS must be provided. An example of such kind of service is BoD, which is described in the following subsection.

1.2.1.1 BANDWIDTH ON DEMAND

A service typically provided by the RENs to satisfy the increasing demand of users with short-term, high-capacity and high-availability demands is BoD. For instance, the Energy Sciences Network (ESnet) [16], the high-speed computer network serving the United States Department of Energy,

provides BoD through the On-Demand Secure Circuits and Advance Reservation System (OSCARS) [23]. Likewise, GÉANT offers BoD to their users through the AUTOBAHN [24] provisioning tool.

In essence, BoD allows to establish end-to-end (E2E) dedicated circuits with a guaranteed bandwidth during an specific period of time. As a consequence, the provisioning of this service requires the reservation of the network resources in advance, in order to ensure that the users will obtain the requested resources as expected.

Resource reservation has been a widely studied topic in communication networks [25–33]. Initial resource reservation systems were focused on *immediate reservations*, where the resources were exploited immediately after their request and for an unspecified period of time [34, 35]. Notwithstanding, the appearance of new scenarios in which the resource reservation was necessary for a specific period of time, such as a video-conferencing systems [36], video-on-demand [37] or massive data transfers in the grid environment [38], introduced the concept of *advance reservations*.

As opposed to immediate reservations, in advance reservations the resources are exploited at some time in the future and during a specific period of time. One of the main benefits of relying in advance reservation mechanisms is the optimization of the network resources utilization [39]. This is a consequence of introducing time constraints on the resource reservation. Since resources are reserved for a specific period of time, this approach allows to re-utilize the same resources for future reservations that do not overlap in time.

First proposals for advance reservations were focused on scenarios where the network resources were rather limited. As stated by Degermark *et al.*, *"where resources are plentiful, not even immediate reservations may be necessary, but where resources are scarce enough to justify reservations at all, it makes sense to be able to make them in advance."* (in [40], page 4). Although today's networks are usually over-provisioned to provide QoS, advance reservations can still make a difference. For instance, even if redundant network devices are necessary to ensure service provisioning, the redundancy ratio could be reduced by utilizing the network resources in a more efficient way. Moreover, a better utilization of the network

resources could enable the network to be able to cope with the growing traffic demands during a longer period of time.

In recent proposals [41, 42], advance reservation mechanisms have been widely applied to Wavelength Division Multiplexing (WDM) optical networks [43]. In WDM optical networks, when a circuit is requested, it is necessary to assign a route and a wavelength, which is known as the Routing and Wavelength Assignment (RWA) problem [44]. A comprehensive survey were the advance reservation mechanisms to address this problem are analyzed is available in [45]. Moreover, the use of advance reservations mechanisms has also become very popular to optimize the bandwidth consumption [46] and minimize the network congestion [47] in a variety of scenarios.

All in all, advance reservation solutions, including the ones utilized to provide BoD, require the utilization of TE strategies, in order to ensure an efficient utilization of the network resources over time. As a consequence, the following section introduces TE in packet networks.

1.2.2 TRAFFIC ENGINEERING IN PACKET NETWORKS

This section introduces TE in packet networks and presents a list of common TE performance objectives and the techniques used to achieve them. Moreover, a brief overview of the evolution of TE in packet networks is included.

1.2.2.1 DEFINITION OF TE

In communication networks, TE consists of the application of strategies and scientific principles to optimize the performance of operational networks [13]. The general objective of TE is to route traffic in a data network so that traffic demands are met, by optimizing a selected performance objective. This usually involves the computation of a path between a given source-destination pair, or the computation of multiple paths to share the load according to specific traffic-splitting ratios.

1.2.2.2 TE PERFORMANCE OBJECTIVES AND TECHNIQUES

The performance optimization of a network is an iterative process in which new technologies and optimization mechanisms are continuously required

[12]. When a TE solution is designed, the performance objective must be selected carefully, since different performance objectives can be mutually exclusive. This section presents a comprehensive list of performance objectives and the techniques that are used for their optimization.

- *Congestion minimization*

In an operational IP context, congestion is one of the most important problems, since it affects delay, jitter and packet loss [12]. Therefore, it is one of the most critical performance objectives in current communication networks.

Congestion minimization can be achieved using different techniques:

- Sharing the network resources by multiple traffic streams.
- Re-allocating network resources by redistributing the traffic over the infrastructure.
- Denying the access to congested resources. Once the congestion is detected, the TE system can only assign not congested resources to new demands.

Sharing the network resources by multiple traffic streams is of special relevance for the congestion minimization, since it is a proactive technique aiming to avoid congestion. This is often achieved by minimizing the links' utilization solving a traditional optimization problem known as the minimum cost multi-commodity flow problem [48]. This optimization problem has been widely studied in the literature [49–52] and it is an ongoing research work. The main purpose of this approach is to balance the traffic load along the network, which results in a better network utilization. This is achieved by splitting the traffic into a set of streams that are routed through multiple paths connecting the ingress-egress router pair. As a consequence, the load is balanced among a higher number of network resources, resulting in a smaller amount of packets queued at the forwarding devices and less occupied bandwidth at the links.

A common approach to solve the multi-commodity flow problem is the computation of an optimal splitting ratio for the incoming traffic demand and the paths to transfer the resulting sub-flows. Traffic splitting can be

achieved in different ways. On the one hand, the simplest mechanism to split the traffic is on a per-packet basis, for example in a round-robin fashion. On the other hand, it is also possible to split the traffic on a per-flow basis, by applying a hash function over a set of the packets' header fields. Current commercial routers can be configured to divide traffic based on the result of hashing different TCP/IP header fields.

The limitations of these two approaches are later explained in Section 1.2.2.4. Notwithstanding, it is worth mentioning that in addition to these two approaches, traffic splitting can also be achieved by forwarding the traffic using a richer combination of header fields, without the need of applying hash functions. For example, this is possible in OpenFlow devices, where the flows can be defined using a combination of header fields from Layer 1 (L1) to Layer 4 (L4), as it will be explained in Chapter II. And this is precisely the kind of approach that novel TE strategies can exploit in SDN.

As mentioned before, congestion can result in a higher E2E delay and packet loss. In other words, congestion minimization can be considered a general performance objective that has a direct impact on more specific performance objectives such as the E2E delay and packet loss minimization. Therefore, the techniques used to minimize the congestion are also useful to minimize these two performance parameters. Notwithstanding, there are other factors besides congestion that can be the root cause of the E2E delay and the packet loss, requiring the utilization of more specific techniques for their optimization. As a consequence, the following subsections present other techniques to deal with the minimization of the E2E delay and the packet loss, where they are considered independent performance objectives.

- *E2E delay minimization*

A typical network-related performance objective that impacts QoS and Quality of Experience (QoE). The minimization of the E2E delay is essential for critical real-time communications. It can be applied on a per-flow basis or as an overall objective that takes into account the E2E delay of all the packets transmitted in the network. One of the most common techniques to minimize the E2E delay is Constrained-Shortest

Path First (C-SPF), where the E2E delay is used as a constraint for the path selection [53].

- *Packet loss minimization*

Another typical network-related performance objective that can also be evaluated per-flow or network-wide. Besides congestion, packet loss can also be the result of failures in the network, such as forwarding devices and links, requiring additional techniques to increase the failure recovery capabilities of the network. This performance objective is usually tackled by over-provisioning the network to increase resilience [54] by means of redundant resources to be used in case of failure. In fact, if multiple paths are available to convey traffic between a given source-destination pair, traffic can be re-routed among the available paths when one of them suffers a disruption.

- *Energy consumption minimization*

This is a performance objective that does not necessarily match a network performance parameter. It is widely used in the scope of *green computing* [55], which aims to lower the environmental impact of Information and Communication Technologies (ICT). This performance objective is usually optimized either by adapting the rate of network operation to the offered workload or by reducing the amount of active resources [56]. In this last case, the energy consumption is reduced when traffic is gathered into a few paths and unused line cards can be powered down in the network equipment. This is a good example of how the different performance objectives can be mutually exclusive, since the minimization of the energy consumption and the congestion minimization cannot be achieved at the same time when this approach is followed.

- *QoE maximization*

The QoE, as defined by the European Telecommunications Standards Institute (ETSI), is a parameter that measures the performance of using an ICT service or product taking into account objective technical

parameters, like QoS, and subjective psychological parameters [57]. In other words, it is a parameter that gets affected by all the elements involved in the E2E transmission, including the end devices, environmental factors such as the light and the network performance. Therefore, the QoE maximization also requires the optimization of the network performance, which is inside our scope of interest. Notwithstanding, maximizing the QoE does not always imply the maximization of the network throughput, and a correlation between the QoE criteria and the network-related performance parameters needs to be defined, as argued in [50].

- *Resource utilization optimization*

The optimization of the resource utilization is another performance objective. For example, computation, buffer space and bandwidth are resources that need to be efficiently used, since they can impact congestion and other parameters. In addition, a good utilization of the resources helps network operators to serve a higher number of service demands without increasing their costs. That is, a good utilization of the network resources allows network operators to allocate a higher amount of traffic using the same network resources.

A common approach to optimize the network resource utilization is to schedule well characterized data transfers. For instance, network operators can decide to transmit backup traffic between various data centers during the night hours, since more resources are available at that time.

As mentioned before, advance reservation systems [58] are also used for the provisioning of the BoD service, since they are meant to optimize the bandwidth utilization. Advance reservation systems allow to maintain a detailed inventory of the resource consumption over time and a better assignment of resources to satisfy new demands.

1.2.2.3 EVOLUTION OF TE

According to Awduche *et al.* [59], TE is considered a control issue where the element in charge of TE acts as a controller in an adaptive feedback control system. In this schema, available control actions must include

the modification of traffic management parameters, the modification of parameters associated with routing and the modification of the attributes and constraints associated with resources. Over the years, TE in packet networks has been tackled using different approaches, as mentioned in the Request For Comments (RFC) 3272 [12]. However, first proposals were not appropriate for TE because they did not satisfy the aforementioned requirements posed in [59].

First routing protocols in the ARPANET were highly scalable and resilient distributed protocols but without the flexibility required by TE [60]. When the Internet became a reality, the adaptive routing protocols used in ARPANET were substituted by dynamic routing protocols. Though, the Interior Gateway Protocols (IGP) that run on the Internet were neither appropriate for TE, since the route selection was based on shortest path algorithms fed with additive link metrics and not on the resources available in the network.

As a first approach to take advantage of TE strategies in the Internet, overlay models were used, like IP over ATM [61]. By means of a secondary technology capable of establishing virtual circuits, point-to-point links between IP routers were served. This way, arbitrary virtual topologies were defined and superimposed onto the physical network topology that resulted in a much easier TE operation. Nevertheless, the use of overlay technologies increased the overall complexity of the network operation. In addition, these strategies were usually based on circuit pre-provisioning, given the lack of efficient mechanisms to create new circuits on demand.

Parallel in time, the Nimrod routing architecture was designed to provide service-specific routing taking into account multiple constraints [62]. Nimrod was based on the distribution of link-state maps that abstracted network connectivity and services information, and introduced the concept of explicit routing to allow the selection of paths at originating nodes. Even if this protocol was never deployed in the public Internet, it introduced some interesting concepts adopted in more recent proposals, like explicit routing.

In the next iteration, Shortest Path First (SPF) algorithms that took into account the requested Type of Service (ToS) were proposed [63]. These approaches lead to an unfair usage of the network resources, where the shortest paths end up congested and other paths remain underutilized.

Next, traffic splitting was introduced by means of the Equal-Cost Multi-Path (ECMP), where traffic was split equally among all the available shortest paths [64]. Although the utilization of traffic splitting mechanisms is not always optimal, as it will be further explained in Section 1.2.2.4, the use of [64] is very extended, and many manufacturers support this protocol in their networking devices.

Later, the Multi-Protocol Label Switching (MPLS) forwarding architecture emerged to provide flexibility and to increase the performance and scalability of the network layer routing [65]. In MPLS, packets are transmitted between the edge nodes of an MPLS domain using Label Switched Path (LSP) and the forwarding decisions at each node are done based on previously assigned *labels*. This results in a higher network performance, since the forwarding decision is performed using a single header field. MPLS is useful for TE because it provides most of the functionalities available from the overlay model in an integrated manner and at a lower level [13]. It supports the creation of explicit LSPs that are not constrained by the destination-based forwarding paradigm, facilitating the multipath routing. Furthermore, MPLS is appropriate for TE because it supports explicit routing and allows traffic aggregation and disaggregation, while the classical destination-based IP forwarding only supports aggregation based on IP subnetting. In addition, with MPLS it is relatively easy to integrate constraint-based routing frameworks.

Finally, the IETF proposed the Path Computation Element (PCE)-based architecture for MPLS and Generalized Multi-Protocol Label Switching (GMPLS) networks [66], which extends packet switching capabilities of MPLS to an open set of networking and switching methods. The PCE-based architecture proposed a dedicated element to be in charge of the path computation making possible the application of complex algorithms such as C-SPF. In addition, it supports the instantiation of point-to-point and point-to-multipoint LSPs, which is known as explicit routing. This architecture can be used when the path computation is CPU-intensive or when there is no visibility of all the network elements involved. As a consequence, this architecture is being adopted for TE [67], and can be used in intra-domain, inter-domain and inter-layer contexts.

In summary, although TE has greatly evolved since the appearance of first communication networks, it is still a challenging topic.

Notwithstanding, the appearance of SDN arises the question of how the novelties introduced by this new paradigm can boost the next generation of TE solutions.

1.2.2.4 LIMITATIONS OF CURRENT TE SOLUTIONS

Although TE solutions have greatly evolved during the last years, they still present some limitations, which are described below:

- *Unrealistic traffic splitting ratios*

Congestion minimization is often achieved using multiple paths, but the current mechanisms to split the traffic present some limitations.

On the one hand, per-packet traffic splitting results in an excessive packet reordering in the destination end-point, which is undesirable, especially for Transmission Control Protocol (TCP) applications. As explained in [68], packet-level multipath routing can entail TCP segments arriving out of order to the destination entity, triggering the TCP congestion avoidance mechanism unnecessarily and resulting in the application throughput and the whole network performance being degraded. In addition, jitter can occur, requiring large buffers to temporarily store the packets received out of order.

On the other hand, per-flow traffic splitting allows individual TCP or User Datagram Protocol (UDP) flows to be distinguished, avoiding the traffic reordering problem. Nevertheless, the traffic splitting granularity is determined by the forwarding element and the hash function that is used to split the traffic. This granularity does not necessarily need to be the same granularity demanded by the TE solution, resulting in the assignment of inappropriate traffic ratios to each path. As a result, the overall network performance and the capacity of the TE mechanism to deal with congestion may not be optimal.

In addition, traffic splitting mechanisms, such as the one utilized in ECMP, where the traffic is split among paths of equal cost [64], may not take into account the potential congestion of the shortest paths used to balance the traffic, resulting in a poor performance [69].

- *Suboptimal path computation algorithms*

Path computation and the required resource handling in Multi-Protocol Label Switching - Traffic Engineering (MPLS-TE) present some limitations as well. In [70], the authors detected that some of the links in an over-provisioned network were experiencing some latency. They analyzed their MPLS-TE solution and deducted that latency inflation was a consequence of both the C-SPF algorithm that they were using and the continuous path re-computations that occur as a consequence of the *autobandwidth* algorithm, which is provided by many MPLS vendors to automatically adjust the reserved bandwidth of the LSPs depending on the traffic demand.

- *TE databases do not reflect the network state in real-time*

Although the PCE-based architecture can improve some of the limitations present in classic MPLS-TE, it also presents some limitations of its own. In the PCE-based architecture, path computation is done using the TE information stored in the Traffic Engineering Database (TED). This database holds an inventory of the resources available in the network, information that is used by the PCE to compute the paths. Notwithstanding, according to the RFC 4655 [66], the TED does not always reflect the network state in real-time. When the TED is not properly synchronized with the network state, which can occur at specific times, the rate of wrong computed paths may increase.

- *Long convergence times of distributed protocols*

Another important limitation that can be found in MPLS-TE and in the PCE-based architecture is their dependence on Resource Reservation Protocol - Traffic Engineer (RSVP-TE). When a network device requests a path, the PCE replies with the computed path information. Then, the network device uses the distributed protocol RSVP-TE to inform the other nodes. As a consequence, the establishment of the path depends on the time to propagate the new path configuration to all the network devices. This has a direct impact on the network stability, since the

already established data flows may not be aware of the new situation immediately. This fact also affects scalability as defined by the RFC 4655, because RSVP-TE is an in-band signaling protocol. This is a limitation that most MPLS-TE based solutions present, as they all rely on RSVP-TE.

1.2.3 SOFTWARE-DEFINED NETWORKING

Since SDN is considered as an enabler for future TE solution, this section presents the fundamentals of SDN, including a brief introduction to the history of SDN, its definition and the proposed architectures.

1.2.3.1 FUNDAMENTALS OF SDN

SDN is the result of three key research areas very popular since their inception in the mid-90s [71]. First, proposals like Open Signaling [72] and Active Networking [73] pushed in favor of *network programmability* by means of open interfaces and code piggybacked inside the user messages respectively. Second, the *control and data plane separation* brought to the fore the possibility to control the network from an external entity [74] and the transition towards a logically centralized control plane [75]. Moreover, the IETF boosted the standardization of the Forwarding and Control Element Separation (ForCES) framework, according to many the first serious precursor of SDN. Finally, the appearance of *network operating systems* and the clean slate approach proposed at the 4D [76] project led to the release of the OpenFlow switch specification and the SDN revolution.

According to the Open Networking Foundation (ONF) [77], the changing traffic patterns within an enterprise DC, the need to accommodate the traffic of new personal devices in a fine-grained manner, the rise of cloud services and the associated increasing demand for network capacity are key computing and communication trends that require a new network paradigm such as SDN.

The ONF defines SDN as an emerging network architecture where the network is directly programmable and where the control and forwarding planes are decoupled. One of the main characteristics of SDN is that the intelligence is logically centralized in SDN controllers. Such controllers maintain a global view of the network, which results in the network

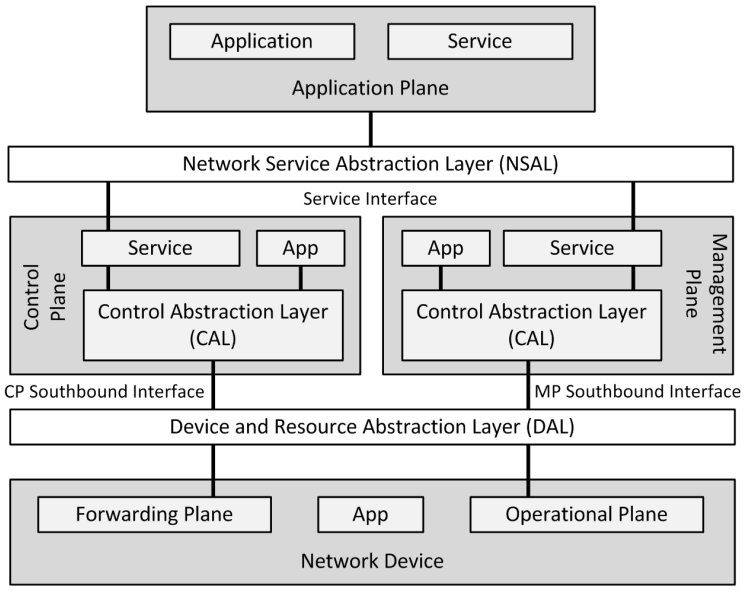


Figure 1.2.: SDN architecture proposed by the SDNRG [80].

appearing to the applications and policy engines as a single, logical switch [78].

With SDN, network design and operation are simplified because the entire network can be controlled from a logically centralized point using open interfaces. Network control becomes vendor-independent and the utilization of simpler network devices is a real possibility, since the devices only need to understand the SDN technology that controls them. One of the main features of this new paradigm is that networks can be programmatically configured, making possible the management of the entire network through intelligent orchestration and provisioning systems. Besides, SDN architectures support a set of Application Programming Interfaces (API) that enable the implementation of common network services, custom tailored to meet business objectives.

Nowadays, not only the ONF but many SDO are dealing with SDN. For instance, the IETF has created a research group focused on this trend, named the Software-Defined Networking Research Group (SDNRG) [79]. Both the ONF and the SDNRG have proposed different SDN architectures, which are described in this section.

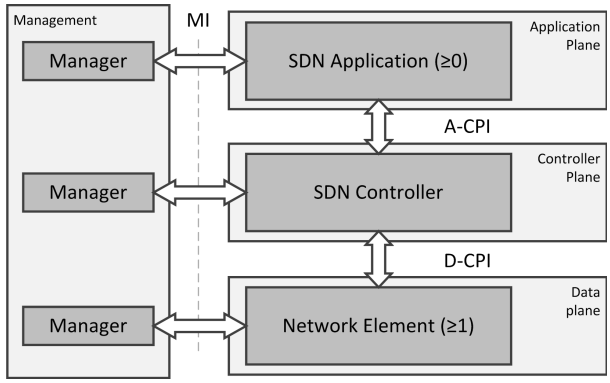


Figure 1.3.: SDN architecture proposed by the ONF [81].

- *SDNRG Architecture*

The architecture proposed by the SDNRG [80] is depicted in Figure 1.2, and defines five different planes. Inside the network device, the *forwarding plane* is the one responsible for handling packets in the data path and it is often referred to as the data plane. Secondly, the *operational plane* is the plane responsible for managing the operational state of the network. Outside the network devices, in an external entity, the *control plane* is the one in charge of taking the decisions about how packets are forwarded at network devices, and it is also in charge of pushing such decisions down to network devices so that they are executed. On the other hand, the *management plane* is the one in charge of monitoring, configuring and maintaining the network devices. Finally, the *application plane*, also located in an external entity, is where the applications that rely on the network to provide services for the end users and processes reside.

In addition to these five planes, the SDNRG architecture also defines two abstraction layers: the *Device and Resource Abstraction Layer* and the *Network Service Abstraction Layer*. The first one abstracts the network devices' forwarding and operational planes and connects to the control plane and management plane through the *Control Plane Southbound Interface* and the *Management Plane Southbound Interface* respectively. The second abstraction layer exposes the control and management planes through a *Northbound Interface* to the application plane.

- *ONF Architecture*

The ONF has also presented a reference architecture for SDN [81], which follows a three layers approach, as depicted in Figure 1.3. It must be taken into account that the main goal of this architecture is to provide a high level overview of the reference points and open interfaces that should be present in every SDN deployment, in order to guarantee a minimum set of capabilities that would allow to control the connectivity provided by the network resources and the traffic flows through them.

The first layer is known as the *data plane* and it is the plane in which the network elements reside. The data plane uses the *Data-Controller Plane Interface (D-CPI)* to expose the network elements' capabilities to the second layer, that is, the *controller plane*. As its name suggests, the controller plane contains the SDN controller, which is the element in charge of controlling the network elements through the previously mentioned D-CPI. The controller plane also exposes services to the third layer, known as the *application layer*, through the *Application-Controller Plane Interface (A-CPI)*. This latter plane holds the applications that specify the behavior of the network through the A-CPI. In addition to these two interfaces, this architecture also considers a Management Interface (MI) to configure and manage the three different planes.

1.3 PROBLEM STATEMENT AND MOTIVATION

As mentioned before, research and education networks demand TE solutions to provide the necessary QoS to their users. Most of the RENs however, still rely on MPLS-TE to achieve such goal in their production networks.

Notwithstanding, the growing traffic demands that RENs must deal with and the need to provide new services to the research community makes necessary the evolution of these types of networks on a regular basis. As such, since the appearance of SDN, RENs have started to evaluate the suitability of this new paradigm to evolve their infrastructure.

In a nutshell, with an SDN-based infrastructure, RENs can benefit from the lower CAPEX and OPEX expenditures. Furthermore, with an SDN-based approach, RENs can overcome some of the limitations

present in current TE solutions, as explained throughout this chapter and summarized in the list below:

- The higher granularity available at the SDN forwarding devices allows the utilization of more realistic traffic splitting rations.
- The logically centralized control plane of SDN enables the implementation of new path computation approaches.
- In SDN, having a centralized control plane which is aware of the global network state in real-time can alleviate current TED synchronization problems.
- The high programmability of SDN and the possibility to control the network out-of-band can overcome the long convergence times of current distributed protocols.

Considering current limitations in MPLS-TE solutions, and considering how SDN can help overcome such problems, this PhD thesis addresses the following research question:

Is it possible for network operators, and more precisely RENs, improve the operation and the resource utilization of their networks with an SDN-based TE solution? And if it is possible, how that can be achieved?

1.4 OBJECTIVES

In order to address the question introduced in the previous section, this PhD thesis presents a solution for TE in SDN networks. The main objective of the solution is to provide a generic framework to support the provisioning of connectivity services, including BoD, based on novel TE strategies that leverage the features present in SDN. Having such an objective in mind, the solution must tackle the specific goals listed below:

- The framework for TE in SDN networks must be technology agnostic, in terms of the D-CPI protocol that is employed. That

is, it must be able to operate over an SDN infrastructure regardless of the technology being used to control the network.

- The framework must be engineered to support the introduction of novel TE strategies and path computation algorithms easily.
- The framework must provide the means to retrieve accurate network state information in real-time, in order to present trustworthy TE information to the algorithms used to achieve the TE performance objectives.
- The framework must provide the means to assure the SLAs agreed between the network operator and the users.
- The framework must provide failure recovery mechanisms, in order to minimize the service disruption time and the packet loss.
- The framework must include open interfaces towards external elements such as orchestrators and external agents. This will allow the utilization of the framework in a variety of scenarios, and will allow RENs to continue using the multi-domain technologies already used in order to establish E2E circuits involving multiple network domains.

In addition, since this generic framework aims to satisfy the needs of the research and education community, it needs to support advance reservations. In that regard, the solution also satisfies the following functional objectives:

- The advance reservation mechanism must be able to accept the highest amount of service reservation requests as possible, which will have a positive impact in the perception of the users regarding the service. In addition, satisfying this objective will allow to increase the revenue of the network operator at the time of providing this service. That is, the solution must be able to maximize the network resource utilization in order to reduce the CAPEX and OPEX expenditures of RENs.
- The solution must be able to respond to the users' requests as fast as possible. That is, the time required to determine whether there

are enough resources in the network to satisfy the demand and to compute the path that will satisfy the connectivity requirements of the user must be minimized.

- Once a service reservation request is accepted by the system, it must be ensured that the service will be provided during the entire reservation period without disruption.

1.5 CONTRIBUTIONS

The main contributions of this PhD thesis are enumerated below:

1. A study of the impact of the different interfaces proposed by the ONF in TE has been conducted. First, a comprehensive list of SDN protocols operating at different interfaces have been reviewed, so as the research proposals relying on these protocols. After a qualitative evaluation, where the SDN protocols have been compared against TE capabilities of the PCE-based architecture, it has been possible to identify that the protocols operating at the D-CPI interface are the ones that benefit the most TE. As a consequence, a more in depth analysis of TE solutions based on D-CPI protocols has allowed to identify the desired features to be included in the generic framework for TE in software-defined networks.

2. A complete analysis of the impact of the data structures that are used to support advance reservations. This analysis has also allowed to identify the strengths and limitations of current advance reservation solutions.

3. The design of a generic framework for TE in SDN, called the Dynamic Path Computation (DynPaC) framework, that supports the introduction of novel TE strategies and path computation algorithms regardless of the SDN technology being used.

4. The introduction of resilience mechanisms such as path protection or path restoration to handle the network failures.

5. A novel data structure and a set of associated algorithms to support advance reservations in SDN that leverages the logically centralized

control plane and high programmability of this new networking paradigm.

6. A set of TE techniques that seek to improve the network resource utilization in software-defined networks, which rely on flow relocations and the disaggregation of the reserved services in a set of sub-services, easier to relocate.

7. The design of a Representational State Transfer (REST) interface towards external elements that allows the integration of the DynPaC framework with orchestrators.

8. The introduction of an operational mode in the DynPaC framework that allows its utilization in the Network Services Framework (NSF) to provide multi-domain connectivity services, which can alleviate the long times usually required to manually configure these kinds of services involving multiple REN.

9. The deployment of the pilot of the SDN-based BoD service in GÉANT, where the DynPaC framework will be tested by a set of selected users of the academic and research community.

1.6 CONTENT OF THE DOCUMENT

This remaining of this document is structured as follows:

- **Part II - State of the art:** This part reviews the most relevant technologies and proposals of application to this thesis. It is divided in the following two chapters:
  - **Chapter II - Traffic Engineering frameworks for Software-Defined Networks:** Reviews current frameworks to support TE in SDN depending on the performance objective that the algorithms seek to optimize.
  - **Chapter III - Algorithms and Data Structures for Advanced Reservations:** Analyzes the impact of the data structures to handle the resource management over time required by advance reservation mechanisms.

- **Part III - Proposal:** This part presents the solution proposed in this PhD thesis. It is divided in the following three chapters:
  - **Chapter IV - Generic framework for Traffic Engineering in SDN:** This chapter describes the architecture of the DynPaC framework. It presents the modules that comprise the framework and the support for multi-domain capabilities.
  - **Chapter V - Support for advance reservations:** This chapter is focused on the support for advance reservations in the proposed solution. It includes information about the reservation model that it supports, the data structures specifically designed to handle advance reservations in SDN and the operations supported by these data structures.
  - **Chapter VI - Optimization Techniques:** This chapter describes a set of optimization techniques that the DynPaC framework supports to improve the network resources utilization, namely *flow relocation* and *flow disaggregation*.
- **Part IV - Validation:** This part is focused on the validation of the contributions presented in this PhD thesis. It is divided in the following chapters:
  - **Chapter VII - Functional Validation:** In this chapter, the set of experiments conducted to demonstrate that the proposed framework is able to satisfy the objectives of this thesis are described.
  - **Chapter VIII - Performance Evaluation:** In this chapter the performance of the advance reservation mechanism proposed by this PhD thesis is evaluated, in order to state its feasibility in REN networks. Furthermore, it is demonstrated that the system is able to perform the admission control of the reservations requests in the order of milliseconds.
- **Part V - Conclusions:** The last part of this document is divided in the following chapters:
  - **Chapter IX - Conclusions and Future Work:** This chapter summarizes the conclusions of this PhD thesis, the

dissemination results consequence of this research effort and presents the future work of this research line.

Part II

STATE OF THE ART

2 Traffic Engineering Frameworks in Software-Defined Networks

> "Any problem in computer science can be solved with another level of indirection."
>
> — David Wheeler

As mentioned before, SDN has been appointed as the next generation TE enabler. Among the benefits provided by SDN, its capability to make network-wide decisions from a logically centralized control plane and its high programmability are the ones that will impact the most future TE solutions.

As a consequence, this chapter presents a comprehensive survey about current proposals for TE based on the utilization of SDN technologies. The main objective of this survey is to identify current frameworks for TE and their strengths and limitations. The analysis conducted to the current solutions has allowed to identify the need of a generic framework to support TE in SDN deployments, especially suited to satisfy the requirements of RENs.

The different proposals analyzed in this chapter have been selected given their relevance, while other solutions like [82–84] are not described in this chapter since they are based on similar techniques or even if they present TE solutions, they do not propose a framework for TE. They have been categorized considering their performance objective, namely *network resource optimization*, *congestion minimization*, *packet loss minimization* and *QoE maximization*. It is worth noting that the analysis of the different

proposals is focused on their architectural design and their support for TE.

This chapter is structured as follows. First, an introductory section presents the benefits that using SDN provides to the TE, where it is identified that the protocols operating at the D-CPI interface are the ones with the highest impact on TE. As a consequence, an overview of a comprehensive set of SDN protocols operating at this interface is provided. Later, Sections 2.2 to 2.5 present the different SDN-based TE solutions, in order to identify the current state in terms of TE support in this new networking paradigm. Finally, Section 2.6 summarizes and compares the different solutions while Section 2.7 presents the conclusion of the analysis.

2.1 INTRODUCTION

Currently, the SDN ecosystem consists of multiple protocols and technologies. While OpenFlow is considered the *de facto* protocol for SDN and most proposals, either focused on TE or not, are based on this technology, other solutions such as Link Layer Discovery Protocol (LLDP), Generic Routing for Encapsulation Network Virtualization (NVGRE) [85] or Virtual eXtensible Local Area Network (VXLAN) [86] are also considered SDN technologies. Notwithstanding, this chapter focuses on the SDN protocols operating at the D-CPI interface, since they are the ones that benefit the most TE, as stated in [87].

2.1.1 IMPACT OF THE SDN INTERFACE TYPE TO TE

Given the extend of the SDN ecosystem, an in-depth survey where the impact of a comprehensive list of SDN protocols to TE was analyzed was conducted in [87] by the author of this PhD thesis. The main objective of this survey was to answer the following question.

How does the interface type in which the SDN protocols operate impact TE?

In summary, nine different protocols considered to be part of the SDN ecosystem were analyzed, namely ForCES, OpenFlow, Interface to the Routing System (I2RS), Border Gateway Protocol - Link State (BGP-LS)

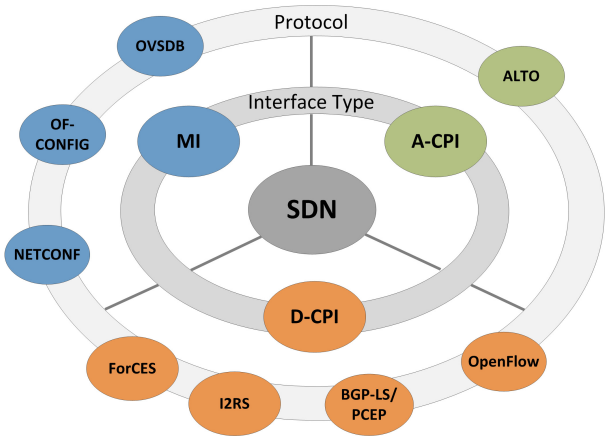


Figure 2.1.: Categorization of SDN protocols depending on the interface type.

in conjunction with Path Computation Element Communication Protocol (PCEP), Open vSwitch Database Management Protocol (OVSDB), Network Configuration Protocol (NETCONF), OpenFlow Management and Configuration Protocol (OF-CONFIG) and Application Layer Traffic Optimization (ALTO). These protocols were categorized depending on the interface type at which they operate as defined in the SDN architecture proposed by the ONF (see Figure 2.1). The three interfaces defined by the ONF are the ones described below:

- *D-CPI protocols:* used to communicate the data plane with the controller plane. The D-CPI is aware of an instance of the data plane's informational model, that is, the set of resources on the data plane and the operations that can be performed on them. These protocols operate on an event timescale, that is, they are able to enable or disable circuits at the data plane within milliseconds.

- *A-CPI protocols:* protocols that are used to communicate the controller plane with the application plane. The protocols in this category can provide an abstraction of the network resources to the applications, or user-friendly and standardized mechanisms to program the network elements.

- *MI Protocols:* the ONF's SDN architecture includes management technologies to operate over the three planes. More precisely, the MIs that operate over the data plane are used to manage

the network elements, and are in charge of tasks such as policy provisioning, port, queues or LSPs configuration and in some cases, even of failure detection. They operate on much slower timescale when compared with the D-CPI protocols, within minutes or hours.

In order to determine the impact of these protocols to TE, they were evaluated taking as a reference the most advanced TE architecture present in today's production networks, the PCE-based architecture. To that matter, the following set of evaluation metrics defined in RFC 4655 [88] and two additional metrics were used (marked with an asterisk).

- **Optimality:** the ability to maximize network utilization and minimize cost, considering QoS objectives, multiple regions and multiple layers.
- **Scalability:** the implications of routing, TE LSP signaling, and PCE communication overhead, such as the number of messages and the size of the messages.
- **Load sharing:** the ability to allow multiple PCEs to spread the path computation load by allowing multiple PCEs to take responsibility for a subset of the total path computation requests. It should not be confused with load balancing the traffic among multiple paths. In the case of SDN protocols, it refers to the ability to have multiple controllers implementing the TE solution.
- **Multipath computation:** the ability to compute multiple and potentially diverse paths to satisfy load-sharing of traffic and protection/restoration needs including E2E diversity and protection within individual domains.
- **Re-optimization:** the ability to perform TE LSP path re-optimization. In the case of SDN protocols, it refers to the ability to relocate flows onto alternative paths.
- **Network stability:** the ability to minimize any perturbation on existing TE state resulting from the computation and establishment of new TE paths.

- **Accurate TED synchronization:** the ability to maintain accurate synchronization between TED and network topology and resource states.
- **TED synchronization speed:** the speed which TED synchronization is achieved with.
- **Impact on data flows:** the impact of the synchronization process on the data flows in the network. This metric refers to the effect that a poor TED synchronization will have on the data flows. For instance, based on outdated information a path could not be the optimal one, resulting in a performance degradation or even a path being computed for already unavailable resources.
- **Granularity*:** refers to the number of possible classifiers that can be used at the network devices to forward the packets. That is, the number of header fields and wild-carding options that can be taken into account to classify the packets at the networking devices (e.g., an IPv4 source address and its mask). The higher the number of available packet classifiers the higher the granularity is, resulting in finer-grain flows. It has clear implications on the multipath capabilities of the solutions, since a higher granularity allows to split the traffic more conveniently. Similarly, it also impacts the optimality and the re-optimization capabilities of the solutions.
- **Equipment configurability*:** capacity to configure the network equipment to enforce the establishment of paths.

The survey concluded that the protocols operating at the D-CPI interface are the ones that benefit the most TE, as summarized in Table 2.1. The symbols indicate that the SDN protocol capabilities are (✓):*better*, (✗):*worse* or (=):*equal* compared to the capabilities of the PCE-based architecture. Among the benefits provided by the SDN technologies operating at the D-CPI interface it is worth remarking the following ones:

- The higher granularity available at the forwarding elements programmed through D-CPI interfaces enables the utilization of more appropriate traffic splitting ratios for load balancing.

Table 2.1.: Summary of the impact of SDN protocols to TE.

	Optimality	Scalability	Load sharing	Multipath comp.	Re-opt.	Net Stability	TED accu.	TED Sync speed	Impact on data flows	Granularity	Equipment config.
ForCES	✓	✓	✓	✓	✓	✓	✓	✓	=	✓	✓
OpenFlow	✓	✓	✓	✓	✓	✓	✓	✗	✗	✓	✓
I2RS	✓	✗	✓	✓	✓	=	✓	✗	=	=	
BGP-LS/PCEP	✓	*	✓	✓	✓	=	✓	✗	=	=	
OVSDB											✓
NETCONF							✓				✓
OF-CONFIG	✗						✗				✓
ALTO	✓	✗	✓	✓	✓	✗	✓	✗	✗	✗	

- The utilization of a logically centralized control plane allows taking decisions based on an up-to-date global view of the network, which facilitates the optimization of the network performance.
- The possibility to have a dedicated element in charge of the path computation as in the PCE-based architecture allows easier introduction of algorithms with different objective functions.
- The high programmability of the forwarding elements allows the application of fast failure recovery mechanisms.
- The utilization of an out-of-band control mechanism reduces the convergence time needed by distributed protocols such as RSVP-TE.

Since the D-CPI protocols are the ones that impact the most on TE solutions, the following subsection will introduce a comprehensive set of protocols operating at this interface.

2.1.2 OVERVIEW OF THE MOST RELEVANT D-CPI PROTOCOLS

As mentioned before, the D-CPI protocols are the ones used to communicate the data plane with the controller plane (see Figure 2.2, where the D-CPI interface is identified). As such, an overview of protocols operating at this interface is presented, namely ForCES, OpenFlow and BGP-LS/PCEP.

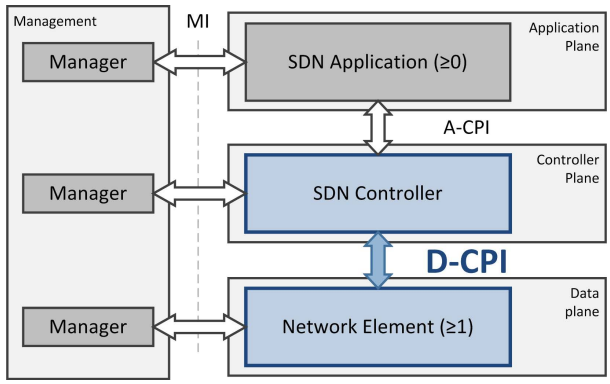


Figure 2.2.: D-CPI interface in the ONF architecture.

2.1.2.1 FORCES

Back in 2003, the ForCES Working Group (WG) of the IETF presented the ForCES framework [89], which enables the separation of the control and forwarding planes of the network elements. Although the framework and the homonym protocol were designed to easily add new functionalities to the forwarding plane, neither the industry nor the academia adopted the proposal. In fact, due to the lack of open implementations of the ForCES protocol, the ForCES framework was ostracized [90]. Currently, with SDN being a hot topic, the ForCES WG has resumed the standardization process. As stated in the RFC 3746 [91], ForCES does not only define a framework, but also standardizes all the associated protocols that make possible the information exchange between the control and the forwarding planes. It can be considered as a framework aiming to improve network programmability through an open interface. However, unlike other SDN technologies, the ForCES framework does not impose a centralized control plane, in fact, it can be used with legacy distributed control protocols.

In the ForCES framework, which is depicted in Figure 2.3, a Network Element (NE) consists of Forwarding Elements (FE) and Control Elements (CE). In short, the FEs are logical entities that use the underlying hardware to provide per-packet processing. They must support a minimal set of capabilities to be able to establish network connectivity. FEs are formed by Logical Forwarding Blocks (LFB), which are programmed by the CE by means of the ForCES protocol

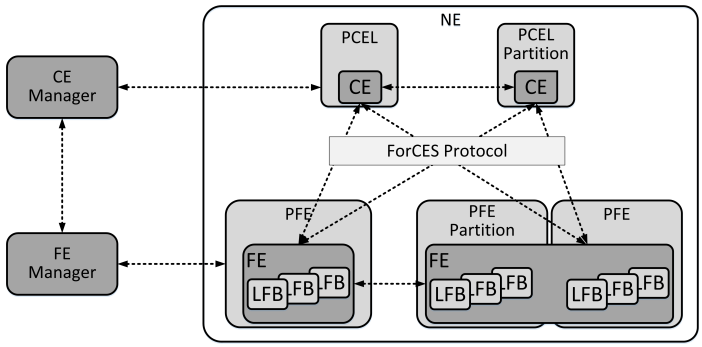


Figure 2.3.: Main components of the ForCES architecture.

to implement a wide variety of logical functions, e.g., Layer 3 (L3) forwarding, Firewall or Network Address Translation. The FEs reside inside Physical Forwarding Elements (PFE), whereas CEs do the same in the Physical Control Elements (PCEL)[1]. Typically, the PFEs and the PCELs are placed in the same physical machine, although, they can also be located separately as specified by the RFC 6041 [92] by a single or multiple hops, as stated in RFC 6053 [93].

There are two operational phases identified in the ForCES framework. First, in the pre-association phase the *CE manager* and the *FE manager* decide whether the CEs and the FEs are part of the same NE. However, this operational phase is out of the scope of the ForCES protocol. Second, in the post-association phase, the FEs and the CEs use the ForCES protocol to associate and exchange information to facilitate packet processing.

The ForCES protocol supports CEs redundancy. Multiple CEs can operate over the same FE, though, the coordination between the CEs is out of the scope of the ForCES protocol. As a consequence, it is possible for different CEs to implement different routing or signaling protocols, where the FE acts as the entity in charge of redirecting the control packets to each one of the CEs according to some filtering rules. Similarly, the framework also supports the coexistence of multiple FEs, which imposes additional challenges. First, the functions that each one of the FEs implement must be very well defined, as it can affect the overall performance of the system. Furthermore, depending on the functions

1 To avoid confusion with the PCE

that each FE is in charge of, it may be necessary to perform multiple forwarding decisions in more than one equipment.

ForCES is a master-slave protocol, with CEs acting as masters and FEs as slaves. The protocol provides the means to associate the different elements of the framework, so as to tear down such associations. It is also in charge of transmitting subscribed-to events from FEs to CEs and of responding to status requests issued from the CEs to the FEs. Additionally, it is used to configure the FEs and the associated LFBs' operational parameters, so as to activate or deactivate the FEs. In the end, the protocol manages the LFBs at the FEs, which are compliant with the FE model defined in the RFC 5812 [94].

As mentioned before, the FEs are comprised of LFBs that are interconnected in a direct graph, and receive, process, modify, and transmit packets along with meta-data. The FE model establishes a formal way to define the FE's LFBs using eXtensible Markup Language (XML), while the configuration components, capabilities and associated events of the LFBs are defined when they are formally created. On the one hand, the FEs can be broadly defined by simply specifying their capabilities. For instance, FEs can be described in terms of IPv4 or IPv6 forwarding support or by the set of matching fields supported for the packet classification. On the other hand, the FE model can also be used to describe the FE state model, which presents a snapshot view of the FE to the CE. For each LFB, the number of inputs and outputs can be specified, as well as the packet types accepted in each of them and the routing criteria.

As stated in [95], the ForCES protocol is powerful enough to define other protocols. For instance, the authors of that paper state that both OpenFlow and NETCONF, could be considered subsets of the ForCES protocol. Therefore, according to the ONF's SDN architecture ForCES could be considered both a D-CPI and an MI.

In summary, ForCES is helpful for TE because it can be used with legacy protocols, supports CE redundancy and it can be used to define other protocols such as OpenFlow and NETCONF.

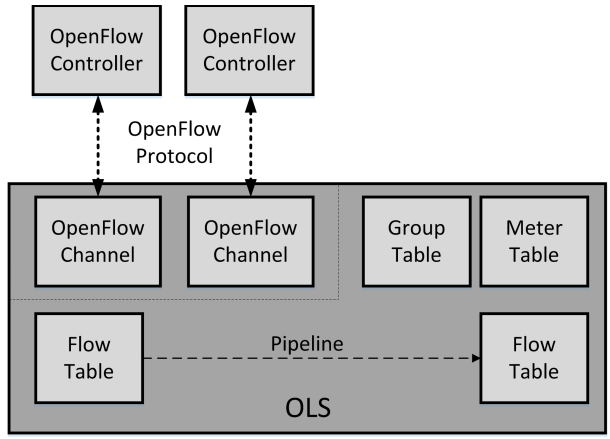


Figure 2.4.: Main components of the OpenFlow switch.

2.1.2.2 OPENFLOW

Back in 2008, the Stanford University released the first stable version of OpenFlow. Since then, the ONF has become the SDO in charge of the standardization of the OpenFlow Switch Specification and its homonym protocol. The OpenFlow Switch Specification defines both the OpenFlow Logical Switch (OLS) and the OpenFlow protocol, used for the communication between the OLS and the OpenFlow controller.

As depicted in Figure 2.4, the OLS consist of one or more control channels and a datapath. The datapath is where the packet look-ups and forwarding are performed, by means of one or more flow tables, a group table and a meter table. The OLS connects to the external controller through the OpenFlow channel, often referred to as the control channel, using the OpenFlow protocol.

Each OLS must have at least a flow table comprised of flow entries, which are formed by the *match fields*, the *priority* that specifies the matching precedence of the entry, the *counters* that hold statistical information, the set of *instructions* that are applied to the matching packets, the *timeouts* and the *cookie* that unambiguously identifies the flow entry (see Figure 2.5). Among the instructions that can be applied to packets, the ability to direct the packets to specific *meters* is of special relevance for TE. Meters are switch elements able to measure and control the rate of the packets being forwarded; therefore, they play a key role in QoS enforcement. Instructions can also be used to apply a

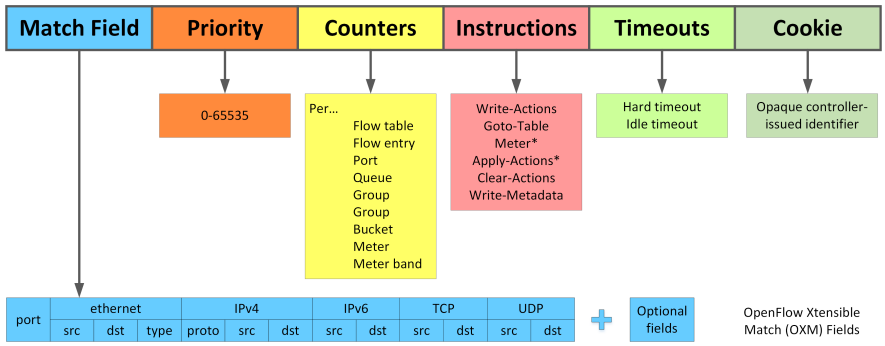


Figure 2.5.: Structure of the flow tables in an OpenFlow switch.

certain set of actions. Available actions include sending the packet to a queue or to an outport (output port), directing the packet to a group table or re-writing a specific field, to cite a few. Another interesting feature of OpenFlow is the fine-granularity that it supports for the matching of packets. OpenFlow takes into account at least the physical ingress port and additional Ethernet, IPv6, IPv4, TCP and UDP header fields. Moreover, additional header fields can be included thanks to the OpenFlow eXtensible Match (OXM), which is a very flexible model where new matching fields are defined as Type Length Values (TLV).

In a nutshell, the external controller populates the flow tables of the OLS with the flow entries that determine the behavior of the traffic that matches them. Each OLS can contain more than one flow table with its corresponding flow entries. The OpenFlow pipeline process defines how packets interact with those flow tables. According to the latest OpenFlow Switch Specification [7], packets are always matched first against the first flow table and in the cases where there are multiple flow-tables, packets are forwarded to the subsequent ones. When a packet matches one or multiple flow entries, the instruction set associated to the entry with the highest priority is applied, which can include directing the packet to another flow table. Then, when the pipeline process finishes, either because there are not more re-directions to subsequent flow tables or because it is the last flow table, all the associated actions are applied. It is worth mentioning that an OLS is able to handle flow miss-matches. Depending on the configuration, when a packet arrives that does not match any of the flow entries installed in a flow table, the OLS can specify

how to process it. As a consequence, packets can be directed to another flow table or be sent to the controller. This feature makes possible to work reactively besides of proactively; that is, to act in response to packets that do not match any entry of a flow table.

There is a single group table per OLS, which makes possible to represent additional forwarding methods. For instance, in an OLS it is possible to flood packets creating a group that associates output actions to all the ports but the ingress port. Each entry at the group table is defined by a unique identifier, a set of counters, the action buckets (ordered list of actions to execute and the associated parameters) that must be applied to the packets and the *group type* they belong to. For the moment, OpenFlow defines four different group types: all, select, indirect and fast fail-over. The first one is characterized by applying all the action buckets defined for the group. The *select* group type uses just one of the action buckets associated to the group for each packet, e.g., in a round-robin fashion. Third, the *indirect* group type supports a single action bucket. Finally, the *fast fail-over* group applies one action bucket at each time, following the order in which they are configured.

Regarding the OpenFlow controllers, it is worth mentioning that network operators can choose between centralized (e.g., NOX [96] POX [97], Trema [98], Ryu [99], FloodLight [100], Beacon [101], Maestro [102], McNettle [103], Jaxon [104], Snac [105]) or distributed (e.g., Onix [106], HyperFlow [107], Helios [108]) controllers. They can also select the programming language to use, being Java and Python the most popular ones. Furthermore, there are also available special purpose controllers, such as FlowVisor [109], Open Virtex [110] and AutoSlice [111], which make possible to virtualize OpenFlow-based networks by slicing the network resources and exposing the network control to other controllers transparently. For further information regarding network virtualization with OpenFlow see [112–114]. In addition, several frameworks have appeared recently that support a set of SDN protocols, including OpenFlow. This is the case of the Open Network Operating System (ONOS) [115], Open Daylight Project (ODP) [116] or Cisco Open Network Environment (ONE) [117]. Further information about OpenFlow controllers can be found in [118].

Undoubtedly, the OpenFlow protocol is a D-CPI used to communicate the OLSs that reside in the data plane with the OpenFlow controller placed in the homonym plane of the ONF's SDN architecture. OpenFlow has a number of mechanisms that can benefit TE. Firstly, it provides the means to add, delete and modify meters for traffic shaping. Secondly, its high-granularity makes it ideal to implement flow disaggregation strategies. Thirdly, it supports alternative forwarding methods through the group table, such as load balancing or fast fail-over.

2.1.2.3 BGP-LS/PCEP

Defined by the IETF Inter-Domain Routing WG [119], BGP-LS is a protocol used to collect and share information about link state and TE [120]. By means of a set of extensions added to the Border Gateway Protocol (BGP) routing protocol [121], BGP-LS retrieves the topological information from the Link State Databases and distributes it to a consumer both directly or through a BGP Speaker or a Route Reflector. A BGP speaker exchanges network reachability information with other BGP speakers, including the intermediate Autonomous Systems (AS) that the traffic must transit to reach destinations, whereas a Route Reflector is mostly used as a concentrator for multiple BGP speakers inside an IGP area. Taking into consideration that BGP is an inter-AS technology, with BGP-LS it is possible to provide information about other IGP areas to the external components.

Although it can work independently, BGP-LS is a mechanism that can also be used by multiple applications, such as PCE and ALTO. For example, PCE performs path computation using TE information, and TE information is never exchanged across different network domains. Since BGP-LS can be used to exchange TE information between different IGP areas and network domains, BGP-LS makes a PCE capable of computing E2E paths across different IGP areas. Thus, BGP-LS is a mechanism that can improve actual TE solutions such as the PCE-based architecture.

BGP-LS can be used to provide information about the maximum bandwidth, the maximum reservable bandwidth or the unreserved bandwidth on a given link. It can also be used to inform about the default TE metric. That is, to inform about the objective function of the TE strategy, such as the minimization of the delay or of the link utilization.

Many vendors have started to include BPG-LS support in their devices (i.e., Cisco or Juniper), so as many SDN platforms like the aforementioned ODP, ONOS and Cisco ONE [122]. BGP-LS by itself cannot be considered a full D-CPI technology, since it is only valid to exchange topological information among network elements and does not provide network programmability. However, most SDN controllers use BGP-LS together with the PCEP protocol, which is the reason why these two protocols working together are considered another D-CPI solution in this chapter.

As mentioned before, the main characteristic of the PCE-based architecture is that the path computation is performed in a dedicated element. In this architecture, a Path Computation Client (PCC) requests a path, which is computed by the PCE using the TE information stored in a TED. In order to fulfill its intended objective, the PCE-based architecture relies on two key protocols: Path Computation Element Communication Protocol (PCEP) [123] and RSVP-TE, defined in RFC 4657 [123] and RFC 3209 [124] respectively .

As stated in the RFC 4657 [123], the PCEP protocol is used for the communication between PCCs and PCEs, so as for the inter-PCE communication. Figure 2.6 depicts how a PCC communicates with a PCE to request a path computation. (1) the PCC sends PCEP *PCReq* messages to the PCE when it wants a path to be computed for one or more TE LSPs. Using the same message, it sends the set of constraints and attributes that the PCE requires to compute the path and a priority number to indicate the urgency of the request. When the PCE has finished computing the path, it replies (2) with a PCEP *PCRep* message, which can be a negative message indicating the reason why the computation has failed or a positive one. In the latter case, the response includes the set of computed paths and the sets of attributes associated with them, such as the path costs (e.g., cumulative link TE metrics and cumulative link IGP metrics) and the computed bandwidth. In order to avoid negative messages, the PCE can notify PCCs that it is unable to satisfy certain requests or that it has been experiencing unacceptable delays. This way, since the PCE-based architecture supports multiple PCEs in the same network domain, the PCC has the opportunity to send its *PCReq* to another PCE.

Using the PCEP protocol, (3) the PCEs send explicit paths to the PCCs specified by means of Explicit Route Objects (ERO). These EROs are

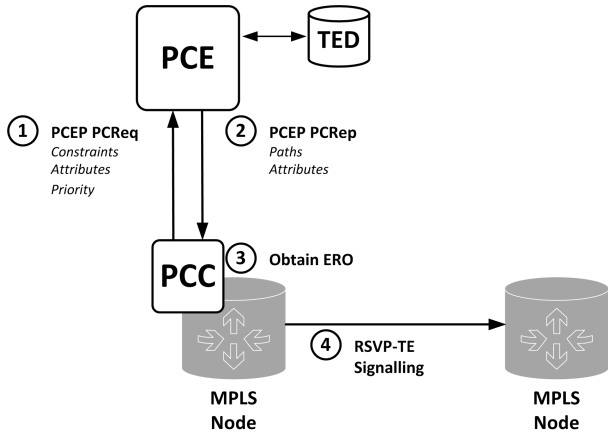


Figure 2.6.: Main components of the PCE-based architecture.

used for the (4) establishment of the LSPs through RSVP-TE in MPLS and GMPLS networks. They consist of sets of IPv4/v6 prefixes and AS numbers, among other possible parameters. Hence, the computation of the paths must support everything that can be expressed in an ERO, like the degree of paths disjointedness or the maximum hop count among others.

It is worth mentioning that PCEP includes support for load-balancing. The PCC can indicate the support for load-balancing and the number of paths that can be included in the balancing group. This is a very interesting feature for multipath communications and the minimization of the link load, because the more paths to split the traffic, the lower the load on each of them will be.

There are multiple PCE types, depending on the number of PCEs involved or the computational model they follow. Among all the possibilities, the centralized PCE is of special relevance for this thesis as it can be considered as a predecessor of current SDNs. PCEs can be stateful or stateless depending on how they manage the network state. On the one hand, a stateful PCE is aware of both, the network state (links state, bandwidth, etc.) and the set of already computed paths and reserved resources in the network. The stateful PCE requires reliable state synchronization mechanisms, which can result in control plane overhead. On the contrary, a stateless PCE has knowledge about the network topology (nodes, links, bandwidth, etc.), information that it uses for

the path computation, but it does not take into account the amount of resources that are already used or reserved in the network (e.g. current link utilization). That is, in a stateless PCE each request is processed independently, without considering the resources allocated by previous requests, which results in a much simpler path computation.

In addition, BGP-LS is not only a protocol that network devices can use to inform about the topological and link state information. It can also be used in Hierarchical Path Computation Element (H-PCE) to exchange TED information between PCEs [125]. In multi-domain path computation, child-PCEs are in charge of the computation of the paths in each domain, while the parent-PCE is in charge of what in the multi-domain terminology is known as the inter-domain path computation. The multi-domain path computation is in fact a two-step process in which first the parent-PCE computes the domain sequence and then the child-PCEs compute the paths inside each domain.

To summarize, on the one hand, BGP-LS is able to retrieve topological and link state information but lacks the necessary mechanisms to program the network elements. On the other hand, PCEP is able to program the network elements but lacks the necessary mechanisms to retrieve information from the network resources. However, the two protocols complement each other, and working together compose another D-CPI technology to be taken into account. Working in conjunction with BGP-LS can lead the PCE-based architecture to a whole new level, since it can be useful to solve many of the limitations found regarding the retrieval of TE information to store it in the TED.

2.2 TE FRAMEWORKS FOR NETWORK RESOURCE UTILIZATION OPTIMIZATION

The optimization of the network resource utilization is one of the most common objective functions in TE, and therefore, in SDN-based TE. This a direct consequence of the higher revenue that network operators can achieve by utilizing the network resources they have available in a more efficient manner. An in this regard, having a logically centralized control plane can impose a great benefit, since it allows to make routing decisions based on the overall network state.

Multiple solutions have emerged in the past few years dealing with the optimization of network resources in software-defined networks. Among the techniques utilized to achieve such a goal, the utilization of load balancing techniques not based in ECMP are very popular. For instance, in [126], a Open Shortest Path First (OSPF) routing optimization scheme that minimizes the maximum link utilization between the FEs on a ForCES router is achieved by means of a weight-smart OSPF algorithm. Similarly, Van der Pol *et al.* use Multi-Path TCP (MPTCP) to distribute the traffic across multiple paths in an OpenFlow controlled Wide Area Network (WAN), while [127] performs load balancing in campus networks. Other solutions deal with scheduled data transfers that exploit the availability of well-characterized traffic patterns, or define hierarchical architectures to support multi-domain service provisioning with QoS.

There is though something that all the above solutions and the ones described in the following sections do have in common. Whatever it is the specific technique utilized to improve the network resource consumption, and whatever it is the network resource being optimized, all these solutions rely on path computation engines embedded in the control plane. This is a natural evolution of the PCE-based architecture, which is sometimes referred to as an early implementation of SDN. Thanks to this approach, TE solutions can easily adopt novel and more sophisticated routing algorithms that take advantage of the up-to-date network state information available at the controller plane.

The following subsections present the different proposals analyzed in this chapter that deal with the optimization of the network resources utilization. They have been selected given their relevance to the field, and because they provide detailed information about their architectural design and TE support.

2.2.1 BIN *ET AL.* PROPOSAL

Cloud Computing has made possible the shared use of computational resources by multiple tenants, enabling the rapid development of new businesses by removing the need to purchase hardware or software components. This is achieved by means of resource scheduling, which affects the utilization of the resources and the quality of the service

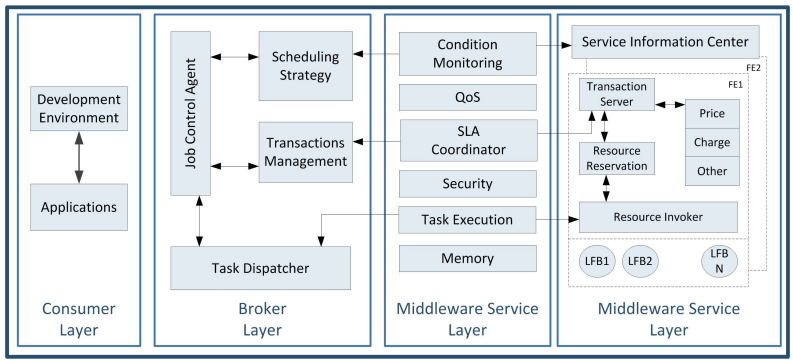


Figure 2.7.: Architecture proposed by Bin *et al.* [128].

provided to the tenant. The same premise is applicable to the network resources. Moreover, network resource sharing is also considered in the Cloud Computing environment, which is known as Network as a Service (NaaS).

Having in mind the benefits provided by the application of these resource scheduling policies in the Cloud Computing environment, Bin *et al.* [128] proposed a solution that applies the same principles to achieve higher and more efficient network resource utilization in a virtualized environment. Their solution is applied to ForCES routers, in order to increase their performance through the application of a resource scheduling algorithm based on an economic model that allows to select resources in a programmable and scalable fashion.

- *Design*

The architecture proposed by Bin *et al.* is depicted in Figure 2.7, where the layers it is comprised of are described below:

- **Consumer Layer**: External entity that allows end-users to send requests to the CE.
- **Broker Layer**: Part of the CE responsible of resource discovery and allocation.
- **Middleware Service Layer**: Part of the CE that enables the construction of the ForCES econometric model, which considers QoS and security information.

- **Resource Service Layer**: Includes the FEs and the LFBs that can be allocated.

In a nutshell, the solution presented in this section allows users to specify a network service request and a SLA that needs to be satisfied. Then, the Broker Layer uses an algorithm called Deadline And Budget Priority based on Deadline Budged Constraint (DABP) to select the optimal combination of available LFBs able to satisfy that demand with the information provided by the Middleware Service Layer, taking also into account budget and deadline constraints. Once the suitable resources have been selected and reserved for the user, the Broker Layer signs the SLA contract ensuring that the network service request will be provided with the necessary guarantees. Finally, the SLA is signed, the request is sent to the Resource Service Layer and the LFBs are instantiated to form the ForCES router instance that will provide the service to the user.

- *TE support*

They propose an algorithm that not only takes into account deadline constraints but also budget constraints, named DABP. The algorithm finds the minimum amount of LFBs available at the moment of making the request to satisfy the user request. To be able to do that, the algorithm selects the LFBs taking into account QoS and pricing objectives and the real-time node computing resource utilization.

- *Conclusions*

Although their solution allows to define a ForCES router compliant with the SLA specified by the user, the only resources that are taken into account to select the optimal set of LFBs are computational resources, such as computer or storage resources. However, even if this resources can have an impact on the QoS provided to the users, network resources such as link capacity are left behind. That is, the solution takes into account node-related resources, while link-related resources are not considered.

Notwithstanding, it is worth noting that the solution does take into account the feedback from the LFBs to the CE in order to help make better choices on future resource selections. All in all, the architecture

that they present is tightly related to the use case. They do not present a generic framework that could be utilized for other use cases making use of ForCES routers.

2.2.2 B4

The implementation of TE strategies based on OpenFlow has become a hot topic since the announcement that Google uses SDN and OpenFlow to optimize the links utilization in B4 [51], one of its internal WANs. B4 uses SDN principles and the OpenFlow protocol for the control of the switches. This network supports simultaneously standard routing protocols and a centralized TE solution implemented as an SDN application. With this solution, Google has achieved a 70 % average link utilization, almost the double of what traditionally is achieved in over-provisioned networks.

It is worth noting that Google's solution is entirely customized to their use case and it should not be considered a generic solution for TE in SDN, although the same principles could be applied for other environments with modifications. Among the characteristics that Google has exploited to customize its solution are: (1) they control all the elements involved in the data transfer, from applications to servers and local Area Networks (LAN), (2) their most bandwidth-consuming application is large-scale data copy between DCs that supports adaptive rates and (3) their WAN interconnects a small number of DCs, making centralized control feasible. All in all, they outline SDN as the main driver to achieve a network resource utilization never seen before with traditional WAN architectures, thanks to the possibility of making decisions based on the global view of the network from a centralized controller.

- *Design*

In B4, the network is abstracted to a TE overlay were the modules depicted in Figure 2.8 control the entire network. Later, the decisions taken from this overlay called *global layer* are issued to a distributed *controller layer* that programs the flow entries of OpenFlow switches. At the global layer, the solution consists of a *TE Server*, an *SDN Gateway* and a *Bandwidth Enforcer*, which are described below.

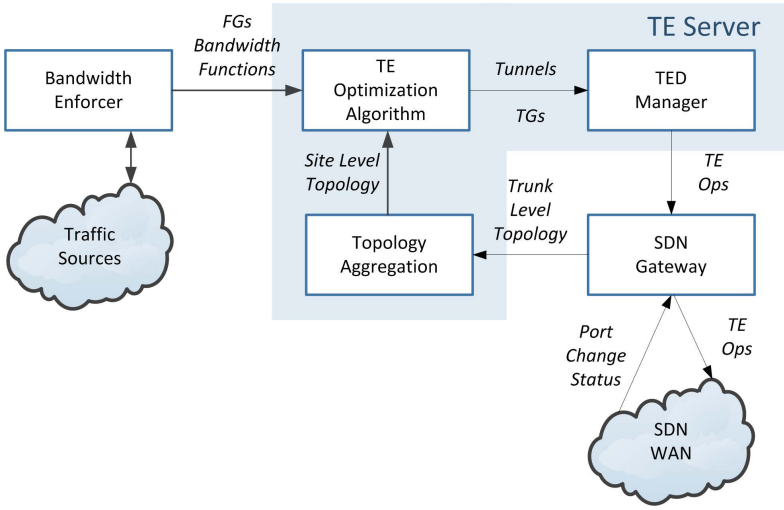


Figure 2.8.: Google's B4 architecture [51].

- **Bandwidth Enforcer**: Computes the *bandwidth functions* of each application, where a bandwidth function can be considered as the SLA between the application and the B4 network. They are computed taking into account the flow's relative priority, and later sent to the TE Server.
- **SDN Gateway**: Abstracts the underlying controller layer comprised of multiple instances of the ONIX controller and forwards the TE operations to it. In addition, it also feeds the TE Server with the topology information.
- **TE Server**: Uses the topology information provided by the SDN gateway to abstract the network topology in a simpler graph that is used by the *TE optimization algorithm*. The algorithm is then applied to *Flow Groups* that aggregate multiple application traffic demands in order to compute the *tunnels*, or the site-to-site paths. These tunnels are later sent to the *TED Manager* for their translation into TE operations to be sent to the *SDN gateway*.

- *TE support*

The algorithm uses the priority of the flow group that aggregates multiple applications' bandwidth functions to select the less congested

path using a max-min fair solution that maximizes network utilization while the fair share of the resources among the individual flows is ensured. Once the paths are computed, it allocates the necessary resources to satisfy the demand. Then, the traffic of the different applications is forwarded to their destinations using ECMP. It is worth mentioning that they use an heuristic approach to adapt the split ratio of the flow group to be compliant with the splitting capabilities of the forwarding devices actually used in the network. Once an edge is fully utilized, representing therefore, a bottleneck, it is removed from the topology graph, so it is never assigned to further applications until it is released.

In addition, this solution also provides failure recovery mechanisms. In essence, once a link failure is detected, the ECMP costs are updated and the traffic is continued to be delivered through another paths in the group in less than 4 ms, which is compliant with carrier grade requirements. However, the failure of a transit router on the network results in more than 3 s of service unavailability, since all the multipath entries of the affected services need to be updates, which usually takes 100 ms. Depending on the application, especially for those with high-availability requirements, a service unavailability of more than 3 ms is excessive.

- *Conclusions*

The solution presented in this section has been proven useful for the optimization of the network resource consumption. According to Google, this network is able to adapt to the growing traffic demands incurring in less costs and over-provisioning compared to traditional WAN architectures.

However, this solution requires prior knowledge about the network, meaning that it is only valid in networks where the traffic demand and pattern is previously known. Google uses the statistical information they collect about the network usage to optimize it. Furthermore, the solution also requires to have access to the end-devices or servers in order to modify the sending rate of the applications so that they are adapted to the resource availability. As a consequence, this solution is entirely customized to fit Google's needs and it would not be possible to apply it directly to control other networks without a similar statistical analysis.

In addition, the architecture for TE that they present for its utilization in the *global layer* is tightly coupled to their use case. Hence, it cannot be considered a generic framework for TE.

Moreover, the solution does not provide the means to support advance reservations, therefore being insufficient to satisfy the needs of RENs and Internet Service Providers (ISP) aiming to provide services like BoD.

2.2.3 SWAN

Similar to B4, Software-Driven WAN (SWAN) [129] is a system for inter-DC WANs based on OpenFlow that improves the network resource utilization coordinating the sending rates of the different services and centrally allocating network paths. In this solution, the SDN controller is the one that computes how much traffic each service can send and the network paths that can accommodate that traffic. The approach is similar to the one followed in B4, where high priority services are guaranteed with the optimal paths and services with the same priority are treated fairly. In addition, the solution considers the utilization of flow relocation techniques, and minimizes the effect of OpenFlow switches' re-programming by reserving a set of bandwidth and flow entries in the devices. Through these advanced TE strategies, they are able to carry up to a 60% of additional traffic compared to MPLS-TE applied in the same scenario. Furthermore, the solution has been proven useful outside the context of inter-DC communications, achieving an improvement of 16-25 % compared to MPLS-TE.

- *Design*

This solution differentiates three priority classes: interactive, elastic and background, being interactive the highest priority class and background the lowest one. All the services make their connectivity requests to the SWAN controller, which is aware of the global state of the network and the services being provided. Then, it estimates the sending rate for each of the services, reserves the shortest paths for higher classes' services, and allocates the necessary bandwidth for these services in strict precedence taking into account their class priority.

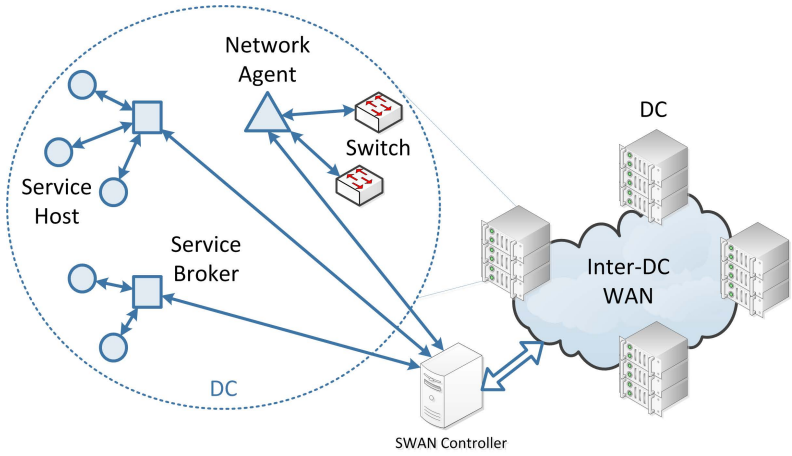


Figure 2.9.: SWAN architecture [129].

As depicted in Figure 2.9, the solution uses *network agents* to obtain information about the network topology. These network agents are used to inform the controller about link or node failures immediately, which triggers a process at the controller to relocate the services. In addition, the network agents also gather information collected by the end-hosts about future service demands every five minutes. This information is later used by the controller to re-compute the service allocations and optimize the overall network usage. Furthermore, the network agents also monitor the real network usage, providing additional information to the controller that can be used to further improve the resource allocation.

As mentioned before, in order to ease the relocation of flows and avoid congestion, SWAN reserves a set of flow entries and bandwidth respectively. Nevertheless, regarding the reservation of this remnant bandwidth, it is used to carry background services unless necessary. In the event of a service relocation into an alternative path, the background traffic is preempted, in favor of the higher priority relocated services.

- *TE support*

In SWAN, the path assignment is performed by means of Linear Programming (LP), which differs from the classic graph theory approach in that the path is found solving a system of equations. As input to a multi-commodity flow function, they use a set of possible paths between

the pairs of source and destination nodes. Although this approach may lead to underutilized resources, their results show that with an input of 15 paths they are able to achieve the necessary overall throughput.

The multi-commodity flow function is executed for each class of service in strict priority precedence. This is how they assure that the best paths are assigned for the highest priority services. After this phase, they invoke the multi-commodity flow function again for the services within the same service class, achieving the fair utilization of the network resources among them. Nonetheless, the output of the LP function is not necessarily feasible. The function does not take into account the flow entry availability in the network devices. As such, it may be necessary to run the LP function again, until a feasible solution is found.

Besides providing an improved network resource utilization, the SWAN controller also handles network failures. The aforementioned network agents inform the controller in case of a link or node failure, so that it can execute the necessary failure recovery mechanisms. In summary, once a failure is detected, the multi-commodity function is invoked again and executed over the new network topology.

- *Conclusions*

SWAN represents, with Google's B4, one of the most successful implementations of SDN-based TE. It is a mature solution, which has been proven efficient to improve the network resource utilization both in the inter-DC context and outside of it. However, the solution presents a set of drawbacks. First, the solution requires the utilization of network agents in order to monitor the network resources and estimate the service traffic demands. However, not all the vendors are willing to include new agents in their devices, making difficult the adoption of the solution. Second, by reserving a set of flow entries in order to deal with the network failures, they limit the amount of resources that are available to accept new demands. Third, the path computation procedure based on LP may result in an invalid result, since the availability of flow entries in the OpenFlow switches is not considered. In addition, although a framework for TE is described, its internal architecture is not public and it does not support advance reservations.

2.2.4 OSCARS

The On-Demand Secure Circuits and Advance Reservation System (OSCARS) software framework is one of the most well-known multi-domain advance reservation systems in the REN environment. Designed, implemented and maintained by the American ESNet, it is one of the first deployments in production of the PCE-based architecture. Currently, as in the case of the BoD service provided by Géant, it runs over an MPLS-TE transport network, and satisfies the requirements of researchers across the world in the need of high-capacity and long-term circuits.

Initially described in [130], OSCARS supports the provisioning of Layer 2 (L2) and Layer 3 unicast circuits. The solution supports both *immediate reservations*, that is, reservations that need to be deployed in the network at the time of their request and *advance reservations*, that is, reservations that need to be deployed in the future. Later, the framework has been extended with anycast circuits support [131], which has improved the overall system performance by reducing the blocking probability.

Being a production level solution with the code publicly available, there have been some efforts to use the OSCARS software framework over an OpenFlow network. For instance, ESnet developed a Path Setup Subsystem (PSS) for OpenFlow-enabled networks based on NOX, which was later adapted to the FloodLight controller [23].

- *Design*

As depicted in Figure 2.10, the OSCARS framework consists of ten modules whose work-flow is handled by a central element called *coordinator*. The most relevant modules for the support of advance reservations are described below.

- **Web Browser User Interface**: Provides a user-friendly Graphic User Interface (GUI) that non-technical users can use to make the service reservations easily.
- **Coordinator**: Manages the entire work-flow of the system. Once a service reservation request is received, it triggers the path computation procedure.

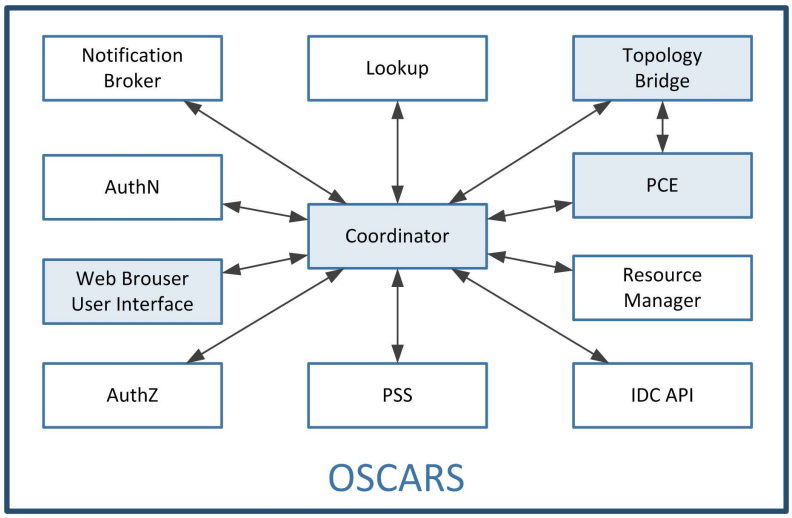


Figure 2.10.: OSCARS architecture [131].

- **Topology Bridge**: Stores the topology information that is later used by the PCE to compute the paths. Originally it was only updated every hour, but in its SDN-capable version, these limitations regarding the topology discovery are mitigated.
- **PCE**: Computes the optimal paths taking into account the topology information stored in the topology bridge and the resources consumed by the already reserved services.
- **Resource Manager**: Keeps track of the service reservations accepted in the system, the time at which they start and terminate and the path they have been assigned with.

The OSCARS software framework has been engineered in a modular fashion, to ease the introduction of new elements or the upgrade of the existing ones to support new features. For instance, the PCE was updated to support anycast connectivity services when the solution was already in its pre-production stage.

- *TE support*

The approach followed by ESNet to support TE in its network by means of the OSCARS software framework consists on the utilization of a PCE

as defined in the PCE-based architecture. Although any routing algorithm can be applied in this module, they have opted for C-SPF to compute the paths.

The PCE has been engineered in a hierarchical fashion, in which the network graph constructed using the information stored in the topology bridge is pruned sequentially. In summary, the PCE retrieves the network graph and removes all the resources that are not available at the time frame for which the new reservation is requested. Initially, the network graph may consist of 1000 nodes and 1500 links, making necessary its reduction to a simpler one. As such, the network graph is first pruned by removing all the links that do not have the required Virtual Local Area Network (VLAN) tags available. Later, the same is done with the links that do not have the necessary available bandwidth. Once the network graph is pruned, the shortest path is computed by means of the Dijkstra algorithm [132]. Once the path is computed, the information regarding the new service reservation, including its start time, its end time and the associated path is stored in a resource scheduling database for future requests.

- *Conclusions*

The OSCARS framework represents, according to this author's point of view, one of the most promising solutions for TE in the REN environment. It defines a generic framework for TE, flexible and modular enough to support different use cases. However, the framework lacks the mechanisms to retrieve accurate information from the network, at least in the MPLS version that runs in the production network, as the network is described by means of a static XML document that gets updated every hour. Furthermore, it lacks the mechanisms to adapt the routes to topological changes and does not provide any mechanism to monitor or analyze the traffic. More precisely, the retrieval of the topology information every hour means that the PCE that runs in the framework may provide false results as a consequence of operating over an outdated topology graph.

Moreover, this also impacts the way fault recovery is provided, which requires the establishment of the backup paths in the network at the time of installing the primary path, therefore, consuming network resources

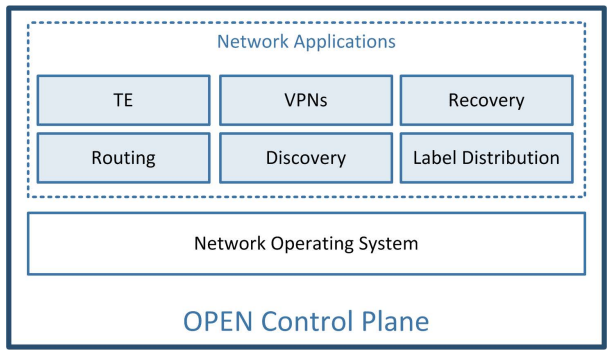


Figure 2.11.: Architecture of OPEN's control plane [134].

unnecessarily. In addition, at the time of installing the service into the network, the solution relies in RSVP-TE to distribute the selected routes among the MPLS devices, and as they noted in [130], this procedure can take several minutes. Luckily, aware of such limitation, some proposals to integrate this framework with OpenFlow have been made using FloodLight controller [133], and more recently ODP.

2.2.5 OPEN

The solution proposed by Das *et al.* [134] seeks to exploit both the advantages of OpenFlow and the highly efficient forwarding of MPLS. The solution is based on the utilization of the Open Programmable Extensible Networks (OPEN) control plane to implement MPLS-TE based on OpenFlow. They apply the OPEN control plane to an MPLS-based data plane implemented using a modified Open vSwitch (OVS) that performs MPLS data plane functionalities. In addition, they use a NOX controller modified to work with some MPLS extensions added to the OpenFlow protocol. However, this solution is merely focused on the application of OpenFlow as an alternative Label Distribution Protocol (LDP), leaving other functionalities such as resilience or dynamic LSPs establishment aside. Moreover, in OpenFlow 1.1 the protocol was extended to support MPLS labels as matching field, and even to push and pop MPLS labels. The same authors implemented in [135] a set of TE techniques in a NOX controller to optimize several services in

OpenFlow networks. The optimization depends on different parameters gathered with real-time monitoring tools at the edge devices of a WAN.

- *Design*

The OPEN control plane consists of a set of network applications that run on top of a Network Operating System (NOS) in order to provide TE in MPLS networks. The architecture of the control plane is depicted in Figure 2.11. It consists of a specialized module called *TE* in charge of the implementation of the TE strategies. In addition, the architecture includes modules in charge of the network topology discovery and of the recovery functionalities. Besides, it has a specific module to handle the distribution of the MPLS labels.

- *TE Support*

In today's networks, MPLS-TE is the preferred choice to support TE. With that premise in mind, the solution proposed by Das *et al.* brings the benefits of the high-programmability and the logically centralized control of the network of SDN to the MPLS domain. In a nutshell, TE is provided through the utilization of a C-SPF algorithm that operates on top of a network graph automatically discovered by the OpenFlow controller. The C-SPF computes the optimal path taking into account bandwidth constraints, and adapts the traffic rate of the services and the reservation itself by means of the *autobandwidth* algorithm [136] thanks to the information gathered by a monitoring system. Furthermore, the solution also supports flow preemption, that is, the relocation of an already installed LSP into an alternative path to make room for higher priority traffic.

The same author [135] outlines some of the benefits of utilizing SDN to provide TE. First, SDN eliminates the need for distributed protocols and enables the aggregation of multiple services to be treated jointly. Second, with SDN it is possible to have application-aware routing. With this approach, different applications can be in charge of the routing needs of different services with different requirements. For instance, Voice over IP (VoIP) requires low latency paths, while a file transfer does not. Third,

as mentioned before, it is possible to adapt the reserved bandwidth for a given service taking into account the actual use of the network resources. And finally, the logically centralized control enables unified failure recovery.

- *Conclusions*

The solution proposed by Das *et al.* is interesting because it merges an MPLS data plane with an OpenFlow control plane. It is worth mentioning how they rely on the feedback of a monitoring system to optimize the network resource consumption. However, the algorithms used for TE are not explained in detail, making difficult to assess the overall suitability of the solution. The framework to support TE is also very basic, without explaining in detail the functionalities carried out by each of the modules, although it does consider most of the required elements for a generic TE framework. Regarding the support for advance reservation, this solution does not currently provide the necessary mechanisms.

2.2.6 OFVN

As stated by Gharbaoui *et al.* [137], even if network virtualization techniques provide important advantages to DC networks, they also raise new challenges regarding DC infrastructure management, specially since operations such as Virtual Machine (VM) provisioning and reconfiguration occur much more frequently than in legacy DCs. In such a context, they present an approach that takes into account the current occupation of DC network links when selecting the server in which to allocate a virtual machine.

In fact, in a DC network with a typical tree-like topology, the selection of a server and the selection of the corresponding network path are very tightly connected. That is, the selection of one of these parameters heavily constraints the options available for the selection of the other. For this reason, the authors present two selection algorithms, which basically differ in the order in which resources are selected, that is, in one of the algorithms the server is selected first, whereas in the other algorithm the network path is selected first. For each type of resource, the authors

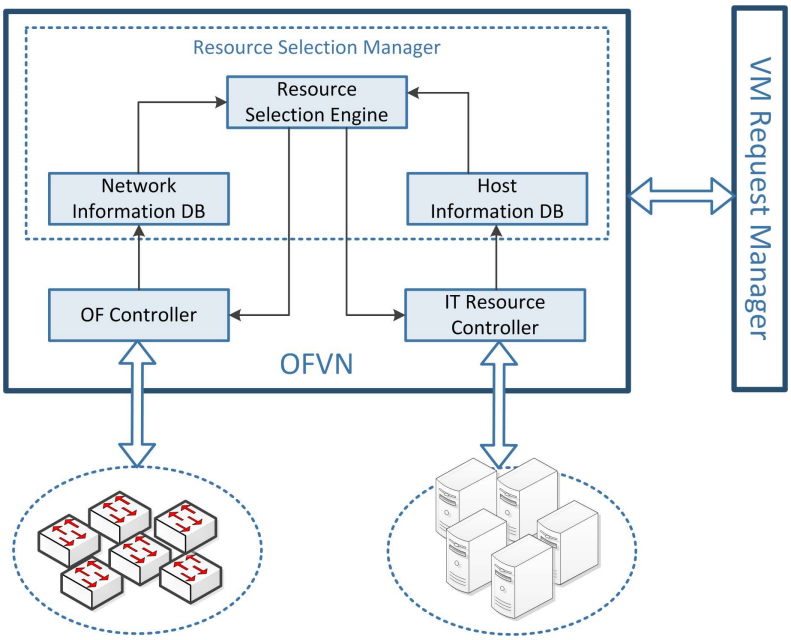


Figure 2.12.: OFVN architecture [138].

propose three selection policies: select the first one with enough available resources, select the most unloaded one or select the most loaded one with enough available resources. In order to implement this approach, the authors propose to use OpenFlow and the flow-level information obtained through statistics messages.

The results obtained through simulation show that the proposed approach provides a lower blocking probability than a solution that allocates servers in a random fashion without taking into account network conditions.

- *Design*

The architecture of the solution implemented by Gharbaoui *et al.* for the DC environment is described in [138]. The OpenFlow-based Virtualization-aware Networking (OFVN) architecture is depicted in Figure 2.12, and consists of the following modules:

- **VM Request Manager**: Entity that handles VM allocation requests.

- **Resource Selection Manager**: Comprised of a *Resource Selection Engine* in charge of the path computation and the server on which the VM needs to be allocated, a *Network Information Database* that holds information about the global state of the network and per-flow monitoring statistics and a *Host Information Database* that holds information about the status of the servers.
- **OpenFlow Controller**: In charge of programming the flow tables of the OpenFlow switches.
- **IT Resource Controller**: In charge of placing the VMs in the servers.

As can be deducted, the solution proposed by Gharbaoui *et al.* is an implementation of the Resource Selection Engine within the OFVN architecture. In a nutshell, once a new VM is requested through the VM Request Manager, the Resource Selection Manager checks the IT and network resource availability and selects the most appropriate server or the less congested path to allocate the VM.

- *TE Support*

Gharbaoui *et al.* present two different approaches to select the optimal server to allocate VMs inside a DC. The first approach is called *Server-Driven* while the second one is called *Network-Driven*. As their names suggest, they differ on which resource is selected first. The type of resource that is selected first also affects the selection of the remaining elements that interconnect the VM, and therefore, has an impact on the network performance that is achieved.

On the one hand, the Server-Driven algorithm selects the server in which the VM needs to be placed, and it is considered as a candidate server until the network resources involved to exchange traffic to and from the VM are checked. This is done by means of an iterative process, by selecting the switches one by one and checking if the links are not congested.

On the other hand, the Network-Driven algorithm selects the OpenFlow switches, following an iterative process as well. In each step, the network congestion of the outgoing links is evaluated, and the next OpenFlow

switch is selected among those reachable through non-congested links. Once the path is computed, the server is selected among the ones that are reachable from the last OpenFlow switch. It is worth noting that the network load in both algorithms is evaluated using OpenFlow statistics, which are gathered by the OpenFlow controller.

- *Conclusions*

Although Gharbaoui presents a valid solution for the selection of paths inside DC networks, the algorithms are tightly coupled to the fat-tree topology that is used in the DC, and the solution could not be applied outside the DC context. In addition, the framework that they propose is also very dependent of the use case, and it cannot be considered a generic framework for the application of TE. Nevertheless, it is worth mentioning how the OpenFlow statistics are used to facilitate the selection of the next OpenFlow switches, which enables the minimization of the network congestion and allows to achieve higher network resource utilization.

2.2.7 IDEALIST

Elastic Optical Network (EON) networks [139] allow the dynamic configuration of the physical parameters of the network in order to provide flexible connection-oriented circuits and enhance the overall network capacity. In such a context, the FP7-IDEALIST project [125] proposes a solution based on the Application-Based Network Operations (ABNO) architecture, defined in RFC 7491 [140], that allows to provide multi-domain connectivity services in flex-grid optical networks [141].

This project proposes the utilization of hierarchical PCEs to achieve multi-domain service provisioning. In a nutshell, a Child Path Computation Element (cPCE) is in charge of the path computation in each of the domains, also known as intra-domain path computation, whereas a Parent Path Computation Element (pPCE) is in charge of selecting the domains that will facilitate the connectivity, also known as inter-domain path computation. In such a scenario, BGP-LS is used to exchange TE information between the cPCEs and the pPCE. With this architecture, several proposals have appeared dealing with the

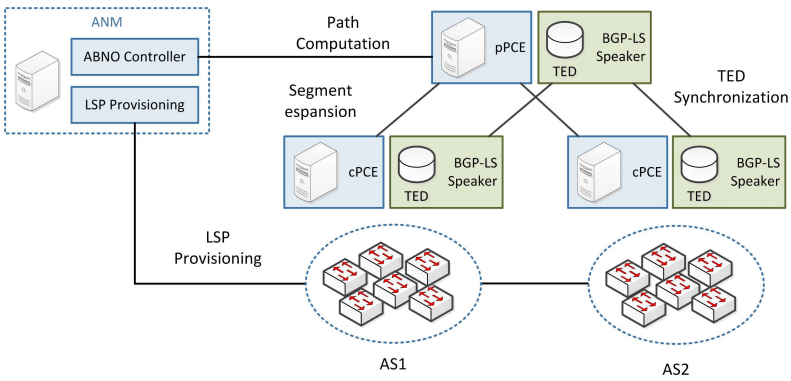


Figure 2.13.: Architecture proposed by the FP7-IDEALIST project [142].

optimization of network resources utilization in flex-grid optical networks, which are further described in the next subsections.

- *Design*

The architecture proposed in the IDEALIST project is depicted in Figure 2.13, and it consists of the following modules:

- **pPCE**: In charge of the domain selection. The BGP-LS speaker retrieves the inter-domain TE information gathered from the BGP-LS speakers of the cPCEs that it aggregates.
- **cPCE**: In charge of the path computation in the domain or AS that it controls. The BGP-LS speaker placed next to each cPCE retrieves the TE information of the domain and exports it to the BGP-LS speaker of its pPCE.
- **ANM**: The Adaptive Network Management (ANM) is comprised of an *ABNO controller*, also referred to as SDN controller, and a *LSP provisioning* module. The first one tracks the network resource utilization to enable the adoption of an stateful-PCE approach whereas the second one uses RSVP-TE to instantiate the LSPs in the network.

It is worth noting that in [143] the solution was extended to deal with transport technologies and control plane heterogeneity in a unified

manner. The proposal integrates OpenFlow and flex-grid networks, using the same hierarchical PCE architecture, but relying on an OpenFlow controller to handle the topology discovery and service provisioning in the OpenFlow domains, instead of the ANM.

- *TE support*

The FP7-IDEALIST project has proposed different techniques and solutions to improve network resources utilization and to ease path computation in multi-domain scenarios. For instance, in the work carried out by Casellas *et al.* [143], the cPCEs of each domain implement a C-SPF algorithm. In the case of the flex-grid domains, this algorithm allows to find the shortest path using bandwidth and spectrum availability constraints. Furthermore, the algorithm takes into account aspects such as spectrum continuity through the different segments of the network. On a first stage, they identify the resources that would satisfy the demand, and they apply the Dijkstra algorithm over the pruned topology.

In addition, Cuaresma *et al.* [144] have evaluated the performance of the path computation procedure depending on the amount of topology information available in the pPCE. The authors compared two algorithms to determine which strategy is more efficient for the E2E path computation. In the first one, only the TE information necessary to perform the inter-domain path computation is sent to the pPCE. Whereas in the second one, all the TE information is sent. In the second case, every time a path is requested to connect two end-points belonging to the same domain, the path computation is performed by the cPCE of that domain. On the contrary, when the two end-points belong to different domains the path computation is performed at the pPCE, which is aware of both the inter-domain and intra-domain topologies of each domain. In this later case, the path computation is performed a 20% faster compared to the first case.

Finally, Martínez *et al.* [145] propose the utilization of two PCEs to optimize the network resource consumption inside a domain. In this solution, the PCEs are in charge of executing a Routing and Spectrum allocation algorithm to compute the physical route and the frequency slot that each LSP will use. In order to re-optimize already installed

LSPs (a time and resource consuming task), they propose the utilization of a front PCE and a back PCE, the first one in charge of the algorithm execution for new and restored LSPs and the second one in charge of the re-optimization of already installed LSPs running in the background. Having this architecture requires both PCEs to be coordinated, which is achieved using BGP-LS for TED synchronization and PCEP to exchange information about the LSPs. In case of a failure affecting an established LSP, the front PCE computes another path and once it is setup, it communicates with the back PCE to inform about the new TED and to request a re-optimization of the affected LSPs.

- *Conclusions*

The architecture proposed by the FP7-IDEALIST project is an interesting approach to deal with multi-domain service provisioning involving heterogeneous networks. The techniques utilized by the PCEs in order to enhance the network resource utilization include the application of C-SPF that takes into account spectrum and bandwidth information to select the optimal path. Furthermore, they also consider the utilization of secondary PCEs in charge of LSPs re-optimization operating in the background with support for network failures.

In addition, they also explore different alternatives to deal with multi-domain path computation. Notwithstanding, according to this author's point of view, even if the approach is proven to be more efficient, it is against the fundamentals of hierarchical PCE, since the pPCE computes the entire multi-domain path without involving the cPCE. Moreover, they do not present a generic framework for TE in SDN.

2.3 TE FRAMEWORKS FOR CONGESTION MINIMIZATION

Traditionally, the minimization of the network congestion has been the preferred TE objective function. The problem has been widely studied in MPLS networks [146–149], where the utilization of traffic splitting mechanisms is a common choice.

With SDN, the high programmability and the high granularity available at the networking elements opens the gate for new strategies aiming to minimize the network congestion. For instance, Braun *et al.* [150] propose a mechanism to distribute the load in case of network congestion through pre-computed disjoint backup paths. Other solutions coordinate with the end-hosts to customize the TCP timers in order to avoid the utilization of congestion avoidance mechanisms [151], while solutions like the one proposed by Kim *et al.* [152] rely on machine learning approaches to reduce congestion. However, although these proposals present novel TE solutions to minimize network congestion, they do not define TE frameworks for SDN.

Consequently, in this section a comprehensive set of frameworks that facilitate TE in software-defined networks is presented, whose main objective function is the minimization of the network congestion.

2.3.1 HUANG *ET AL.* PROPOSAL

Grid computing requires high-speed data transfers between the different sites where the nodes reside. A very common approach to satisfy such need is by means of GridFTP [153], where multiple TCP streams are used to mitigate the effects of the congestion avoidance mechanism of TCP and reduce the packet losses. However, the traditional routing protocols do not take into account information from the application layer, which results in all the TCP streams being provided through the same path.

In order to solve such limitation, Huang *et al.* propose an SDN-based solution for GridFTP [154] that relies on the OpenFlow protocol to route different TCP streams along different paths between the given endpoints. More specifically, the authors propose to build an OpenFlow controller that pre-computes a fix number of available paths between the source and destination nodes using Breadth First Search (BFS). Then, the OpenFlow controller installs the appropriate flow entries in the OpenFlow switches in order to divide the TCP streams uniformly through the previously computed paths. With this approach, GridFTP is able to improve the data transmission time by using multiple parallel TCP streams.

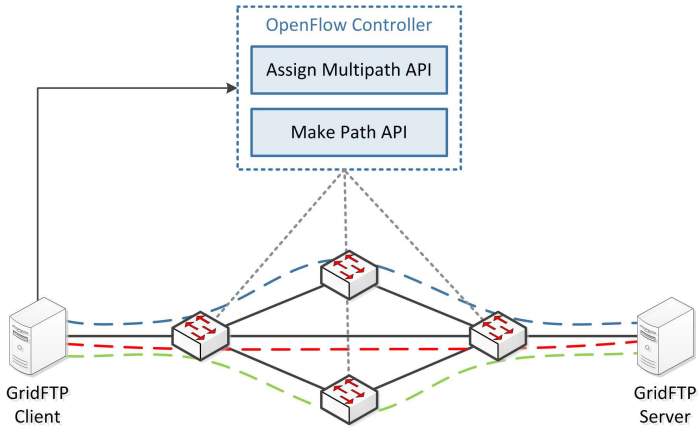


Figure 2.14.: Architecture proposed by Huang *et al.* [154].

- *Design*

The OpenFlow controller proposed by Huang *et al.* to support SDN-based GridFTP is illustrated in Figure 2.14. It consists of two applications accessible through an API that are described below:

- **Assign Multipath**: Application in charge of pre-computing a set of possible paths between a given pair of source and destination nodes. It exposes an open API called *Assign Multipath API* that is used by the GridFTP client when a high-speed data transfer is about to begin.
- **Make Path**: Application in charge of assigning a specific path to a TCP stream. It exposes an open API called *Make Path API* that is used by the GridFTP client when a new TCP session is about to begin and the ports that are going to be used to differentiate that session are known. Once the Make Path application knows the ports, it generates the necessary flow entries for the OpenFlow switches along the path and installs them.

- *TE support*

In this proposal, the authors are able to achieve higher data transfer speeds by utilizing a quite simple TE mechanisms. First, they alleviate

future path computations by generating a set of possible paths between the source and destination nodes that will involve the data transfer. To that matter, they use a BFS algorithm, and store the paths for their later use arranged from shortest to longest ones. Once the GridFTP client wants to use a specific port to setup a TCP connection, the client sends the port information to the OpenFlow controller. At that moment, the shortest available path is selected among the ones pre-computed beforehand and it is installed in the network.

With this approach, they are able to reduce the time required to finish the data transfer by the number of parallel paths used to connect the two end-points. That is, in Figure 2.14, given that there are three parallel paths between the GridFTP client and the GridFTP server, the data transfer is achieved three times faster compared to when a single path is used.

- *Conclusions*

The solution presented in this section is a clear example of the benefits that SDN can bring to TE. In this case, the improvement is achieved because the granularity available at the data plane allows forwarding the traffic based on information from the application layer. All in all, the solution is able to reduce the time required to complete the data transfers between two endpoints. However, the solution could be enhanced with additional TE mechanisms such as fault recovery or make use of more sophisticated routing algorithms. In addition, the solution does not provide a generic framework for TE that could be used in other use cases.

2.3.2 LI *ET AL.* PROPOSAL

The solution proposed by Li *et al.* [155] directly handles congestion minimization by applying a C-SPF algorithm. They propose to use OpenFlow border routers in order to connect IPv4 and IPv6 islands in an efficient manner. As a consequence, they are able to eliminate the necessity of dedicated equipment in charge of the interoperability between islands of different types.

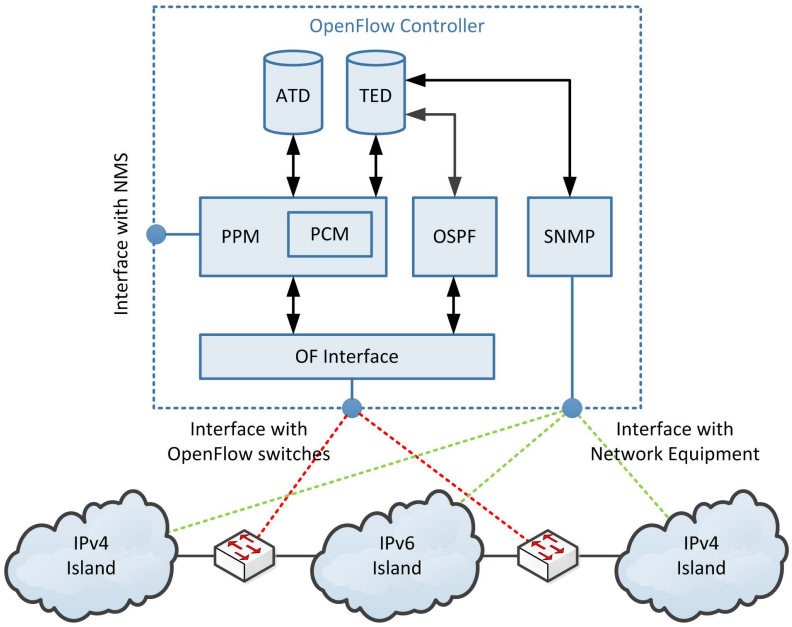


Figure 2.15.: Architecture proposed by Li *et al.*[155].

In order to compute the optimal path, they monitor the traffic load using legacy protocols such as OSPF and Simple Network Management Protocol (SNMP) for the legacy IP islands. With this approach, they are able to reduce the overall congestion of the network, which they demonstrate by injecting video traffic and comparing the quality of the image received.

- *Design*

The architecture proposed by Li *et al.* to provide flexible TE is shown in Figure 2.15, and it is comprised of the following modules:

- **ATD**: The Address Translation Database (ATD) stores the IP addresses that facilitate the interconnection of different islands. That is, it allows to perform the translation from IPv4 addresses to IPv6 addresses and *vice versa*.
- **TED**: The TED stores the TE information regarding the network status as informed by the SNMP module.

- **PPM**: The Path Provision Module (PPM) is the entity in charge of the provisioning of the paths. In other words, it translates paths into OpenFlow messages. In addition, it manages the ATD and the TED.
- **PCM**: The Path Computation Module (PCM) is the element in charge of the computation of the paths taking into account the network state information contained in the TED.
- **OSPF**: A single instance of the OSPF module is used for all the IP routers of the islands. It is used to retrieve the network topology information, which is later fed to the TED.
- **SNMP**: The SNMP module allows to retrieve link state information from the IP routers, which is later stored in the TED.
- **OF Interface**: Allows the communication through the OpenFlow protocol with the OpenFlow switches.

- *TE support*

The OpenFlow routers are connected to a centralized controller, which has an overall view of the network topology and state. For each incoming flow, the OpenFlow routers contact the centralized controller to compute the optimum path according to the current network conditions. As mentioned before, the algorithm that they use is C-SPF, which takes into account the traffic load on the current network to select the optimal path. Once the path is computed, the controller programs the OpenFlow routers so that the traffic is forwarded along the computed path, encapsulating IPv4 packets in IPv6 or *vice versa* as needed. This way, the controller is able to select paths that reduce the E2E delay and the network congestion, compared to alternatives that implement TE mechanisms using dual stack routers or pre-configured static IP tunnels.

- *Conclusions*

On the one hand, this solution stands out because the OpenFlow switches are used both to direct the traffic to and from less congested

islands and because it eliminates the need of dedicated equipment in charge of the translation between IPv4 and IPv6 domains. In addition, the solution integrates support for multiple protocols in the same controller, where OSPF and SNMP are used in conjunction with OpenFlow.

On the other hand, the architecture that they propose is tightly related to the use case, and it is technology dependent. However, it is worth noting that they outline the need for the joint utilization of monitoring tools, automated topology discovery mechanisms and path computation to achieve TE.

2.3.3 BAATDAAT

In order to reduce congestion in DC networks as a consequence of greedy algorithms that overload the best possible paths, *BaatDaat* [156] proposes the utilization of TE strategies that leverage the centralized nature of SDN. The authors of this solution propose a real-time traffic monitoring tool embedded in the hardware devices that sends information about the network resource utilization to an OpenFlow controller. Then, the monitoring information is processed to build an updated topology-wide network utilization map. With this information, a scheduling algorithm implemented in the controller plane computes the optimal paths to avoid network congestion, even if that implies selecting detour paths. The support of non-SPF algorithms for the selection of the paths is of interest to this topic because there are cases in which the optimization function is not significantly affected by the number of the traversed hops. With this solution, the authors have achieved an improvement of an 18% of overall network utilization compared to ECMP.

- *Design*

Unlike the other architectures previously reviewed in this chapter, the architecture of BaatDaat is comprised of elements both in the data plane and the controller plane. As illustrated in Figure 2.16, at the data plane a module in charge of monitoring the real-time occupation of the switch links is included, called *Link Measurement*. This module computes the link utilization by adding the bytes of each packet that comes through

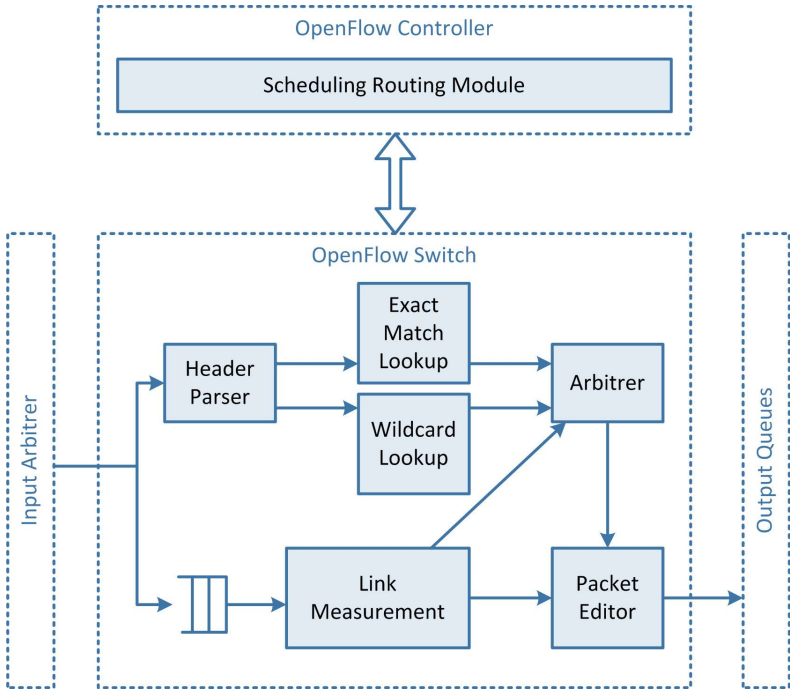


Figure 2.16.: BaatDaat architecture [156].

the link. In addition, it also computes the utilization of the outgoing link after the packet is processed by the output port.

On the other hand, at the controller plane a specific module is embedded in the OpenFlow controller, named *Scheduling Routing Module*, which is in charge of computing the paths that allow to balance the load inside the DC network.

- *TE support*

The approach followed by BaatDaat aims to reduce network congestion by finding detour paths that can help alleviate the greedy utilization of some of the links in the network. It relies on real-time measurements and a mixed scheduling scheme that is both applied at the data plane and the controller plane. On the one hand, the monitoring of the links utilization is performed at the OpenFlow switches, and it is used to send the new arriving flows through the less utilized link. When the same level

of utilization is reported in all the outgoing links, the ECMP protocol is used to transmit the flow packets through all the links randomly.

When the Link Measurement module detects congestion in any of the links, it requests a detour path to the controller. More precisely, it notifies the controller when any of the shortest path links is above its 30% of utilization. It is worth noting that in order to simplify the scheduling procedure, they limit the detour paths to the network segment between the Top of Rack (ToR) and the aggregation switches in a fat-tree like topology. Furthermore, they only admit detour paths two hops longer than the shortest path, in order to prevent oscillation and unnecessary network resource consumption.

- *Conclusions*

BaatDaat proposes a flow scheduling algorithm that is only triggered when the link utilization is above a certain threshold. Unless congestion is detected, a modified OpenFlow switch with real-time monitoring capabilities is able to balance the load without involving the controller. This approach allows to reduce the traffic sent to the OpenFlow controller, which in a DC network with a flow inter-arrival time in the order of *ms* can result in the congestion of the control channel. In addition, they demonstrate that the utilization of non-shortest paths can actually help minimizing the network congestion, just using detour paths that are only 2 hops longer that the shortest ones.

However, the solution does not take into account the type of traffic being send. For instance, even if the detour paths are only 2 hops longer, this can be unacceptable for real-time applications with a high sensitivity to E2E delay. Secondly, the solution depends on the real-time monitoring module that they have embedded in a Field-Programmable Gate Array (FPGA)-based OpenFlow switch. This makes difficult the adoption of the solution, since most switch vendors do not provide the means to include such modules in their products. In addition, they do not provide a generic framework for SDN that would suit the needs of other use cases and do not support advance reservations.

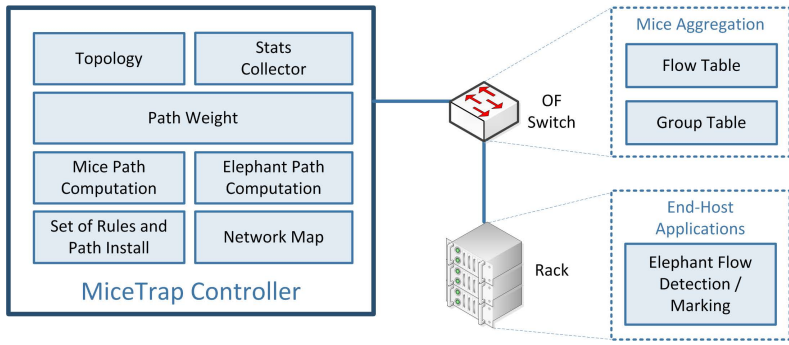


Figure 2.17.: MiceTrap architecture [157].

2.3.4 MICETRAP

As stated by Trestian *et al.* [157], the most common approach to apply TE in DC networks is to distinguish between long-lived flows, known as elephant flows, and short-lived flows, known as mice flows. Usually, the 10% of the flows in a DC network are elephant flows, but they carry the 80% of the traffic. In such a context, TE strategies are only applied to elephant flows, while mice flows are routed according to baseline routing methods.

Although this approach facilitates the scalability of TE strategies, it might also cause congestion to mice flows, which can correspond to critical network traffic. As a consequence, Trestian *et al.* propose MiceTrap, an approach to extend TE to mice flows without hindering the scalability. The main idea behind MiceTrap is to leverage the flow aggregation capacity provided by OpenFlow to handle a number of mice flows together. Then, it applies a weighted routing algorithm that uses the actual link utilization information to compute the splitting ratios that improve the balance of the mice flows.

- *Design*

Figure 2.17 depicts the architecture of MiceTrap, which is comprised of the seven modules described below:

- **Topology Block**: Module in charge of storing the topology information, that is, the nodes and links that comprise the network.

- **Stats Collector Block**: Retrieves per-link statistical information from the switches in order to provide information to the path computation algorithms about the real link utilization.
- **Path Weigth Block**: Module that computes the paths.
- **Mice Path Computation Block**: Computes the paths for the mice flows.
- **Elephant Flow Path Computation Block**: Computes the paths for the elephant flows.
- **Set of Rules and Path Installation Block**: Takes the computed paths and translates them into OpenFlow rules that are sent to the involved nodes.
- **Network Map Block**: Retrieves network state information, such as the current traffic matrix or the routes that are already installed.

- *TE support*

MiceTrap applies multiple TE strategies to reduce the congestion in DC networks. It differs from other solutions for TE in DC in that it tries to optimize the path selection for mice flows and not only for elephant flows. MiceTrap applies two different path computation algorithms depending on the type of flow that needs to be routed. However, the proposal is focused on the path computation algorithm that is applied to mice flows.

First, in order to reduce the amount of flow entries necessary to route the mice flows in the OpenFlow devices, it aggregates multiple mice flows. That way, they use a less granular flow match description, such as the destination IP address, that is matched by the entire set of aggregated mice flows. Then, the aggregated flow is treated in the group table, where the load is balanced among multiple shortest paths able to connect the given source with the given destination node.

Instead of using ECMP to share the load, the Mice Path Computation Block computes a set of possible shortest paths between the source and destination. Then, with the statistical information retrieved by the Stats Collector Block and the topology information available at the Topology module it computes the weights that need to be assigned for each of

the paths, which depend on the current load. Once the weights are computed, the action buckets that conform the group entry assigned to the aggregated mice flow are updated with the weights. As a consequence of this approach, not all the paths valid for the aggregated flow carry the same amount of traffic, achieving a better utilization of the network resources and reducing the congestion.

- *Conclusions*

As in the previous case, one of the limitations encountered by this solution is the need to install additional software in the OpenFlow switches. More precisely, it is necessary to identify which flows are elephant or mice flows. Depending on the vendor, this may not be an option, which may result in additional equipment being necessary. In addition, the solution only computes the set of shortest paths to share the load among the aggregated mice flows. Even if they are able to minimize the overall congestion of the network, it could be interesting to measure the behavior of the solution when non-shortest paths are also used.

On the other hand, the architecture of the MiceTrap controller is one of the most complete frameworks. It includes monitoring modules, topology discovery and abstraction modules and PCEs. Although the path computation modules are use case dependent, it represents a good example of how a generic framework for TE should be.

2.4 TE FRAMEWORKS FOR PACKET LOSS MINIMIZATION

As outlined in Chapter I, the packet loss minimization is an objective function that can be achieved as a result of the minimization of the congestion, but it is also a TE objective function on its own. Generally, when the packet loss is not a consequence of the network congestion, it is caused by the failure of a node or a link in the network.

With that premise in mind, packet loss minimization is a well-addressed objective in SDN by means of techniques aiming to improve network resilience. For instance, there are proposals dealing with fault restoration in ForCES routers [158], where FE redundancy is achieved by means of

cloned virtual machines. Other solutions deal with network resilience in WANs, as in [159], where the authors propose a flow relocation mechanism to minimize service disruption in case of a natural disaster. On the other hand, the solution proposed by Pisa *et al.* [84] minimizes the packet loss in DC networks by re-configuring the forwarding tables of the hypervisors in DC networks in order to provide seamless VM migration.

In such a context, and once again, the high programmability and the logically centralized control plane of SDN are the key features leveraged by these solutions, since they facilitate the rapid re-programming of the network elements in order to adapt to the changing conditions. Among the available proposals dealing with failure recovery in SDN [160–164], the solutions that are described in this section have been selected because they define a framework or an architecture to minimize the packet loss in SDN.

2.4.1 QNOX

QoS-aware Network Operating System (QNOX) [165] is an architecture that aims to provide QoS and facilitate the introduction of TE mechanisms in order to optimize the performance of hybrid networks. It is defined using the ForCES framework, although it considers the possibility of its application to both legacy networks and OpenFlow networks. In the latter case, OpenFlow would be the chosen protocol to be used in the ForCES interface, as stated in [166].

Using this architecture, Jeong *et al.* [167] have implemented a multi-layer fault restoration mechanism using ForCES. Their proposal is a scheme that relies on the fault restoration capabilities of different layers. It uses the fast fault detection mechanisms of the physical layer, some of the classic TE strategies available at the MPLS layer and the hierarchical priority-based resource sharing available at the IP layer.

- *Design*

The architecture of QNOX is illustrated in Figure 2.18 and it has been designed to operate both with OpenFlow devices and legacy equipment. To that purpose, it exposes a ForCES interface to communicate the FE

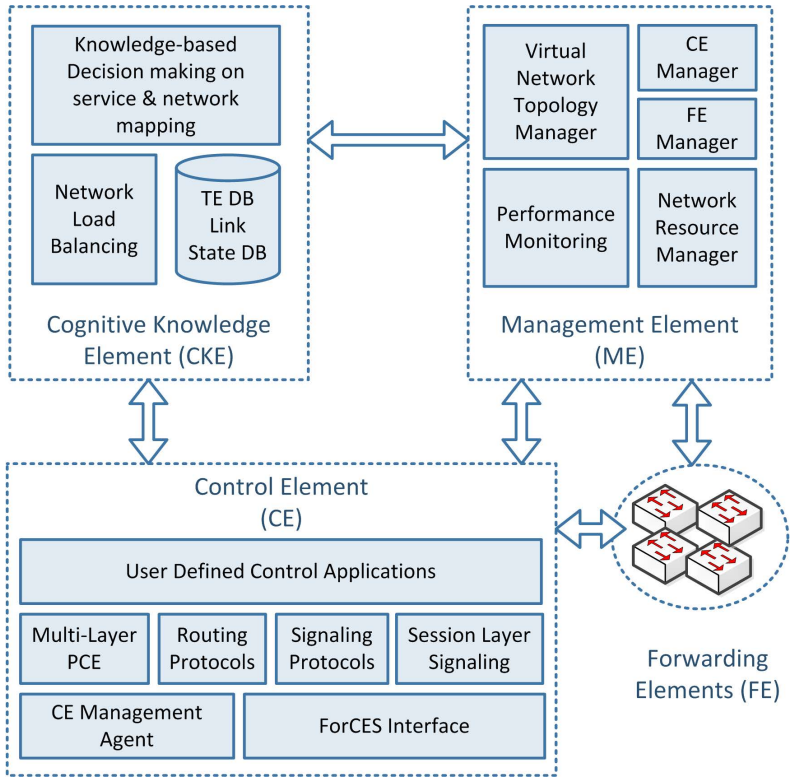


Figure 2.18.: QNOX architecture [165].

with the CE, which in conjunction with a Cognitive Knowledge Element (CKE) is able to provide QoS and TE in hybrid networks.

In this architecture, a Service Element (SE) receives a new service request and checks whether it can be provided or not with the required QoS and fault restoration capabilities. In order to determine whether the service can be provided or not, the PCE or the legacy routing protocols are used to compute the paths, depending on the underlying transport network technology. These modules are executed using the information available at the Management Entity (ME), which is the entity in charge of retrieving the network resources and handling the virtualized topology. In addition, the CKE maintains a link state database and a TED, which can be used to apply load balancing and other TE techniques to optimize the network resources utilization.

- *TE support*

As pointed out in [168], fault restoration in transport networks requires detecting the failure, reacting to the failure, notifying the neighbors or the controller in the case of SDN, computing an alternative path and installing the path in the networking elements. As it can be deducted, all these phases have an impact on the service disruption time and therefore, the experienced packet loss.

In order to minimize the packet loss as a consequence of a network failure, Jeong *et al.* have proposed a multi-layer approach that they have implemented in QNOX. In a nutshell, they rely on the fault restoration capabilities available at different layers of the protocol stack. As such, once a new service is requested, they compute both the primary LSP and the secondary LSP to be used in an MPLS network taking as input the global state of the network. As such, they keep an up-to-date view of the network in the CE, information that they use to perform the path computation. In order to compute the backup path, once the primary path has been computed, they remove all the resources included in the primary path and they execute the algorithm again. With this approach, they ensure that the backup path does not share any resource with the secondary path, that is, they ensure that the primary and the secondary path are totally disjoint paths, as long as the network supports it.

Then, in the event of a network failure, they rely on the FE itself to detect the failure, and switch the data transmission to the secondary LSP. They compare their solution with the failure recovery capabilities of distributed OSPF, and they conclude that with this mechanism, they are able to reduce the service disruption time in an order of magnitude in order to be compliant with the carrier grade requirements.

- *Conclusions*

The solution proposed by Jeong *et al.* is proven to be useful compared to the classical approach followed in distributed OSPF where the convergence time of the algorithm is not enough to satisfy the carrier grade requirements. Although the novelty of their solution relies on the use of a path computation element in the controller plane to compute

both the primary and the secondary LSPs taking into account the global state of the network, the solution uses the same failure recovery approach that is already used in MPLS-TE. That is, both LSPs are installed in the network, even if that means that only the 50 % of the reserved resources will be used at any given time. When a failure is detected by the network element, the secondary LSP gets active, without involving the controller or without requiring the computation of another path.

The framework presented in this section is heavily constrained by the ForCES framework, including legacy protocols that for instance with OpenFlow would not be necessary. Moreover, the solution does not include mechanisms to support advance reservations.

2.4.2 YOON *ET AL.* PROPOSAL

As mentioned before, one of the key elements in failure recovery is detecting that a failure has occurred. Multiple techniques can be used for that matter. For instance, a very common approach in SDN is to let the switch notify the failure to the controller, so that the controller can react upon that event. Notwithstanding, active protocols such as Bidirectional Forwarding Detection (BFD) can be used as well, which is precisely what Yoon *et al.* propose in [169].

In this proposal, BFD is included in the ForCES architecture to detect network failures in DiffServ-aware MPLS transport networks. The solution aims to deliver broadband real-time services with QoS. In such a context, BFD is used to detect link failures and make performance measurements, which are controlled by the ForCES CE. The CE activates BFD for each LSP and receives information about the link failures and the measured performance in return. This information is later used to compute the paths using a C-SPF algorithm.

- *Design*

The architecture proposed by Yoon *et al.* follows a hierarchical approach, in which a dedicated CE is in charge of controlling the interfaces of the FE with BFD support. In addition, a centralized controller handles

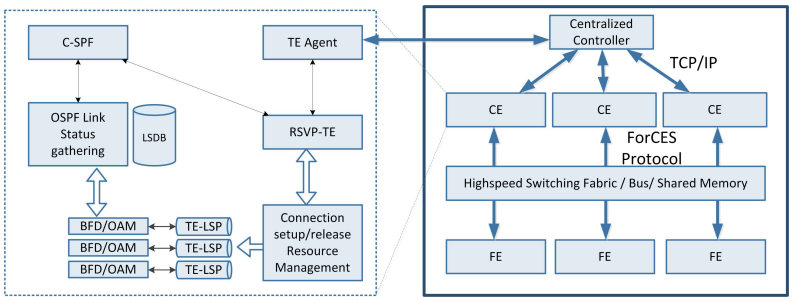


Figure 2.19.: Architecture proposed by Yoon *et al.* [169].

the routing functionalities, as depicted in Figure 2.19. The modules of the CE elements of this architecture are described below:

- **C-SPF**: As it name suggests, it is in charge of applying the C-SPF algorithm, with the information available in the Link Status Database (LSDB).
- **LSDB**: Stores the information regarding the status of the link, including its availability and the utilization.
- **BFD/OAM**: They monitor the status of the LSPs.
- **RSVP-TE**: Handles the setup and release of the LSPs and their configuration, such as the bandwidth or wavelength allocation.
- **TE agent**: Communicates with the central controller, also known as the TE manager, to setup the desired QoS policy.

- *TE support*

In this architecture, the TE support is distributed among the centralized controller and the CEs associated to each interface. The centralized controller is in charge of the overall routing and switching capabilities. However, the link state control is kept in the CEs near the interface.

This approach allows a faster network failure detection, since in addition to the utilization of an active protocol such as BFD, the CE is kept near, reducing the latency that notifying a remote controller would

impose. Furthermore, at each CE there is also a module in charge of the path computation, which also improves the efficiency of the solution.

They are able to keep the solution compliant with the carrier grade requirements by sending BFD packets every $5 - 10\ ms$. Once the system detects that three BFD messages have been lost, it notifies the failure in order to establish an alternative path.

- *Conclusions*

The hierarchical approach followed by Yoon *et al.* is interesting for two reasons. First, it uses BFD to detect the network failures, which reduces the detection times considerably, making the solution compliant with the carrier grade requirements. Second, they use a dedicated CE element for every interface, allowing a faster response in case of failure since no additional delay is introduced.

However, the solution still relies in legacy protocols, presenting therefore, the limitations described in Chapter I. In addition, each CE has its own C-SPF module operating over a local copy of the LSDB. Although there is a centralized controller in charge of the TE management, they do not specify how that is handled. In case of a simultaneous failure in two links, it could be possible to have inconsistent topology views in these local copies, which could lead to an invalid path computation.

As in previous solutions, even if it defines a suitable architecture for TE in SDN, the solution is use case dependent and therefore it could not be applied to other use cases.

2.4.3 PHEMIUS *ET AL.* PROPOSAL

The architecture proposed by Phemius *et al.* [170] aims to provide TE in WANs. The solution relies on a monitoring module that keeps track of the network utilization and detects link failures in order to redirect the traffic to less congested paths and react upon network failures. For that purpose, they propose an architecture for TE that prioritizes critical flows, allowing the preemption of already provisioned flows with a lower priority.

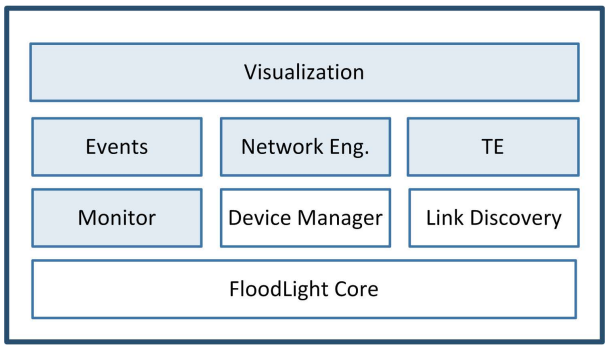


Figure 2.20.: Architecture proposed by Phemius *et al.* [170].

In order to monitor the network, they rely entirely on the capabilities provided by OpenFlow. That is, they monitor the utilization of the network by polling the switches for per-flow statistics and they detect the network failures upon switch notification.

- *Design*

The architecture proposed by Phemius *et al.* is depicted in Figure 2.20. It extends the FloodLight controller with the five custom modules marked in blue that enable the utilization of TE strategies in OpenFlow networks.

- **Monitor**: This module is in charge of monitoring the network status, which includes retrieving bandwidth consumption and latency calculation.
- **Events**: This module triggers alarms when a network failure occurs or when a network resource usage exceeds a certain threshold.
- **TE**: It is in charge of selecting the paths for the flows taking into account their priority and the QoS constraints.
- **Network Engineering**: It facilitates the programming of the network devices and keeps track of the installed flow rules.
- **Visualization**: GUI that allows the end-users to easily request new service demands and allows to keep track of the consumed resources.

- *TE support*

This solution provides both network resource optimization and packet loss minimization. The main contribution is their monitoring strategy, which unlike other monitoring tools implemented in OpenFlow networks does not only compute the consumed bandwidth, but also computes the latency on a given path. On the one hand, the per-flow bandwidth usage is computed by retrieving the per-flow statistics periodically. On the other hand, the latency is computed by injecting packets from the source node and redirecting them to the controller once they have reached the destination node. By analyzing the time difference between both events using time-stamps, they are able to estimate the latency experienced by the packets through that specific path.

Once they have these QoS metrics available, every time a new flow establishment is requested they compute the cost of each link and they use the Dijkstra algorithm to compute the constrained shortest path. It is worth noting that their TE solution considers the priority of the flows. As a consequence, flows marked as critical are allowed to preempt other non-critical flows, even if it results in the relocation of the non-critical flow or even its removal. This strategy is applied both, when a new flow is requested and when a failure occurs in the network.

In the latter case, once the Events module detects a failure, it prioritizes the relocation of the critical flows into alternative paths. That can result in a non-critical flow being switched to an alternative path. Moreover, it can also result in a critical flow also being relocated to an alternative path, which can lead to a waterfall effect in which a preempted flow forces lower priority flows to use alternative paths.

- *Conclusions*

The solution presented by Phemius *et al.* relies entirely on the monitoring capabilities of OpenFlow. However, monitoring in OpenFlow needs to be carefully handled. For instance, the accuracy of the bandwidth monitoring depends on the polling frequency. The bandwidth is computed by comparing the bits that have matched a certain flow between two instants. However, that measures the average bandwidth, which may differ

from the actual bandwidth usage in each instant in the case of traffic with drastic rate changes. On the other hand, there must be a perfect synchronization between all the elements of the network, including the controller, which is not trivial, since the latency at the control channel can affect the accuracy of the retrieved network state information.

Regarding the path computation, the preemption mechanism that they include may result in a waterfall effect where multiple flows may be relocated unnecessarily. This is a consequence of computing the paths sequentially. The solution could be improved in this case by means of an LP approach, that could optimize the flow placement taking into consideration their priorities.

Finally, the architecture that they propose is well engineered. It includes monitoring and TE functionalities. However, it is dependent on the FloodLight controller and lacks support for advance reservations.

2.5 TE FRAMEWORKS FOR QOE MAXIMIZATION

Since QoE is a performance parameter that depends on both objective and subjective factors, the QoE maximization can be tackled from different perspectives. Nonetheless, for the purpose of this analysis, this section reviews the TE frameworks aiming to maximize the QoE in software-defined networks from the network optimization perspective.

As in the case of the previous sections of this chapter, there are other solutions that deal with QoE in SDN that have not been included in this analysis because they do not present a well-defined framework. More examples of such solutions are described in [171–173]. Hence, a single proposal is analyzed in this section since it is the only one where the architecture of the framework is described.

2.5.1 KASSLER *ET AL.* PROPOSAL

Kassler *et al.* [50] propose an architecture for service negotiation and path optimization in SDNs that seeks to maximize the QoE. The main premise of the solution is that different traffic types may need different optimization functions. For instance, VoIP traffic may need the selection

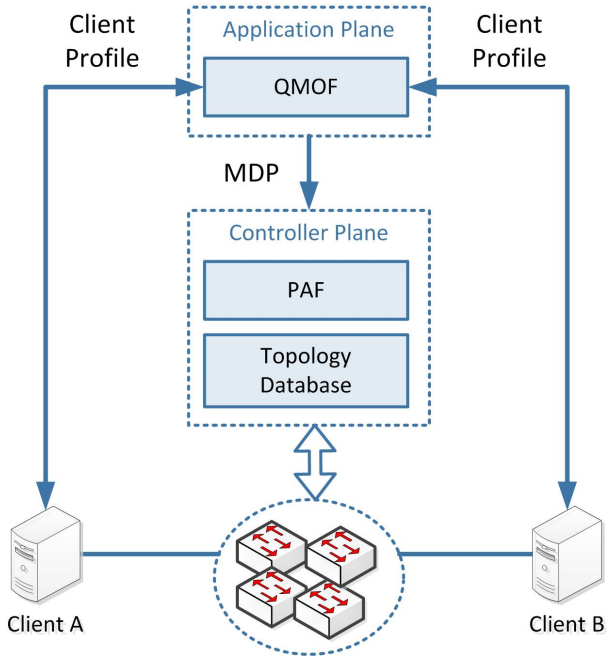


Figure 2.21.: Architecture proposed by Kassler *et al.* [50].

of a path with the minimum delay, whereas a data transfer may require a path with the highest available bandwidth.

In such a context, the solution is based on two key enablers: QoS Matching and Optimisation Function (QMOF) and Path Assignment Function (PAF). The former one resides in the application plane, whereas the latter one resides at the OpenFlow controller and maintains updated information about the flows installed in the network. It also holds a topology database populated with the topological information provided by OpenFlow. The result is an architecture able to reassign paths in order to admit new services.

- *Design*

The architecture proposed by Kassler *et al.* is depicted in Figure 2.21, which relies on the three main components described below:

- **QMOF**: Element that resides in the application plane and is in charge of processing the user requests and generating the optimal service configuration to satisfy the users' demands. This results in

the generation of a Media Degradation Path (MDP) (later explained in this section), which is sent to the PAF.

- **PAF**: Element that resides in the controller plane and computes the optimal paths to provide the service with the demanded QoE by the users using the information contained in the MDP. Once the path is computed, the necessary flow rules are programmed in the OpenFlow devices.
- **Topology Database**: Element that resides in the controller plane and maintains an up-to-date view of the network graph and the active flows in the network.

- *TE support*

One of the most interesting features of this proposal is the generation of MDPs. In order to adapt to dynamic conditions in the network, the output of the QMOF is the generation of multiple configurations for the same service. For instance, if a user requests a video stream, it may prefer to prioritize the quality of the voice over the quality of the video. As a result, the QMOF generates a MDP specifying an ordered set of configuration preferences. In this particular case, the MDP would consist of four possible configurations, the first one with the user requirements as they were specified, the second one with the allowed degradation of the video, the third one with the allowed degradation of the voice and the fourth one with the allowed degradation of both of them.

Once the PAF receives the MDP, it computes the optimal path for the first configuration taking into account the network state information stored in the Topology Database. In case of unavailability of resources to provide the first configuration, the PAF computes in strict precedence the optimal paths for the remaining configurations, until a suitable configuration is found. Furthermore, the PAF also provides the means to relocate already provisioned services into less preferred configurations in order to make room for new service demands that cannot be provided using any of the configurations specified in their MDPs. Regarding the path computation algorithm implemented in the PAF, the solution relies on a variant of the multi-commodity flow problem defined by the same authors in [174].

- *Conclusions*

The solution presented by Kessler *et al.* stands out thanks to the generation of multiple configurations to provide a service, taking into account the user preferences. This approach allows to increase the number of services being provided through the network, considering the degradation of the service that the user considers acceptable. Nevertheless, the solution is also capable of degrading already provisioned services, which may be unfair to the users. The QoE is a subjective parameter and the very same fact of degrading the QoS to a user, even if this degradation was already considered in the original request of the service, may result in the minimization of the QoE.

In addition, the framework provided by Kessler *et al.* would benefit from additional elements, such a monitoring module to check the actual network resources consumption.

2.6 SUMMARY OF SDN-BASED TE FRAMEWORKS

Table 3.1 summarizes the SDN-based TE proposals analyzed in this chapter. The table identifies which is the performance objective used in each proposal, the context in which it is applied, the D-CPI protocol in which it is based and the Path Computation Algorithm (PCA) that is used. In addition, the functionalities supported by the frameworks or architectures defined in each proposal are identified, namely Topology handling (Topo.), Monitoring (Mon.), Resilience (Res.), Load Balancing (LB), Flow Relocation or Re-optimization (FR) and Advance Reservations (AR) support. Please note that in the table the following additional acronyms have been used: CC stands for *Cloud Computing*, FC for *ForCES* and OF for *OpenFlow*.

It is worth remarking that whatever the strategy is to compute the paths, all the analyzed frameworks rely on a dedicated TED to store the topology-related information. Notwithstanding, depending on the scope at which the solution is applied, different sets of functionalities are desirable.

On the one hand, for Inter-DC WAN networks, the two solutions analyzed in this chapter rely on load balancing techniques to optimize

Table 2.2.: Summary of SDN-based TE frameworks

Proposal	Perf. Objective	Context	D-CPI	PCA	Framework					
					TED	Mon.	Res.	LB	FR	AR
Bin	Network resource optimization	CC	FC			✓				
B4	Network resource optimization	Inter-DC WAN	OF	Max-Min fair share	✓		✓	✓		
SWAN	Network resource optimization	Inter-DC WAN	OF	LP MCFP Precomp	✓	✓	✓	✓	✓	
OSCARS	Network resource optimization	REN WAN	PCEP OF	C-SPF	≈		✓			✓
OPEN	Network resource optimization	WAN	OF	C-SPF	✓	✓	✓		✓	
OFVN	Network resource optimization	DC	OF	C-SPF	✓	✓				
IDEALIST	Network resource optimization	WAN	BGP-LS PCEP	C-SPF	✓		✓		✓	
Huang	Congestion minimization	WAN	OF	BFS Precomp	✓			✓		
Li	Congestion minimization	WAN	OF	C-SPF	✓	✓				
BaatDaat	Congestion minimization	DC	OF	non-SPF	✓	✓		✓		
MiceTrap	Congestion minimization	DC	OF	W-SPF Precomp	✓	✓		✓		
QNOX	Packet loss minimization	WAN	FC	C-SPF	✓		✓			
Yoon	Packet loss minimization	WAN	FC	C-SPF	✓	✓	✓			
Phemius	Packet loss minimization	WAN	OF	C-SPF	✓	✓	✓		✓	
Kassler	QoE maximization	WAN	OF	LP MCFP	✓				✓	

their network resource utilization. In addition, both solutions provide failure recovery mechanisms thanks to the logically centralized control plane and the high programmability of SDN.

On the other hand, all the proposals designed for WANs also agree on the necessity of having failure recovery mechanisms. This is an understandable requirement, since NSPs and RENs operating such kinds of networks are subject to strict quality controls by governments and customer associations, and must be able to react to network failures as agreed in the SLAs subscribed with their users. In this regard, it is also worth mentioning that the only framework with advance reservation capabilities is OSCARS, the only one design to operate in a REN.

Finally, it is also clear that in DC networks one of the features in which all the authors agree is monitoring. This may be the result of the fat-tree like topologies usually employed in such networks, where the path computation itself is not very challenging but keeping track of the resources consumption and reacting based on that information can result in a more efficient utilization of the network resources.

2.7 CONCLUSIONS

This chapter has presented a representative set of SDN-based solutions where a framework or an architecture to support TE is defined. The solutions range from architectures aiming to optimize the network resource utilization to solutions dealing with network congestion, packet loss or the maximization of the QoE. They are based on the utilization of D-CPI interfaces such as OpenFlow, ForCES or BGP-LS/PCEP in different network types such as WANs or DCs.

The overall impression after conducting the analysis to the existing frameworks is that the high-programmability of SDN, the possibility to make decisions from the centralized controller using up-to-date information about the network topology or the resources' consumption, and the fact that advance path computation engines can be embedded on the control plane are of great value to improve the performance of the network.

Depending on the objective function they are trying to achieve, the architectures include a different set of modules. However, all of them rely

on a PCE on the controller plane which uses the topology information contained in a topology database. Besides, most of them are not only focused on a single objective function. For instance, many solutions dealing with the optimization of the network resources also provide resilience.

Nonetheless, none of the analyzed solutions present a generic framework for TE that could benefit multiple use cases. In general, the architectural frameworks are tightly coupled to the use case in which they are applied. Moreover, besides the OSCARS framework, none of the solutions support advance reservations, which as stated in the Chapter I, is one of the requirements that the solution presented in this thesis aims to address.

All in all, this analysis has allowed to identify a set of features that a generic SDN-based TE framework should provide, which are listed below:

- The framework must include the necessary mechanisms to retrieve the up-to-date network topology and abstract it to ease the path computation.
- The framework must be flexible enough to make the introduction of novel TE strategies possible.
- The framework must be able to support multiple use cases and scopes.
- The framework must not impose the installation of software agents in the network devices to be easily applied with devices of different vendors.
- The framework must be prepared to react to network failures in order to ensure the service provisioning.
- The framework must allow to take forwarding decisions based on application layer information.
- The framework must provide the mechanisms to retrieve monitoring information from the network.

3 Algorithms and Data Structures for Advance Reservations

"Study the past if you would define the future."

— Confucius

As mentioned in Chapter I, advance reservation mechanisms allow to reserve a set of resources to be exploited in the future, and are of special relevance for RENs.

Supporting advance reservations is not trivial, since the utilization of such mechanisms imposes some challenges. As identified by Wischik and Greenberg in [175], in advance reservation systems the control and responsibility of the service provisioning is moved from the network to the user, who needs to specify in advance the desired characteristics of the traffic to be exchanged. In addition, the reservation of the network resources may lead to a lower network resource utilization if it is not handled correctly, since new service reservations may be rejected when the reserved resources are not actually being used. Furthermore, as noticed by Schneider and Linnert [176], the utilization of advance reservation mechanisms can actually result in a high and undesirable resource fragmentation. That is, that the availability of the resources may vary constantly, resulting in high amounts of different situations with different traffic loads to consider. As a consequence, advance reservation systems need to be carefully engineered.

With the appearance of SDN, solutions to provide advance reservations have appeared during the last years [177–179]. Nevertheless, these solutions are focused on the architectural aspects, or new path computation algorithms that leverage the logically centralized nature of this paradigm, without considering novel data structures specially designed to support the novel TE approaches feasible in SDN.

In such a context, this chapter provides an overview of a representative set of advance reservation solutions where the objective function is the improvement of the network resources consumption taking into account bandwidth and timing constraints. More precisely, this chapter is focused on analyzing the effect of the data structures used to handle the timing constraints in advance reservation systems. Since, as stated by Burchard, *"... the time spent in the data structures in a sample network management system adds up to about 60 percent of the total processing time required by the bandwidth broker, whereas routing requires only about 8 percent and the administrative tasks take the remaining 32 percent of the total processing time."* (in [180], page 1).

This chapter is structured as follows. First, Section 3.1 introduces the fundamentals of advance reservations, where some concepts about algorithms and data structures are also presented. Then, the analyzed solutions are categorized depending on how they deal with the resource management; Section 3.2 analyzes the solutions where the data structures keep information related to the link utilization profiles, whereas Section 3.3 analyzes alternative solutions to deal with the resources management. Finally, Section 3.4 compares the solutions presented in this chapter, while Section 3.5 summarizes the conclusions.

3.1 INTRODUCTION

In order to better understand current solutions dealing with advance reservations, this section presents the fundamentals of these types of systems. In addition, since the focus of this review is to understand the impact of the data structures used to handle the resources utilization, a brief introduction to algorithms and data structures is provided.

3.1.1 FUNDAMENTALS OF ADVANCE RESERVATIONS

This subsection introduces the fundamentals of advance reservations in communication networks, where (1) the reservation models typically employed, (2) the time interval types, (3) the resource variability, (4) the architecture types and (5) the scope of the data structures are described. These aspects will be later used in Section 3.4 to compare the different solutions reviewed in this chapter.

3.1.1.1 RESERVATION MODELS

The reservation model describes the type of advance reservation that is supported by the solution. Usually, the advance reservation model supported depends on the timing constraints that are specified in the reservation request. In that regard, Zheng and Mouftah introduced the following classification [181].

- **Specified Time Specified Duration (STSD):** Advance reservations where the start time and the duration of the reservation or the end time are specified. STSD reservations can be of *fixed window*, where the possible start time is unique or of *flexible window*, where multiple start times are considered valid. This is the reservation model typically supported by most advance reservation systems and it is usually formulated as a *connection feasibility* problem or as a variant of the *soonest completion* problem where the start time needs to be within a certain time interval.
- **Specified Time Unspecified Duration (STUD):** Advance reservations where the start time is specified but the duration is unknown. Many authors formulate this problem as a *maximum duration* problem, where the resource consumption over time is analyzed in order to find the path able to sustain the connection between the specified end-points as long as possible. In addition, *immediate reservations* can also be considered a sub-type of this advance reservation category, where the start time is the time at which the request is performed.
- **Unspecified Time Specified Duration (UTSD):** Advance reservations where the start time is unknown but the duration or the

end time is specified. Usually, these types of advance reservations seek to transfer a huge amount of data before a certain deadline. Some authors formulate this problem as the *soonest completion* problem, where the goal is to find the path able to satisfy that the data transfer will finish as soon as possible, and therefore, will start as soon as possible.

- **Unspecified Time Unspecified Duration (UTUD):** Advance reservations where both the start time and the end time are not specified.

3.1.1.2 TIME INTERVAL TYPES

In advanced reservation systems, new service requests must be analyzed for different time intervals, each of them representing the resource consumption or availability over a specific period of time. As a consequence, it is a characteristic of advanced reservation systems that the running time of the algorithms will inevitably depend on the number of time intervals to be analyzed.

Advanced reservation systems can be based on *fixed* [182, 183] or *dynamic* time intervals [184–186]. On the one hand, fixed time intervals are used by systems that divide the time spectra into well-defined time slots of the same duration. Although this approach simplifies the analysis, when the time intervals are too small, the number of time slots to be analyzed grows, impacting negatively the running time of the employed algorithms. In addition, when the time slots are too big, the optimality of the system is degraded, since the reservations must be extended to fit within this wider time intervals and therefore consume resources when there was not needed. On the other hand, the solutions based on dynamic time intervals require more complex data structures, but do not impose such penalties on the optimality of the solution.

3.1.1.3 RESOURCE VARIABILITY

It is worth noting that even if the most common approach in advance reservation systems is to consider a sustained consumption of a given resource over time (i.e., bandwidth), some proposals consider the possibility of reserving different fractions of the resource in different

time intervals. As a consequence, advance reservations can be further categorized in the following sub-types:

- **Static reservations:** The reserved bandwidth does not vary over time.
- **Elastic or malleable reservations:** The reserved bandwidth varies over time, but no time slots where the transmission is suspended are considered [187].
- **Non-continuous reservations:** The reserved bandwidth varies over time, supporting time slots where the data transmission is suspended [188].

3.1.1.4 ARCHITECTURE TYPES

Depending on the type of entity that handles the reservation requests, advance reservation architectures can be differentiated between *centralized* and *distributed*:

- **Centralized:** A single entity handles all the reservation requests, the scheduling and the nodes configuration.
- **Distributed:** Each node handles the reservation requests that it receives and performs the admission control using its local information.

The type of architecture employed to support the advance reservations has a direct impact on how the path computation is performed. In this regard, centralized architectures make possible the utilization of path pre-computation strategies or the use of pruned topologies to ease on-demand path computations. On the other hand, distributed architectures impose the utilization of distributed routing protocols.

3.1.1.5 DATA STRUCTURE SCOPE

The scope of the data structure utilized to keep track of the resource consumption can have a huge impact on the optimality of the solution. In this regard, most advance reservation systems keep track of the utilization profiles on a per link basis. As a consequence, the data structure that is

used needs to be replicated per every link on the network. Besides the impact on the memory consumption, this results in additional constraints to perform the admission control of the reservation requests. With this approach, and given that each link supports a different set of reservations in each time slot, the resource consumption over time of each link evolves differently, which directly impacts how the path computation must be handled. This approach is mandatory, when a distributed architecture is used.

On the other hand, the data structure could be used to reflect the resource consumption over time on a per path or a per graph basis. In the latter case, this could result in a lower memory consumption, since a single instance of the data structure would be needed. In addition, the evolution of the network resource consumption could be handled atomically. Although this approach could simplify the admission control and the path computation in advance reservation systems, there are very few proposals that rely on these types of approaches. Even the solutions where a centralized architecture is used to handle the reservation are usually based on utilization profiles on a per link basis.

3.1.2 ALGORITHMS AND DATA STRUCTURES

As defined by Sedgewick and Wayne, algorithms are "methods for solving problems that are suited for computer implementation", which "go hand in hand with data structures - schemes for ordering data that leave them amenable to efficient processing by an algorithm" ([189], page 3). The impact of the data structures in the implementation of network algorithms has been widely studied by Robert Endre Tarjan [190].

A very important question when designing algorithms is to know their efficiency. To that matter, the most common approach is to estimate their *asymptotic complexity* using the *Big-O notation*, which provides an upper bound of the growth rate of an algorithm as a function of the input. It is represented by the symbol $\mathcal{O}(f(n))$, where n is the input. Throughout this document, the *running time* of an algorithm or the *computational complexity* of an algorithm are used as a synonyms of its asymptotic complexity.

In computer science, problems are categorized in tractable or intractable, depending on the complexities of the fastest known algorithms able to solve them. The separation between these two types depends on whether the algorithm runs in *polynomial* time or *non-polynomial* time respectively.

Since the goal of this chapter is to analyze the relationship between the data structures employed to keep track of the resources consumption over time and the performance achieved to process the reservation requests, this section briefly introduces some relevant data structure types to this topic:

- **Array:** a collection of elements of equal type identified by a unique index, where the indexes can be obtained through a mathematical formula [191]. During this chapter, only unidimensional arrays are considered.
- **Linked List:** an ordered collection of elements in which each entry, or node, points to the next one by means of a pointer.
- **Rooted Tree:** a hierarchical data structure where an element called *root* points to a set of *child* nodes. These child nodes can also be the *parent* of other nodes. The nodes that do not have any children are known as *leaf* nodes. On the one hand, the *depth* of a node is the amount of intermediate nodes to reach the root of the tree. On the other hand, the *height* of a node represent the number of nodes to reach a leaf node. There are multiple types of trees:
  - *Binary tree:* each node in the tree but the leaves have two child nodes.
  - *Balanced tree:* binary tree where the elements are sorted.
  - *Segment tree:* a tree specially designed to store intervals or segments.
- **Disjoint Set:** a data structure that keeps track of sets of elements divided in disjoint sub-sets, that is, where each sub-set only contains elements not included in the remaining ones. This data structure provides the means to identify the set that contains a certain element (find operation) and to form a new set as a union of two

disjoint sets (link operation). They are usually implemented using a tree.

3.2 RESOURCE MANAGEMENT ON A PER LINK BASIS

In a nutshell, advance reservation systems are the consequence of an exhaustive research effort in the field of network resource utilization. One of the most extended approaches to handle time-dependent resource management consists on keeping the information regarding the bandwidth utilization as a function of time on a per link basis. This approach is easy to implement, and can be supported both in distributed and centralized architectures. However, it requires the utilization of multiple instances of the data structure, one per each link in the network, which increases the memory consumption of the solutions.

An example of such approach is the one followed in [192], where advance reservations are proposed to enhance the network resource utilization in gigabit networks. In their paper, Varvarigos *et al.* present a protocol to enable the bandwidth reservation in advance using per link utilization profiles as a function of time, where the routing algorithm presented in [193] is used to select the optimal paths. Pretty much at the same time, Wischik *et al.* [175], demonstrated that advance reservation mechanisms were beneficial when the resources were booked a critical time in advance, with no positive effect when the time in advance was too small or too big.

Notwithstanding, even if these two proposals represent good examples of resource management on a per link basis, they do not delve with the impact of the data structures to process the reservation requests. As such, the following subsections present a comprehensive set of solutions where resource consumption on a per link basis can be handled to support advance reservations, and where the impact of the data structure is analyzed. As long as possible, the reservation model, the data structure and the algorithms on which the solution relies are presented.

3.2.1 SCHELÉN *ET AL.* PROPOSAL

According to Schelén *et al.* [194], admission control mechanisms are necessary to ensure that the service requests are provided with

the minimum requested quality. With such mechanisms, the network providers are able to reject service requests due to resource unavailability, and guarantee to the accepted requests that the services will be provided as desired.

In such a context, they have designed a QoS agent that behaves as a Bandwidth Broker, in order to enforce path-sensitive admission control able to scale in backbone networks. To state which data structure is the most suitable one to handle the time-constraints of advanced reservations, they compare a binary search tree over time and a segment tree that uses slotted time. Their analysis reveals that although a higher memory consumption is required, the segment tree behaves better for long service reservations than the binary search tree.

With both data structures handling the bandwidth consumption per link, they rely on the QoS agent to compute the path that satisfies the reservation requirements. To that matter, the QoS agent retrieves the topology and the available resources in the network using OSPF and SNMP, and computes the path taking into account that information and the resource availability of each link. Moreover, in order to state which routing approach is more appropriate, they compare on-demand path computation and path pre-computation strategies.

- *Reservation model*

The solution proposed by Schelén *et al.* supports both immediate and advance reservations, where the peak bandwidth is used as the QoS metric to select the paths. As stated in [195], the coexistence of these two reservation types is handled by aggregating the bandwidth of every immediate reservation and checking during a short period of time whether the demand can be satisfied taking into account the resource consumption of already accepted reservations. Regarding the advance reservations, they consider the STSD reservation model, where requests are identified by their start and end times, the source and destination nodes and the peak bandwidth.

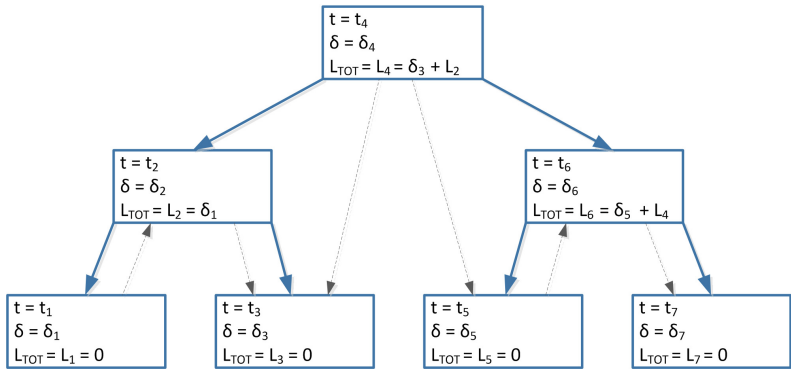


Figure 3.1.: Structure of the binary search tree over time proposed by Schelén *et al.* [194].

- *Data structures*

On the one hand, in the binary search tree over time, whose structure is depicted in Figure 3.1, every reservation is represented by means of two nodes. The first one represents the starting point of the reservation with a positive bandwidth, whereas the second one represents the ending point of the reservation with a negative bandwidth. If multiple reservations start or end in the same point in time, the node represents the aggregated bandwidth of all the reservations (identified as δ in the figure), where the bandwidth of the starting reservation is added and the bandwidth of the ending reservation is subtracted. In addition, the node also points to the next node in time and stores the aggregated bandwidth of the left sub-tree (identified as L_{TOT} in the figure).

When this data structure is used, the admission control process involves a binary search to find the starting point, the computation of the aggregated bandwidth of the left sub-trees and a linear search to study all the nodes that overlap with the new request to check. In the case when the node reflects a positive value of the bandwidth, it is checked whether the request can be accepted or not. In this case, since they conduct first the admission control and then they update the data structures, when the requests start to be rejected, the computation time decreases, since the process stops earlier.

On the other hand, the segment tree proposed by Schelén *et al.* to handle advance reservations is shown in Figure 3.2. In this segment tree,

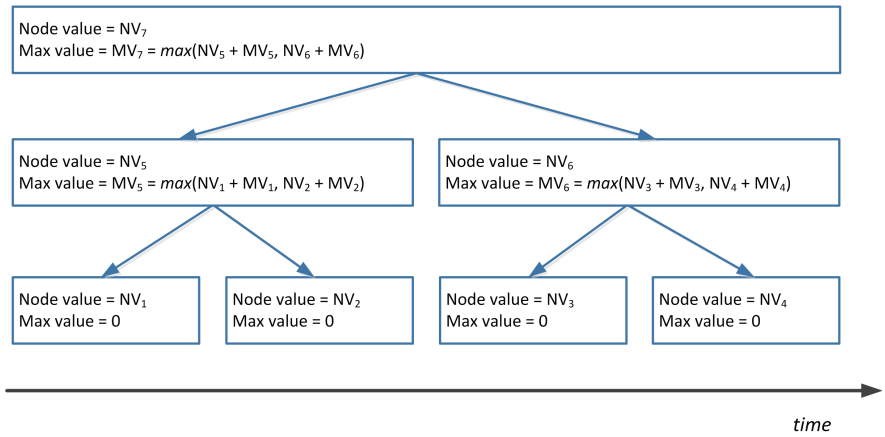


Figure 3.2.: Structure of the segment tree proposed by Schelén *et al.* [194].

the root node represents the entire time frame in which service requests can be accepted, where each frame is recursively divided into smaller and equally sized time frames until the leaf nodes represent a single time slot. As depicted in the figure, each node contains information about the available bandwidth during the time interval that it represents (identified as NV in the figure), and the maximum available bandwidth in its child nodes (identified as Max value in the figure)

When a new request arrives, the tree is traversed from the root to the leaves, checking in all the cases that applies if the bandwidth constraint remains satisfied. If positive, the request is accepted and the affected tree node is updated. All the operations are performed in constant time, independent of the number of reservations but dependent on the number of time slots and the length of the requested duration.

When the requests start to be rejected, this data structure results in a slower running time, since the data structure is updated at the same time that the admission control is performed. Besides, in addition to the unnecessary extra overhead imposed by the node updates, an extra overhead is imposed by the need of restoring the data structure to its previous state.

- *Algorithms*

In order to assess which strategy is the most appropriate one for advance reservations, the authors compare on-demand path computation and path pre-computation. On the one hand, using on-demand path computation, the admission control mechanism (either based on the binary search tree over time or the segment tree) must be invoked on a per hop basis. In a nutshell, they rely on OSPF, where the next hop is selected among those with enough resources available through the entire duration of the new reservation request. This approach may lead to path re-computations and the re-set of the already reserved resources in the path, since a selected path may lead to a neighbor node with no links available satisfying the demand.

On the other hand, they use the OSPF information available at the QoS agent to pre-compute shortest path trees between all the nodes in the network. Then, when a new reservation requests arrives, the tree is checked to find a path with enough resources in all of its links. Their experiments conclude that using a path pre-computation approach the admission control procedure takes a longer time, but it is still within the acceptable parameters in trunk networks.

- *Conclusions*

The comparison between the data structures and the path computation strategy conducted by Schelén *et al.* reveals a set of limitations common in the classic approaches to handle advance reservations.

First, regarding the data structures that are used, the analysis concludes that the segment tree is a better choice to handle advance reservations. However, their approach presents some drawbacks that could be improved. On the one hand, the use of slotted time results in a static structure where even the time periods in which no reservations exist must be represented using a leaf node, leading to an unnecessary memory consumption. In addition, they use a binary segment tree, which increases considerably the number of nodes that need to be kept in memory, even if those nodes do not represent relevant information. On the other hand, their approach to update the tree before checking all the affected time

slots may result in unnecessary tree traversals to revert to the original situation. This could be improved by executing first a checking phase and later a tree update phase.

Second, keeping track of per-link resources' consumption can lead to unnecessary path re-computations. As mentioned before, when the on-demand path computation approach is used, the admission control is enforced on a per hop basis. As a consequence, when a hop is found with no available links to satisfy the reservation demand, the already reserved resources in previous nodes must be re-set. It is worth noting, however, that this disadvantage could be solved following the path pre-computation approach.

3.2.2 GUERIN AND ARIEL PROPOSAL

As stated by Guerin and Ariel, advance reservations require enhancements in three major areas: (1) protocol and signaling capabilities to express advance reservation requests, (2) extensions to handle time constraints and (3) routing algorithms able to take into account temporal characteristics.

Their work is focused on the effect of the introduction of time constraints in the routing algorithms. To that matter, they present routing algorithms for both basic advance reservations and maximum total bandwidth, although the later one is out of the scope of this analysis. They consider that the time to make the reservations is fixed, and equally divided into time slots of the same duration. Their solution is based on per-link resource consumption management, which they represent using a vector that describes the bandwidth consumption at each time slot. Additionally, they also use delay constraints for the selection of the optimal paths. For the two reservation types that they analyze, they conclude that the path computation time is influenced by the number of time slots that are overlapped by the newly requested reservation.

- *Reservation model*

In their article, Guerin and Ariel present path computation algorithms for both basic advance reservations and maximum total bandwidth. In

$$L_i = [\{b_i[0], b_i[1], b_i[2], \ldots, b_i[n]\}, \delta_i]$$

Figure 3.3.: Data structure proposed by Guerin and Ariel [196].

the case of basic advance reservations, they study the running time of the path computation algorithms used to achieve the following objectives. First, to identify a path that satisfies the required constraints through the entire period of time for which the reservation is requested. Second, to find a path that maximizes the duration of a connection between two given points with a sustained bandwidth. Third, to find a path that minimizes the time at which a transfer of sustained rate and known duration starts. That is, Guerin and Ariel proposal supports the STSD and STUD reservation models.

- *Data structures*

In order to handle the time constraints, the authors of this proposal consider that advance reservation systems have a fixed number of time slots of equal duration. This allows to represent the bandwidth availability over time at a given link by means of a vector. As shown in Figure 3.3, each entry of the vector represents the available bandwidth at a given time slot. Additionally, in order to support additional constraints such as delay at the time of computing the paths, the data structure used to represent the resource consumption may include additional entries (in the figure, δ represents the delay in the given link).

Since the model considers a maximum time T in the future for which reservations in advance can be made, the vector is of known size, making easier to manage. Nevertheless, the utilization of a vector to represent the bandwidth consumption over time results in the running time of the algorithms being multiplied by a factor of m, being m the size of the vector.

- *Algorithms*

The procedures followed for the path computation are described below:

- **Connection feasibility:** The vector of each link is analyzed to check whether the requested period of time has enough resources to satisfy the reservation demand. If negative, the link is removed from the graph representing the topology. Once all the necessary links have been removed, the path computation is performed using SPF.
- **Maximum duration:** First, the maximum time for which the requested bandwidth can be sustained is identified in each link of the network, information that is later used to compute the widest path using BFS.
- **Soonest completion:** For each link in the topology, it is checked whether the starting time can sustain the requested traffic rate through the entire duration. It is not possible, the analysis continues with the next entry on the vector, until a valid starting time is found. If the analysis ends without finding a suitable starting time, the link is removed from the topology. As in the case of the connection feasibility, once the topology has been pruned, the path is computed using SPF.

- *Conclusions*

The authors of this paper characterize the effect of timing constraints on the path computation algorithms to support a variety of reservation types. In summary, the path computation is intrinsically affected by the fact that all the time slots that a new reservation request overlaps must be analyzed independently. This results in the running time of the algorithms being multiplied by at least a factor m (in the case of the soonest completion problem the vector can be scanned twice) , being m the number of affected time slots.

As a consequence, the granularity of the time slots plays a key role in the effectiveness of the solution. With bigger time slots, the amount of positions in the vector to be analyzed is lower than with smaller time slots. However, the utilization of bigger time slots may result in the reservations lasting more than the necessary time.

Their solution for the path computation in advance reservation systems could be improved by utilizing a more efficient data structure.

Furthermore, handling the resources on a per-link basis also imposes some additional challenges, since the vectors representing each link evolve differently.

3.2.3 BURCHARD PROPOSAL

As mentioned before, Burchard demonstrates the importance of the data structures and their effect on the admission control. He estimates that the effect of the data structures is around the 60% of the time required to analyze a request, whereas the routing takes the 8% and administrative tasks take a 32% [180]. Taking into account that failure recovery can be seen as a new reservation request and that re-optimization procedures to improve the network resource utilization require accessing those data structures, their efficiency is of uttermost importance.

In the analysis conducted by Burchard, the segment tree proposed by Guerin and Ariel and arrays are compared in terms of memory efficiency and the speed of the admission control process, concluding that arrays are easier to implement and more efficient, especially in terms of memory consumption. Nevertheless, the comparison conducted by Burchard assumes that the reservations are short compared to the book-ahead interval, which may not be extrapolated to different use cases where advance reservations are long, as it happens in the REN environment.

Burchard has deeply studied the performance of advanced reservation systems. Although this section is focused on the data structures that are used, it is worth mentioning that this author has also presented solutions to deal with network failures using re-routing [197, 198] and to improve the performance of resource utilization in the grid environment using malleable reservations, that is, reservations that can be satisfied in any given time interval [187] with varying bandwidth reservations over time.

- *Reservation model*

Burchard defines the book-ahead interval as the time interval for which requests can be issued. This book-ahead interval is divided into equal length time slots, each of which keeps track of the amount of allocated bandwidth in a specific time interval. The simplest reservation model

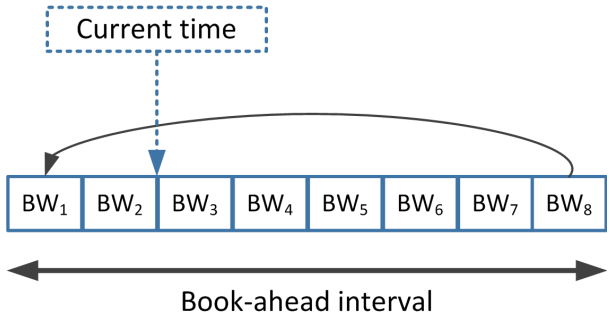


Figure 3.4.: Array proposed by Burchard *et al.* [180].

consider by Burchard corresponds to the connection feasibility with the assumption that the requested time is short compared to the book-ahead interval, that is, it considers a STSD reservation model. In addition, it also compares the two data structures for the soonest completion problem (STSD with flexible window) and malleable reservations.

- *Data structures*

The first data structure analyzed by Burchard is a circular array. Each entry in the array represents a time slot, and stores the accumulated bandwidth usage over time on a per link basis. As depicted in Figure 3.4, the array proposed by Burchard is of fixed length, where only the time slots of the book-ahead interval exist. In order to support the advance of the book-ahead interval, the array is implemented as a ring buffer, where the last index of the array points to the first entry. The current time is expressed by means of a pointer, which advances over the array as the time passes. This approach allows to limit the amount of memory consumption required to store the information related to the reservations, being the amount of memory $(b.s)$, where b is the length of the book-ahead interval and s is the memory required to store a single slot.

The segment tree used to make the comparison is the one proposed by Guerin and Ariel. However, Burchard has opted for an array-based implementation of the tree to reduce the memory consumption. In the original proposal, the segment tree is dynamically built using pointers. Instead, Burchard leverages the fact that the tree is binary to store it in an array, where shifting operations can be used for its efficient traversal.

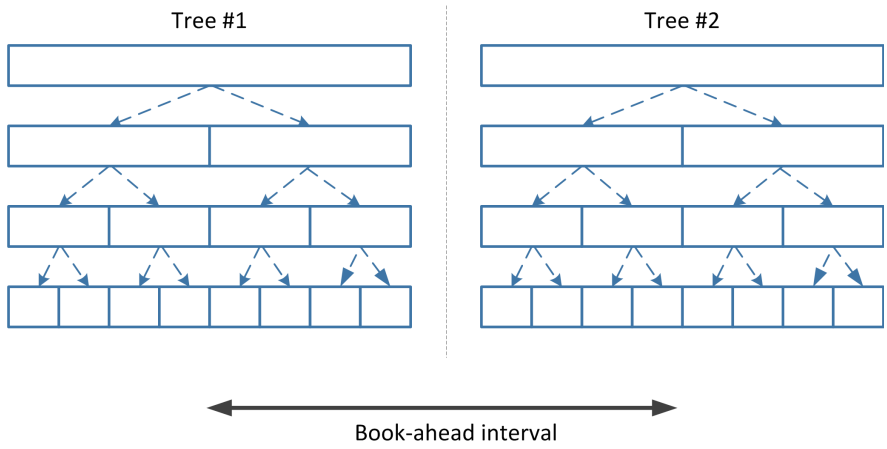


Figure 3.5.: Modifications to the segment tree introduced by Burchard *et al.* [180].

In addition, in order to cope with the fact that it uses a fixed book-ahead interval, it uses two segment trees, as depicted in Figure 3.5, to deal with the advance of the book-ahead interval as the current time passes.

- *Conclusions*

The analysis conducted by Burchard is focused on the memory consumption of the data structures. In that regard, it is obvious that an array will always behave better than a hierarchical tree of any kind, since trees require the creation of additional auxiliary nodes not necessary in the arrays.

Nonetheless, in terms of admission control efficiency, the comparison shows that arrays behave better only when the time requested for the reservation is short compared to the book-ahead interval. This assumption is valid for the use case proposed by Burchard, focused on the grid computing environment, but it is not true for all the use cases. In fact, in the case of BoD, service reservation requests are rather long.

Notwithstanding, the performance of segment trees is better than the performance of the arrays in a very specific advance reservation type, the connection feasibility problem. For other reservation types, this is not sustained. In fact, for the case of malleable reservations, the use of segment trees always imposes a penalization, since higher nodes are not

very useful, and finding a suitable interval (not necessarily the first one) requires accessing the leaf nodes.

3.2.4 WANG AND CHEN PROPOSAL

Aware of the limitations present in the advance reservation systems at that time, Wang and Chen proposed the utilization of more efficient data structures to ease path computation. Their data structure was presented in [199], where a hierarchical tree named Bandwidth Tree is used to represent the bandwidth availability on a per link basis.

With this data structure, they are able to leverage the utilization of very granular time slots while the computational complexity of the solution gets reduced. As stated before, when a linked list or an array is used to represent the bandwidth consumption over time, the running time of the path computation algorithm increases linearly with the number of affected time slots. With such data structures, smaller time slots result in a higher amount of time slots to be analyzed, hence imposing a penalization in the running time of the algorithms.

In order to solve this problem, Wang and Chen propose a hierarchical tree in which higher nodes, known as *internal nodes*, aggregate two or three consecutive child nodes. This way, a hierarchical tree in which the root node represents the maximum reservable time interval is formed. With this approach, the internal nodes represent wider time intervals, which correspond with the union of the time intervals of their children. As a consequence, it is possible to analyze multiple consecutive time slots together, where the available bandwidth announced at the internal node represents the minimal available bandwidth through the entire time interval.

- *Reservation model*

As in the work carried out by Schelén *et al.*, the data structure proposed by Wang and Chen aims to be able to handle multiple advance reservation types. On the one hand, they consider the problem of the connection feasibility between a source node and a destination node for a given time interval with a specific bandwidth, that is, an STSD reservation model.

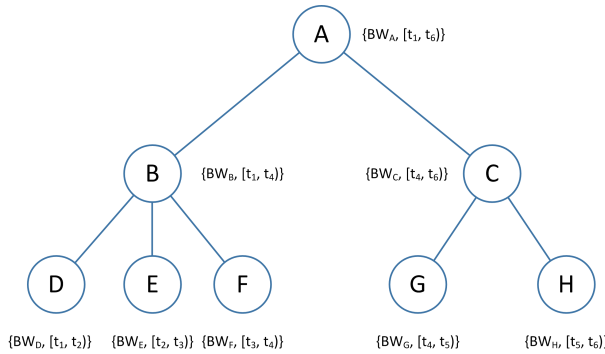


Figure 3.6.: Structure of the Bandwidth Tree [199].

On the other hand, they also consider the maximum duration problem, that is, to find the path that allows to connect a pair of endpoints with a guaranteed bandwidth for the longest possible time. Therefore, they also support an STUD reservation model. Finally, they also consider the soonest completion problem, which aims to find the earliest starting time for the reservation (STSD with flexible window).

- *Data structures*

The Bandwidth Tree, depicted in Figure 3.6, is a hierarchical data structure where the leaf nodes have all the same height and represent a non-empty time interval that it is not constraint by a specific value, therefore the solution supports variable length time slots. These leaf nodes are aggregated into internal nodes that represent a wider time interval where the minimum available bandwidth through the aggregated time interval is specified. In order to keep the Bandwidth Tree balanced, each internal node aggregates two or three consecutive nodes. Finally, the root node represents the maximum reservable time interval.

- *Algorithms*

On the one hand, Wang and Chen introduce a novelty with respect to the segment tree proposed by Schelén. Instead of checking the resources and updating the tree at the same time, they first conduct a *check* phase

and only if it is possible to make the reservation they conduct the *update* phase.

In order to check whether there is enough bandwidth to accept the reservation or not, they start checking the higher nodes. If the higher nodes can sustain the reservation, the request is accepted, if not, the lower height nodes are analyzed until the whole time interval of the reservation is checked. It is worth noting that in each node, instead of keeping the available bandwidth, they keep the minimal bandwidth. As such, the actual available bandwidth in a leaf node is computed by adding the minimal bandwidths of all the nodes from the root to the leaf node.

Since the Bandwidth Tree represents the per link bandwidth availability over time, the running times of the path computation algorithms are still dependent on the number of nodes and edges in the network. The procedures followed for the path computation are described below:

- **Connection feasibility:** In order to obtain a path from source to destination that satisfies the bandwidth constraints during the entire period of time, they check on every link in the topology if there is enough bandwidth. The links that cannot sustain the requested reservation are removed from the topology, and once they obtain the pruned version of the topology they run the Depth First Search (DFS) algorithm to obtain the shortest path.

- **Maximum duration:** First, they analyze the Bandwidth Tree of each link to obtain the maximum duration in each of them. Then, they use the maximum duration to compute the cost of each link and they construct the maximum spanning tree using the Kruskal algorithm [200]. The path between the source node and the destination node in the maximum spanning tree is the one that can sustain the reservation of the maximum duration.

- **Soonest completion:** In order to find the earliest starting time for the reservation, they check on the Bandwidth Tree of each link if the reservation can be satisfied using the first possible starting time. Once the topology is pruned with the links that can satisfy the demand, they use DFS to check whether a path to connect the source and destination nodes exists. If it does not exist, they

continue making the analysis with the next starting times, until a suitable one is found.

- *Conclusions*

The work carried out by Wang and Chen introduces very interesting concepts to handle advance reservations in an efficient manner. First, they check whether the reservation can be accepted or not and only if it is possible they perform the necessary updates to the data structures holding the resources information. Second, by aggregating multiple nodes into higher nodes they reduce the running times of the algorithms, both to check for resource availability and to compute the paths. With simple data structures as linked lists or arrays the running time depends linearly with the number of time intervals to be analyzed, whereas with a hierarchical approach the dependency is logarithmic.

Nonetheless, the solution also presents some limitations that could be improved. First, by using a root node to represent the maximum reservable time interval their solution results in a higher amount of nodes. Since the root or internal nodes aggregate consecutive nodes, this forces the creation of leaf nodes to represent time intervals where no reservation has been made. In addition, by limiting the number of child nodes to two or three the number of auxiliary nodes increases and therefore the memory consumption, while another way of aggregating the child nodes could result in a more efficient memory consumption.

Finally, as it happened with the other solutions included in this category to deal with advance reservations, each link requires its own Bandwidth Tree to represent the resource availability. This results in different Bandwidth Trees, since each link's resource utilization evolves differently depending on the services that are been carried out through them.

3.3 ALTERNATIVE RESOURCE MANAGEMENT APPROACHES

The present section analyses the advance reservation mechanisms where the resource management is not handled on a per link basis, such as on a per graph basis, or on a per network basis.

3.3.1 WOLF AND STEINMETZ PROPOSAL

Back in 1995, Wolf and Steinmetz [201] introduced a basic model for Resource Reservation in Advance (ReRA) systems, that is, systems with the capability of supporting resource reservations of known duration to be exploited in the future. Their purpose was to provide a generic model to support a wide range of services that may require advance reservations, such as video conferences or video-on-demand.

Among their contributions to the field of advance reservations, they defined a set of features to be provided by advance reservation systems. First, the system must provide the means to *translate and interprete* the QoS requirements of the applications in terms of affected resources. Once the translation has been done, the system must be able to *check* whether the required resources are available to satisfy the demand and if positive, *compute* the optimum QoS performance achievable. Then, the resources must be *reserved* and *scheduled* to be activated during the requested period of time.

Later in 1997, they further developed their work with a first implementation of the ReRA system [202], where they presented the data structures used to handle the advance reservations. It is worth noting, that this proposal has been included in this section since they do not specify if the resource management is performed on a per link basis or on a different approach. The reservation model, the architecture and the data structures that they use are described in the following subsections:

- *Reservation model*

The ReRA system presented by Wolf and Steinmetz supports both immediate and advance reservations. In order to avoid the systematic

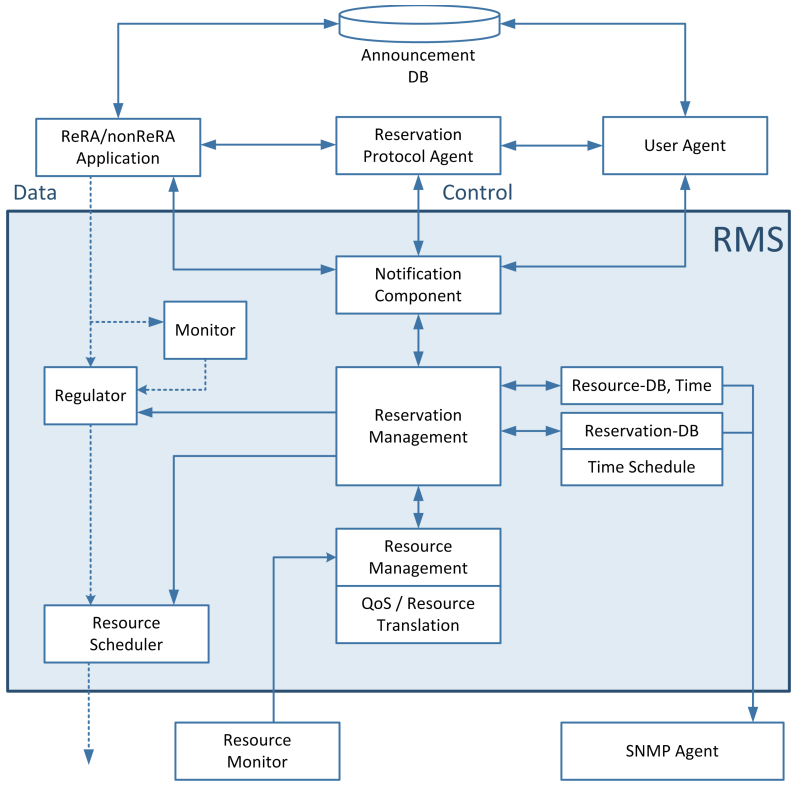


Figure 3.7.: Architecture of the ReRA system proposed by Wolf and Steinmetz [202].

rejection of immediate reservation requests due to the reservations made in advance, the pool of resources is divided in two different partitions. The first partition is devoted to immediate reservation, whereas the second partition is to be used by the advance reservations. In order to avoid the under-utilization of any of the partitions, the solution considers a dynamic threshold that gets updated depending on the actual reservations.

- *Architecture and work-flow*

The architecture for the ReRA system proposed by Wolf and Steinmetz is depicted in Figure 3.7.

In a nutshell, the ReRA system presented in this section supports reservation requests from multiple agents, such as *ReRA and non-ReRA* applications, *Reservation Protocol Agents* (i.e., Resource Reservation

Protocol (RSVP) agent) or *User Agents*. This is achieved by means of a component in the Resource Management System (RMS) named the *Notification Component*, which provides an interface to interact with the *Reservation Management* module. When a new reservation request arrives to the Reservation Management module, it coordinates with the *Resource Management* and *QoS/Resource Translation* modules to compute the QoS requirements, taking into account the actual network load available through the *Resource Monitor*. Once the QoS requirements are computed, the Reservation Management module uses the information contained in the *Resource-DB* and the *Reservation-DB* to schedule the service using the time-aware algorithm implemented in the *Time Schedule* module. Once the reservation request gets accepted in the system, the *Resource Scheduler* activates and deactivates the necessary resources to provide the service.

In their model, each reservation goes through five different stages: REQUEST, CONFIRM, DEMAND, ACCEPT and USE. The first 4 being part of the negotiation phase and the last one being part of the resource usage phase. In a nutshell, once a reservation is requested it goes to the REQUEST state, which transitions to CONFIRM only if there are enough resources to satisfy the demand. When the user is about to exploit the reserved resources, it DEMANDS the reservation, which triggers a procedure to check whether the exploitation of the reserved resources is still possible. If positive, the ReRA system marks the reservation as ACCEPT, and informs the user or the application that has requested the resources about it. It is worth noting that these last two states are optional, since the system can be configured to automatically exploit the reserved resources without interacting with the user or the requesting application. Finally, once the resources are actually being exploited, the reservation transitions to the USE state.

- *Data structures*

In the ReRA system, the time is divided in different QoS slices that represent a set of reservations. During the entire time slice, the QoS parameters and the state of each reservation remain unchanged. In order to represent the QoS slices, Wolf and Steinmetz propose the utilization of

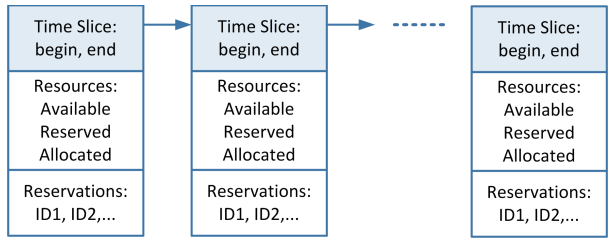


Figure 3.8.: Data structure for time management proposed by Wolf and Steinmetz [202].

a linked list, where the QoS slices are arranged chronologically, as depicted in Figure 3.8.

Each QoS slice is identified by its start and end times, and holds information about the available, reserved and allocated resources. This approach allows to handle variations in the available resources, as a consequence of network failures etc. and to distinguish the reservations that are actually being exploited. In addition, each QoS slice keeps track of the set of reservations that exist in that period of time, represented by their identifiers.

Furthermore, in order to handle each reservation, a second data structure is used, whose design is illustrated in Figure 3.9. For each reservation, the start and end times are stored, so as the set of QoS slices in which they are included. In addition, in order to handle periodic reservations, the data structure keeps separate track of a set of time sub-periods, where the resource load assigned to the reservation may vary.

- *Conclusions*

The approach followed by Wolf and Steinmetz for the ReRA system presents both advantages and disadvantages. On the one hand, the solution aims to handle all the resources needed to provide the E2E service, including the network resources. To that matter, the system automatically translates the service demand into a set of QoS constraints, which may vary along the different QoS slices to adapt to the varying availability of resources. In addition, the systems considers the possibility of having both ReRA and non-ReRA applications coexisting together, by assigning a different and disjoint set of network resources to be used by

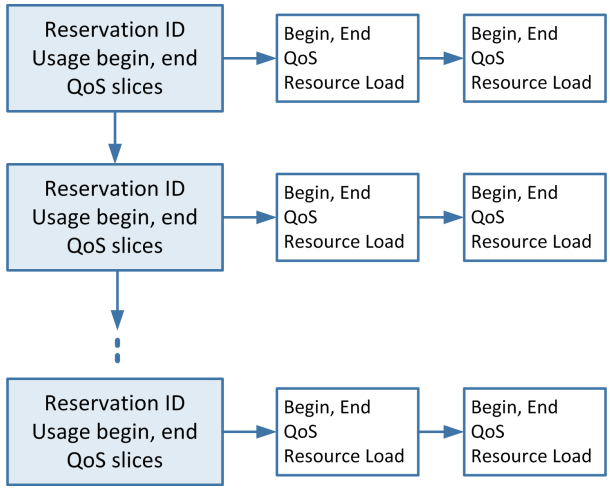


Figure 3.9.: Data structure for reservations management proposed by Wolf and Steinmetz [202].

each of them. Finally, they also state the need of external databases to replicate the reservation information and handle system failures.

On the other hand, even if the RMS has been designed as a centralized component, the reservation information needs to be distributed to the network elements part of the E2E circuit by means of distributed protocols such as RSVP, approach that presents some drawbacks. First, the reservation protocol used in the network needs to be modified to support time constraints. Second, each node is only aware of its own resource reservation, making difficult the detection of failures at other nodes along the path.

Moreover, regarding the resource management, the solution could be improved with a more efficient data structure. Although the use of a linked list imposes no constraints on the length of the reservations, it results in a computational complexity of $\mathcal{O}(n)$, being n the number of QoS slices. Finally, although the data structures are presented, the algorithm used to translate the service demands into QoS requirements and the algorithm used to check whether the reservation can be accepted or not are not described, making difficult a more comprehensive assessment of the solution.

3.3.2 ANDREICA *ET AL.* PROPOSAL

Andreica *et al.* [203] propose several data structures to handle advance reservations, namely arrays, disjoint sets, balanced trees, block partitions and extended segment trees. For each data structure, they compare the running time to perform the check and update operations, where the extended segment tree stands out with the lowest running time.

It is worth noting that in this proposal, resource utilization is handled both on a per link and on a per path basis. They argument the need to handle resources on a per path basis to increase the performance of advance reservation mechanisms. However, the approach they follow where the data structures are upgraded to handle multiple dimensions does not achieve this goal.

In such a scenario, this section reviews the different data structures and operations proposed by Andreica *et al.*. In addition, how they deal with the resources consumption on a per path basis is introduced to later finalize with a summary of the conclusions.

- *Reservation model*

This proposal introduces an interesting concept to the reservation model: requests are both characterized by constraint parameters and optimization parameters. On the one hand, constraint parameters specify a range of acceptable values for the reservation, such as the allowed start time, the duration, the minimum bandwidth, etc. On the other hand, optimization parameters are related to the objective function of the solution, that is, parameters that need to be minimized of maximized, such as congestion, delay, etc.

As mentioned before, they handle advance reservations on a per link and on a per path basis. In the advance reservation model where the data transfers are scheduled over a single link, a reservation r is defined by the earliest start time (ES_r), the duration of the data transfer (D_r), the latest finish time (LF_r) and the minimum required bandwidth (B_r), where the relation between the different parameters is defined in the expression 3.1. They also consider that once a data transfer starts, it cannot be interrupted. Therefore, it is an advance reservation of the STSD

with flexible window type. It is worth noting that they first consider a bandwidth model in which each request asks for the whole bandwidth on the link. Later, they introduce the possibility of requesting a fraction of the bandwidth link, in order to support multiple reservations at the same time intervals.

$$ES_r \leq t \leq t + D_r \leq LF_r \tag{3.1}$$

Regarding the advance reservations on a per path basis, they extend the advance reservation model to support the request of the bandwidth along a specific path. That is, in addition to the parameters described before, they also specify the list of nodes that the data transmission will use.

- *Data structures*

TIME SLOT ARRAY

The simplest way to handle the timing constraints of the reservation model that they propose when the resources are managed on a per link basis is by utilizing an *array*. The array divides the time spectra into equally sized time slots. Being m the size of the array, and therefore, the number of time slots, checking whether a reservation request can be satisfied or not takes $\mathcal{O}(m)$, which is also the time required to update the array once a reservation request has been accepted. This stands for both bandwidth models, when the request occupies the entire link capacity or when it occupies a fraction of it.

DISJOINT SET

For the first bandwidth model where the request occupies the full link capacity, the utilization of a disjoint set structure can speed the running time of the check phase. With this data structure a set identified by its start and end times represents a maximal interval where the link is occupied. Using this approach, they claim that the running time of the check procedure is reduced, because larger time intervals can be analyzed per operation. In the case of the update procedure, the running

time is $\mathcal{O}(m * log(m))$, which for high values of m actually imposes a penalization. They do not specify how they handle the second bandwidth model with this data structure. Notwithstanding, it represents a case of a data structure based on dynamic time slots. A disjoint set structure can be implemented by means of a linked list.

BALANCED TREE

The next improvement they introduced to the data structure is to organize the maximal intervals on a balanced tree, where a maximal interval is a set of consecutive slots with the same bandwidth availability. Although they do not provide an assured running time for the check phase, they consider that it will be faster than $\mathcal{O}(m * log(m))$. Moreover, by using a max-heap with the lengths of the intervals where all the bandwidth is available they can speed up the check procedure for the first bandwidth problem to $\mathcal{O}(1)$, since the longest available interval would always be stored at the head of the max-heap. On the other hand, the time complexity of the update phase is $\mathcal{O}(log(m))$, therefore improving the efficiency of the disjoint sets data structure.

BLOCK PARTITION ARRAY

In [204], Andreica presents a data structure based on block partitions. An array of length n, where each entry represents a time interval, is divided into k blocks of n/k elements. Each block keeps track of the first time interval and the last interval that it aggregates. This approach allows to speed up the checking and update phases to run in $\mathcal{O}(k + n/k)$, since multiple time slots can be checked and updated simultaneously. This data structure is valid for both bandwidth models.

EXTENDED SEGMENT TREE

The last data structure analysed by Andreica was presented in [205]. The segment tree is a balanced tree where the whole time interval where reservations are accepted is represented by the root of the tree. Then, the root is divided in two child nodes of equal length, and the same happens with these child nodes, until the leaf nodes that represent a single time slot are no longer divided. In order to ensure a running time of $\mathcal{O}(log(m))$, they keep track in each node of the aggregated bandwidth consumed by

the reservations that stopped during the represented time interval. In addition, they also keep track of the aggregated bandwidth of the cells aggregated by the node.

- *Support for resources consumption on a per path basis*

Andreica *et al.* is, as far of this PhD Thesis knows, the only author that has dealt with advance reservations on a per path basis. However, the reservation model that it considers includes the path as an explicit parameter of the reservation. Therefore, no impact on the path computation is analyzed.

All in all, Andreica considers multidimensional data structures to handle multiple constraints simultaneously. For instance, a two dimensional data structure could be able to handle network links and time slots simultaneously. Although it does not mention it, extending the time slot array to an additional dimension would be straightforward. However, this author has not been able to improve the performance of the advance reservation system with this approach, nor extend all the data structures. In his words *"We also extended the data structures to multiple dimensions, but the efficiency decreases with the increase in dimension. We were unable to make the extended segment tree support all the pairs of range queries and updates that can be supported in one dimension"* (in [203], page 292).

- *Conclusions*

The work carried out by Andreica is one of the most extensive ones dealing with the impact of data structures to support advance reservations. Although his achievements are related to advance reservations on a per link basis, it is worth mentioning that he considers handling advance reservations on a per path basis as the next step in this research area. According to the author of this PhD Thesis, we should extend this approach to handle advance reservations on a per network basis.

All in all, the analysis performed by Andreica shows that the data structure with the best performance is the extended segment tree, with a running time for both check and update phases of $\mathcal{O}(log(m))$.

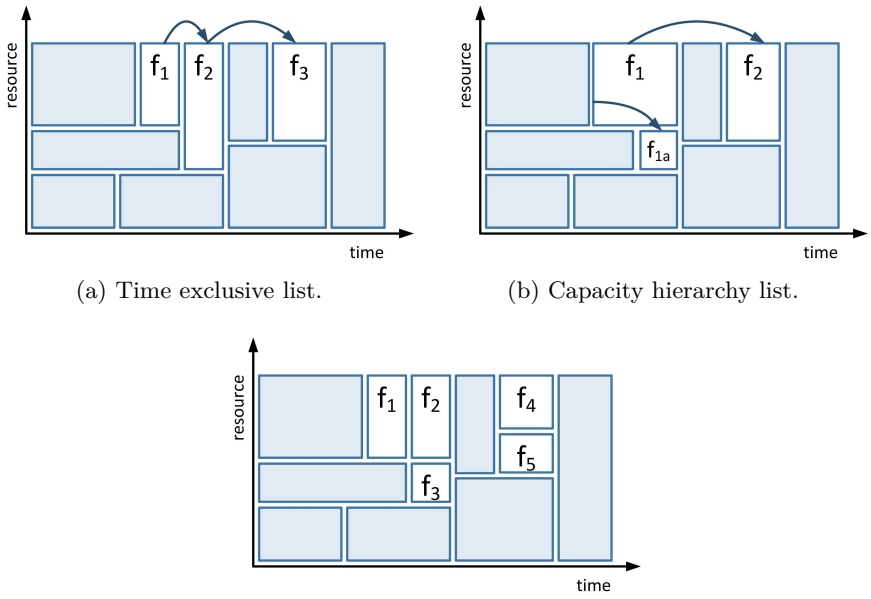
3.3.3 SCHNEIDER AND LINNERT PROPOSAL

In the grid computing environment, advance reservations are commonly used to schedule jobs among computing resources. When network resources are also reserved in advance, this requires making coordinated reservations, which is known as co-allocations. When co-allocations are supported, the advance reservation systems needs to be able to handle multiple resource types, requiring optimized data structures.

In such a scenario, Schneider and Linnert propose the utilization of lists of free blocks [206], as opposed to the common slotted time approach in order to avoid resources and time fragmentation [176]. In their solution, they create a dynamic list where each entry on the list represents a time range of free resources, and they analyze three possible methods to organize the items in the list. It is worth noting that they do not define any algorithm for the path computation and do not state the scope of the data structure, therefore, this section will only present the supported reservation model and the data structure that they have designed.

- *Reservation model*

The reservation model considered by Schneider and Linnert belongs to the STSD with flexible window. The reservation requests are characterized by a set of resource types, the requested capacity for each of them and the duration of the resources usage. In addition, the requester of the service may specify a *booking interval*, during which the reservation needs to take place. That is, their solution is meant to identify the start time that guarantees that the requested reservation will be provided for the specified duration during the booking interval. If no booking interval is specified, they consider the reservation of the immediate reservation type.



(a) Time exclusive list.

(b) Capacity hierarchy list.

(c) Mixed list.

Figure 3.10.: List of free blocks proposed by Schneider and Linnert [206].

- *Data structures*

Aware of the limitations imposed by the utilization of slotted time data structures to handle advance reservations, such as (1) the time and resource fragmentation, (2) the need to specify a book-ahead interval and (3) the fact that more slots than the necessary ones are checked in order to admit a reservation or not, Schneider and Linnert proposed the utilization of lists of free blocks.

A list of free blocks is a list where each entry represents a time interval with a set of available resources. As depicted in Figure 3.10, they consider three options to organize the items on the list.

- **Time exclusive list:** where there is a single entry per point in time, arranged chronologically to represent the available resources (see Figure 3.10a).
- **Capacity hierarchy list:** where each item represents the available resources through a sustained period of time and associated sub-

lists may be used to represent sub-periods of time where additional resources may be free (see Figure 3.10b).

- **Mixed list:** where the list items do not follow any rule and are ordered by their start time and the available capacity (see Figure 3.10c).

On the one hand, with the time exclusive list, the allocation process is enhanced compared to the slotted time approach since it is possible to analyze non-adjacent blocks of variable length. On the other hand, as stated by the authors, although checking for available resources may be simplified in a capacity hierarchy list, the allocation process may require splitting list entries in two or three list entries. In this case, having a capacity hierarchy tree complicates the process of updating the data structure. Finally, they state that the allocation process using mixed list is very complicated.

- *Conclusions*

Schneider and Linnert propose an interesting approach to handle advance reservations. The utilization of lists of free blocks is very similar to the disjoint set data structure proposed by Andreica *el al.* (which will be later explained in Section 3.3). Nonetheless, this solution stands out since by handling ranges of available resource, it is possible to have entries in the list not necessarily adjacent to each other in the time domain. In other words, they can have an entry to reflect the resource availability over June, and a next entry to reflect the resource availability over August, not being necessary to have an entry for July.

In addition, they introduce the possibility to create hierarchical lists of free blocks. Although as they pointed out, this data structure requires a higher processing time to make the necessary updates when a reservation is accepted into the system due to the splitting procedures that are required. Nonetheless, this approach could be useful, in scenarios with lower reservation demands.

3.3.4 BALMAN *ET AL.* PROPOSAL

In [207], Balman *et al.* propose a novel advance reservation algorithm that allows to suggest possible time intervals on which the reservation can be scheduled. The algorithm was designed to be included in OSCARS, the advance reservation framework of ESNet.

Among the novelties introduced by Balman *et al.*, the utilization of different network graphs that evolve through time to represent the resource availability is noteworthy. Instead of handling the resource consumption on a per link bases, it uses a linked list to represent different time intervals on which the network presents an stable resource availability. As such, this section reviews the reservation model that this solution supports, the novel data structure and the algorithms that it uses for the path computation. The algorithm is able to process reservation requests in $\mathcal{O}(n^2 * r^2)$, where n is the number of nodes in the network and r is the number of accepted reservations.

- *Reservation model*

The solution proposed by Balman *et al.* supports two reservation models. First, they are able to provide multiple alternatives to accept the reservation by solving the soonest completion problem. In this case, since they specify the latest possible end time, the reservation model belongs to the STSD with flexible window type. In this reservation model, the reservation requests are defined by the earliest start time, the latest end time, the maximum requested bandwidth, the size of the data to be transferred and the source and destination nodes. In addition, they also provide the means to advertise the starting time of the interval that ensures the shortest transfer duration, requiring the specification of the data volume that needs to be transferred.

- *Data structures*

In order to support advance reservations, they use a linked list where each entry represents a *time step*, where a time step is a time interval in which an static graph represents the available bandwidth in all the links of

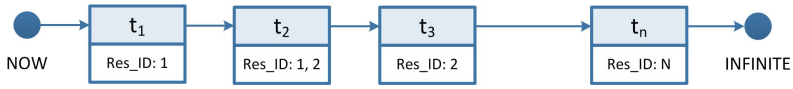


Figure 3.11.: Linked list proposed by Balman *et al.* [207].

the network (see Figure 3.11). That is, it represents a time interval where a set of reservations coexist. Instead of keeping the resource availability per time step, each entry on the linked list keeps track of the identifiers of the coexisting reservations. Every time a new reservation is accepted, it is necessary to update the linked list with new entries. And those already existing entries affected by the reservation need to be updated with the identifier of the new reservation. The data structure does not impose any constraints regarding the duration of the time steps, nor imposes a maximum book-ahead interval.

- *Algorithms*

In order to determine whether a new reservation can be accepted or not, they need to find a path able to sustain the desired bandwidth through the entire duration. This requires analyzing multiple *time steps*, which are computed on-the-fly, by checking on the linked list the time steps that overlap with the new reservation. In addition, it is necessary to ensure that the reservation can be accepted in all of the affected time steps.

As such, they create *time windows*, as time steps representing wider time intervals where the graph representing the minimum available bandwidth is stored. On the one hand, to find the time interval that guarantees the shortest transfer duration, they analyze first the time windows of smaller length, and if that window cannot sustain the reservation, they continue analyzing longer time windows. On the other hand, in order to solve the soonest completion problem, they start analyzing the time windows that present the earliest end time, also prioritizing the ones of smaller duration. It is worth noting, that since the linked list only keeps track of the reservation identifiers, the static graph needs to be computed on-the-fly.

In order to state whether a time window can sustain the reservation, they calculate the maximum available bandwidth from source to

destination using the static graph associated to the time window. Then, for the case of shortest duration transfer, they compute the volume of data that could be transferred during that time window taking into account the start and end times of the reservation, since it may not correspond with the start and end times of the time window. Once a window is found, the reservation is accepted, and the information regarding the reservation, including its start and end times, bandwidth and computed path are stored.

- *Conclusions*

The solution proposed by Balman *et al.* introduces a novel concept, very interesting for future solutions dealing with advance reservations: the utilization of static graphs to represent the resource availability during a specific period of time. This approach differs from the classical approaches where the resource availability was handled on a per link basis and presents numerous advantages. First, the time analysis gets simplified, since the graph represent a time interval where the computed path is valid. Second, the utilization of time windows covering multiple time steps ensures, with a single path computation, that the path is valid through a wider time interval.

Nonetheless, according to this PhD thesis, this solution could be improved by keeping track of the available resources on the linked list entries. With their approach, every time a new time window needs to be analyzed it is necessary to compute the static graph. This procedure could be avoided, by just keeping track of the bandwidth availability of the links, for instance, with an array. In addition, this graph-based approach could support the utilization of additional TE techniques such as flow relocation that could improve the network resources utilization and that are not implemented in this solution.

3.3.5 BARSHAN *ET AL.* PROPOSAL

Barshan *et al.* [208] present an architecture to support advance reservations in media production networks, which is depicted in Figure 3.12. It is worth noting, that they propose this advance reservation

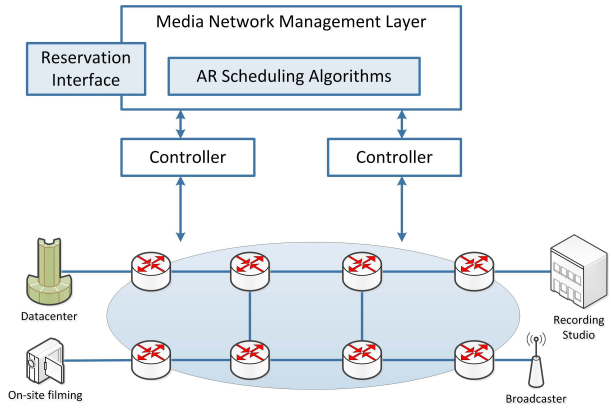


Figure 3.12.: Architecture proposed by Barshan *et al.* [208].

solution for its application in SDN. In media production networks, the traffic demands are usually known in advance, at least some hours before the data exchange begins, therefore, it represents a good use case scenario to apply advance reservations. It is worth noting that the work presented in this section has been used as basis to support resilient service provisioning in the same scenario [209], and to dynamically re-optimize the schedule scenarios using information retrieved from the network [210].

In their solution, the do not keep track of the resources consumption over time with any data structure. Instead, they store the information regarding the reservations and check in every time slot the already accepted reservations that need to be taken into account. This is a consequence of the reservation model that they use, where elastic and non-continuous advance reservations are supported and where the already accepted reservations can be routed through alternative paths to facilitate the acceptance of new incoming requests.

- *Reservation model*

In this proposals, multiple advance reservation models are considered, since they support two types of service requests that in addition, can be dependent to each other. That is, a set of sequential reservations may conform a scenario, and if and only if the network is able to accept all the reservations the scenario is accepted for scheduling.

First, they support video streaming, which requires a constant bit rate during a specific time interval. Second, they support file transfers, where a volume to be transferred before a certain deadline is specified. In addition, these two types of services can be executed sequentially as a result of a dependency relationship.

As such, video streaming can be STSD if the start time is specified, but it can also be UTSD if it needs to be scheduled after another service. Similarly, file transfers are considered STUD where they do not depend on a previous reservation, and UTUD when they depend on a previous reservation.

It is worth noting, that since no data structures are used to keep the resource consumption of previous reservations over time, it is necessary to specify the duration of each interval (I). As such, the nth reservation in a given scenario is specified in expression 3.2, where s^n and d^n are the source and destination nodes, t_s^n and t_e^n are the start and end times respectively, i^n refers to the duration of the reservation, b^n to the requested bandwidth and v^n to the volume of the files.

$$r^n = (s^n, d^n, t_s^n, t_e^n, i^n, b^n) \tag{3.2}$$

- *Algorithms*

The algorithms proposed by Barshan *et al.* are meant to identify when and how to schedule the reservation requests over the network. In that regard, they propose two algorithms: Static Advance Reservation Algorithm (SARA), applied when all the requests are known in advance, and Dynamic Advance Reservation Algorithm (DARA), applied when new scenarios enter into the advance reservation system.

In order to avoid the computational complexity of a LP-based solution, they use an heuristics-based approach. On the one hand, the SARA algorithm arranges the previously-known scenarios taking into account their earliest starting time and the amount of resources pending to be consumed. Then, it stores the current state of the reservation and the resources that it is consuming and checks whether a re-schedule is feasible. If positive, it re-schedules the reservation and if not, the scenario is kept in its current schedule. On the other hand, the DARA algorithm is

executed every time a new scenario is requested, and takes into account the scenarios currently being provisioned through the network in order to avoid disrupting the already accepted services.

To check whether a reservation can be scheduled or not, the algorithm checks the reservations that are scheduled for the entire time interval that the reservation has been made. This procedure needs to be repeated every I seconds. In the case of the video streaming services, it is necessary to have a sustained rate in all the intervals, whereas with the file transfers it is possible to assign a different bandwidth.

- *Conclusions*

The solution proposed by Barshan *et al.* stands out due to the variety of advance reservation models supported. However, their decision of not keeping track of the consumed resources over time degrades the performance of the solution.

The experiments that they have conducted reveals that the time slot granularity that they specify for each scenario request affects the processing time. As such, the smaller the time slot is, the higher the processing time is, resulting in 10 s being necessary to process the requests when 1 min time slots are used. On the contrary, they also demonstrate that the utilization of smaller time slots reduce the blocking probability of the system. Hence, there is a trade-off between the time required to process the requests and the requests acceptance ratio.

3.4 COMPARISON OF ADVANCE RESERVATION SOLUTIONS

In order to assess the suitability of the different data structures to handle advance reservation, Table 3.1 summarizes the proposals analyzed in this chapter. Please note that the variables used to express the running time are the ones that follow: "k" refers to the number of links in the network, "m" refers to the number of entries in arrays and lists or nodes in hierarchical structures, "b" refers to the size of the blocks in the array of block partitions proposed by Andreica, whereas "r" refers to the number of already accepted reservations.

Table 3.1.: Comparison of advance reservation solutions

Proposal	Data Structure	Reservation Model	Architecture	Resource variability	Scope of DS	Slot type	Running time
Schelén *et al.*	Binary search tree	STSD	Distributed	Static	Link	Dynamic	$k * \mathcal{O}(log(n) + m)$
Schelén *et al.*	Segment tree	STSD	Distributed	Static	Link	Fixed	$k * \mathcal{O}(log(n))$
Guerin and Ariel	Array	STSD & STUD	Centralized	Static	Link	Fixed	$k * \mathcal{O}(m)$
Burchard	Circular array	STSD	Centralized	Static	Link	Fixed	$k * \mathcal{O}(m)$
Wang and Chen	Interval Tree	STSD & STUD	Centralized	Static	Link	Dynamic	$k * \mathcal{O}(log(m))$
Wolf and Steinmetz	Linked List	STUD	Centralized	Static	Unknown	Dynamic	$\mathcal{O}(m)$
Andreica *et al.*	Array	STSD	Centralized	Static	Link & Path	Fixed	$\mathcal{O}(m)$
Andreica *et al.*	Disjoint Set	STSD	Centralized	Static	Link & Path	Dynamic	$\mathcal{O}(m * log(m))$
Andreica *et al.*	Balanced Tree	STSD	Centralized	Static	Link & Path	Dynamic	$\mathcal{O}(log(m))$
Andreica	Block partitions	STSD	Centralized	Static	Link & Path	Fixed	$\mathcal{O}(b + m/b)$
Andreica	Segment tree	STSD	Centralized	Static	Link & Path	Dynamic	$\mathcal{O}(log(m))$
Schneider and Linnert	List of free blocks	STSD	Unknown	Static	Unknown	Dynamic	$k * \mathcal{O}(log(m))$
Balman *et al.*	Linked List	STSD	Centralized	Static	Graph	Dynamic	$\mathcal{O}(k * r^2)$
Barshan *et al.*	None	ALL	Centralized	NC	None	Fixed	Unknown

In a nutshell, there is a clear relation between the employed data structure and the running time of the solution. On the one hand, using arrays results in a linear running time, since no aggregation of time intervals is possible. The only solution that is based on arrays and presents a better performance is the *block partition array* introduced by Andreica, where the running time gets reduced by enabling checking blocks of multiple array cells simultaneously.

The linked lists present the same inconvenience, they impose a linear running time. However, they make possible the utilization of dynamic time slots, which can result in a lower memory consumption since the nodes can be created dynamically based on the service demands.

Finally, the hierarchical trees are the ones that behave more efficiently at the time of handling the resource utilization evolution over time. The utilization of hierarchical trees, either if it is a segment tree, a binary tree or a balanced tree guarantee a logarithmic running time. In addition, these kinds of structures can be based for both fixed and dynamic time slots and in distributed or centralized architectures.

3.5 CONCLUSIONS

As demonstrated with the analysis conducted in this chapter, data structures have a huge impact on advance reservation systems.

On the one hand, the resource management approach plays a key role. Most classical approaches, the first ones dealing with advance reservations, considered a link based resource management. Notwithstanding, the bandwidth on a given link is the resource being shared by the reservations. However, keeping track of the resource utilization profile per each link on the network requires the utilization of multiple instances of the data structure. In the end, this imposes a penalization in terms of memory, and does not avoid the fact that the process needs to be repeated k times, being k the number of links in the network in order to compute the path that satisfies the reservation constraints. Although this approach is required by distributed architectures, centralized architectures could benefit of a more efficient approach to deal with advance reservations. This is outlined by Balman *et al.*, where a graph-based resource management approach is considered.

On the other hand, the data structure itself has an impact on the efficiency of the solution. It is indubitable that handling advance reservations requires taking into account the evolution over time of the resource consumption. When arrays or linked lists with fixed time slots are used, the running time of the algorithms are always linearly dependent on m, being m the number of time slots considered in the book-ahead interval. In addition, time intervals where no reservations at all exist need to be handled as well, increasing the memory consumption. Furthermore, the size of the fixed time slots can result in unnecessary resource and time fragmentation, and impact the optimality of the solution.

In that regard, the utilization of hierarchical data structures presents some benefits. First, by allowing the analysis of wider time slots the running time of the algorithms gets reduced to a logarithmic function. Second, the hierarchical data structures are able to support non-consecutive time slots, therein reducing the need to keep track of the resources during the entire book-ahead interval. This last fact results in an additional benefit: there is no need of a book-ahead interval.

All in all, there are currently no solutions based on hierarchical structures where the resource management is performed on a per graph basis. The utilization of SDN, with its logically centralized control plane that facilitates the automatic abstraction of the network graph represents the perfect environment to re-think how advance reservations are handled, and leverage the lessons learned throughout these last decades on the impact of data structures on such systems.

Part III

PROPOSAL

4 A generic framework for traffic engineering in SDN

> "As an architect, you design for the present, with an awareness of the past, for a future which is essentially unknown."
>
> — Norman Foster

The first contribution presented in this thesis is the architecture of the DynPaC framework, which is depicted in Figure 4.1. As mentioned before, DynPaC is a generic framework aiming to facilitate the adoption of TE strategies in software-defined networks. It consists of 6 well-defined modules and a REST API that facilitates its integration with external elements, which are described in the following sections.

4.1 MODULES

This section describes the modules that conform the DynPaC framework, namely PCE, Topology Abstractor, Resilience Module, Monitoring Module, Network Programmer and Service Manager.

4.1.1 PATH COMPUTATION ELEMENT (PCE)

As already stated in Chapter I, in communication networks, TE consists in the application of strategies and scientific principles to optimize the

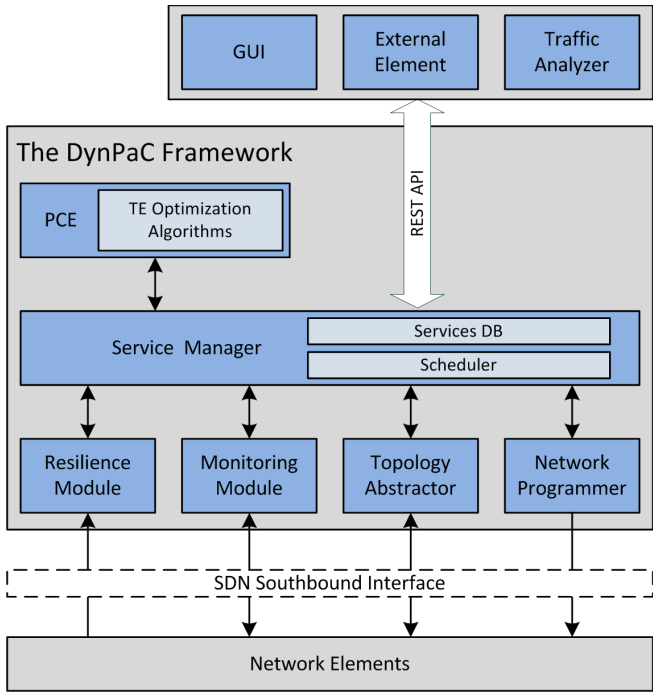


Figure 4.1.: Architecture of a generic framework for advance path computation in SDN.

performance of operational networks, as described in RFC 2702[13]. Having such an objective, path computation plays a key role in TE, since it directly impacts the network resource utilization. As a consequence, given that the DynPaC framework aims to bring TE to software-defined networks, the framework includes an element whose solely purpose is to perform path computation. The approach is very similar to the one proposed in the PCE-based architecture, defined in RFC4655 [88], where a dedicated element is in charge of the path computation in MPLS [65] / GMPLS [211] networks.

One of the main benefits of having a dedicated element in charge of the path computation is that the algorithm can be easily replaced or updated. For example, it is possible to use different routing algorithms such as Dijkstra [132] or Ford-Fulkerson [212] depending on the use case, just by including such algorithms in the PCE. Furthermore, it eases the utilization of novel and customized path computation algorithms, with different objective functions such as the minimization of the power

consumption if the framework is used for green computing, or the maximization of the QoE, if the framework is used to provide video on-demand services. The specific algorithm used for the BoD application is later explained in Chapter V.

4.1.2 TOPOLOGY ABSTRACTOR

The PCE requires a model of the network in order to perform the path computation. Moreover, the path computation procedure gets greatly affected by the data structures and the models that are used to represent the network. For instance, the performance of the well-known algorithm Dijkstra is dependent on the data structure. The first implementation of Dijkstra's algorithm run on $\mathcal{O}(n^2)$, where n is the number of nodes in the network. Notwithstanding, more recent implementations based on the utilization of minimum priority queues or heaps have reduced the running-time of the algorithm to $\mathcal{O}(m + n * \log(n))$, where m is the number of links in the network [213]. The classic approach is to represent the network as a graph, which requires the identification of the network elements. That is, the vertexes of the network graph and the links that unite them, also known as the edges of the network graph.

Usually, SDN controllers use the LLDP and Broadcast Domain Discovery Protocol (BDDP) [214] for this purpose. However, once the topology is discovered, the optimal way to represent it may vary depending on the network type or the use case. As such, some topologies may require the utilization of adjacency lists, whereas other may require the utilization of adjacency matrices.

In such a scenario, the main objective of the topology abstractor is to construct the correct network model that will facilitate the path computation. For example, for the implementation of this framework the network graph is retrieved automatically and abstracted as a unidimensional array, where each entry represents the available bandwidth of a bidirectional link that interconnects two given nodes, leveraging the fact that the services are considered symmetrical and bidirectional.

4.1.3 RESILIENCE MODULE

Since the minimization of the packet loss is a common TE objective, this framework includes a resilience module that aims to minimize the service disruption and the packet loss in the event of a network failure. The resilience module monitors the state of the network infrastructure and when it receives an event informing about a node or a link failure, it communicates with the service manager (later explained in this section) so that it executes the recovery mechanisms. It is worth noting that the framework does not impose any specific mechanism for the detection of the network failures. For instance, the network failure could be detected either by means of an active protocol such as the BFD protocol defined in RFC 5880 [215], or passively by listening to notifications from the switches. The framework neither imposes any particular recovery mechanism; some use cases may require path protection, whereas others may require path restoration [216], as detailed below:

- *Path protection:* Failure recovery mechanism where the backup resources are pre-computed and reserved. Only after the network failure occurs the reserved resources are used in the actual provisioning of the service.
- *Path restoration:* Failure recovery mechanism where the backup resources are not reserved. In the event of a network failure, path restoration may require the computation of the backup path.

In both cases, when the backup path is pre-computed or when it is computed on-demand, it is necessary to check whether there are enough network resources or not. Only in case there are enough resources, the backup path is installed in the network.

Either case, once the resilience module detects the failure, the DynPaC framework re-programs the affected network elements in order to install alternative routes to keep providing the affected connectivity services. This feature is of especial relevance for multiple use cases, i.e., for applications where the high availability is necessary, such as emergency services and urban traffic signaling, or real-time communications.

4.1.4 MONITORING MODULE

The DynPaC framework includes a monitoring module that keeps track of the resources consumed by each connectivity service provisioned in the network. Monitoring is essential from both the end-users and the network operators' perspective, since it allows to verify that the SLAs subscribed between the two of them are being fulfilled.

It needs to be taken into account that the optimization of the network resource utilization may encompass the adoption of TE techniques. A classical approach consists of splitting the incoming traffic into multiple paths to minimize the congestion. When such technique is utilized, it is necessary to verify that the service is being provisioned as promised, since the enforcement of SLAs through mechanisms like rate limiting usually take place at the edge network elements, but not in the intermediate elements.

In such a scenario, the monitoring module has been designed to keep track of per-flow and per-service network resource consumption. This way, the monitoring module is able to deal with the disaggregation of an original service demand into multiple sub-services, which as will be later explained, is one of the optimization techniques used in the DynPaC framework.

The module does not impose any particular mechanism to obtain the QoS metrics. On the one hand, in a SDN network with OpenFlow devices it could be programmed to obtain different metrics from the statistical information kept at the switches per each flow, such as the bandwidth or the delay [217]. On the other hand, it could retrieve the information from a NetFlow or sFlow collector. In summary, the monitoring module should be able to retrieve information about the metrics relevant for the computation of the paths, and react upon the violation of any SLA, for example, by notifying the service manager about the situation.

4.1.5 NETWORK PROGRAMMER

The network programmer is the module in charge of programming the network elements involved in the provisioning of a given service. As mentioned before, in advance reservation systems, services are reserved for

the future and for a certain amount of time. This implies that the network resources that will be consumed by the service are free to be used by other reservations until the service start time reaches. Once that happens, the optimum path that the PCE has computed for the service needs to be installed in the network. Similarly, once the service end time reaches, the network programmer is in charge of removing all the associated state from the involved network resources. Overall, the network programmer is in charge of translating the information contained in the form of a path, that is, a sequence of network elements, in the appropriate messages in order to program the network devices. For instance, in the case of OpenFlow devices, this would imply the generation of the necessary *flow modification* messages in order to install or remove the flow entries from the OpenFlow switches.

4.1.6 SERVICE MANAGER

The *Service Manager* is the core of the DynPaC framework, since it is in charge of the work-flow management, the services' life-cycle and the resource management, which are individually described next.

4.1.6.1 WORK-FLOW MANAGEMENT

All the modules of the framework and the REST API communicate with the Service Manager, which orchestrates the interaction of the different modules in order to provide the requested connectivity service.

Firstly, the Service Manager triggers a mechanism in the *Topology Abstractor* to retrieve the topology of the network, which performs a set of abstractions that eases future path computations performed at the PCE. Once the topology is abstracted to the format required by the PCE, the Service Manager is ready to receive new service requests.

Every time a new service is requested, the Service Manager requests a path or a set of paths to the PCE. Then, the Service Manager checks whether the service request can be accepted or not, taking into account the resources available in the network, the paths provided by the PCE and enforcing, if necessary, the set of TE optimization techniques also available in this module (see Chapter VI). Furthermore, once a service is accepted, it listens to the events produced by the *Resilience Module*

and the *Monitoring Module* to react upon network failures and ensure the fulfillment of the SLAs respectively. In addition, the Service Manager communicates with the *Network Programmer* every time a service needs to be installed in or removed from the network.

4.1.6.2 SERVICES' LIFE-CYCLE MANAGEMENT

One of the main tasks performed by the Service Manager is the management of the services' life-cycle. It is worth noting that the DynPaC framework has been designed to support advance service reservations. That is, the framework supports the reservation of services that are automatically installed in or removed from the network in a future time. As a consequence, it is necessary to keep track of the state of each service, in order to know if the service is reserved for the future, installed in the network, or has expired. For this purpose, the Service Manager uses a data-store named *Services DB*. The services' life-cycle differs depending on the operational mode at which the framework operates, namely *synchronous* and *asynchronous* modes, which are later explained in Section 4.2.2.1.

4.1.6.3 RESOURCE MANAGEMENT

As mentioned before, one of the main features of the DynPaC framework is its support for advance reservations. For such a feature, it is necessary to keep track of the resources consumption over time, which is done by means of a data-store named *Scheduler*. This approach is consistent with the *Stateful PCE*, described in RFC 4655 [88], which uses not only the information contained in the TED for the path computation but also the resources consumed by already accepted connectivity services in the network. Since the efficient management of the network resources to support advance reservations is one of the main objectives of this thesis, it will be explained later in Section 5.3.

4.2 INTERFACE TOWARDS EXTERNAL ELEMENTS

One of the most important features of the DynPaC framework is that it provides an open REST API for its integration with external elements. On the one hand, the interface can be used by external traffic analyzers to send information about the traffic that could be useful to optimize the

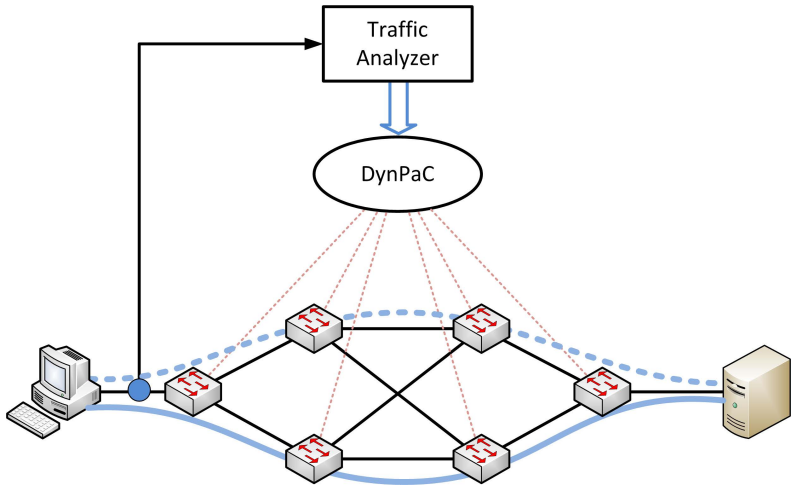


Figure 4.2.: Integration of the DynPaC framework with a traffic analyzer.

network resource consumption, as it will be explained in Chapter VI. On the other hand, the interface can be used by external elements such as orchestrators to handle the life-cycle of the services remotely. Both cases are explained in detail in the following subsections.

4.2.1 TRAFFIC ANALYZERS

Historically, TE solutions have benefited from previously known traffic patterns and service demands. However, these kinds of solutions based on traffic matrices are only valid for the specific network for which the traffic is characterized, and they cannot be generalized to other networks.

Having this limitation of classic TE solutions in mind, the REST API exposed by the DynPaC framework has been designed to facilitate the interconnection with external traffic analyzers, or in general, to any external element capable of providing on-demand information about traffic patterns, as illustrated in Figure 4.2. This information can be later used to optimize the network resource utilization. For instance, a traffic analyzer can use this interface to provide information about the periodicity of a certain traffic. This use case is of special relevance for the Internet of Things (IoT) in which the IoT devices usually transmit their data in specific time slots to achieve energy efficiency. Knowing in

advance the time evolution of a traffic pattern can be useful to rearrange the network load in a safe manner.

Furthermore, an external element can provide information about the bandwidth consumption for different traffic types through this interface, allowing to characterize different traffic sub-flows inside an ongoing service. This could make possible the disaggregation of original services into multiple sub-services that could be more easily accommodated in the network, making easier the acceptance of new service demands that with the previous network load configuration would be rejected.

4.2.2 ORCHESTRATORS

With the advent of NFV and the need for multi-domain communications, the REST API of the DynPaC framework has been designed to ease the integration of the framework with external orchestrators. The REST API provides the means to retrieve all the relevant information about the DynPaC framework and the accepted services. Moreover, all the operations supported by the DynPaC framework, such as the reservation of a new service or its termination, are also exposed through the REST API. As a consequence, external elements such as NFV or multi-domain orchestrators can use the REST API to communicate with the DynPaC framework and request the connectivity services that they require. Similarly, the REST API can be used by an external GUI to ease the interaction with the framework. This way, the users or the network administrators can have access to the information about the services in a more user-friendly manner, or make the service reservations by simply filling a form.

4.2.2.1 MULTI-DOMAIN CAPABILITIES

Of special relevance in this section is the possibility that the REST API provides to support multi-domain connectivity services. More precisely, in order to provide multi-domain capabilities, the DynPaC framework has been designed to be compliant with the Network Services Interface - Connectivity Service (NSI-CS) [218] protocol, which is part of the NSF standardized by the Open Grid Forum. This has been achieved extending the REST API exposed by DynPaC and introducing a second

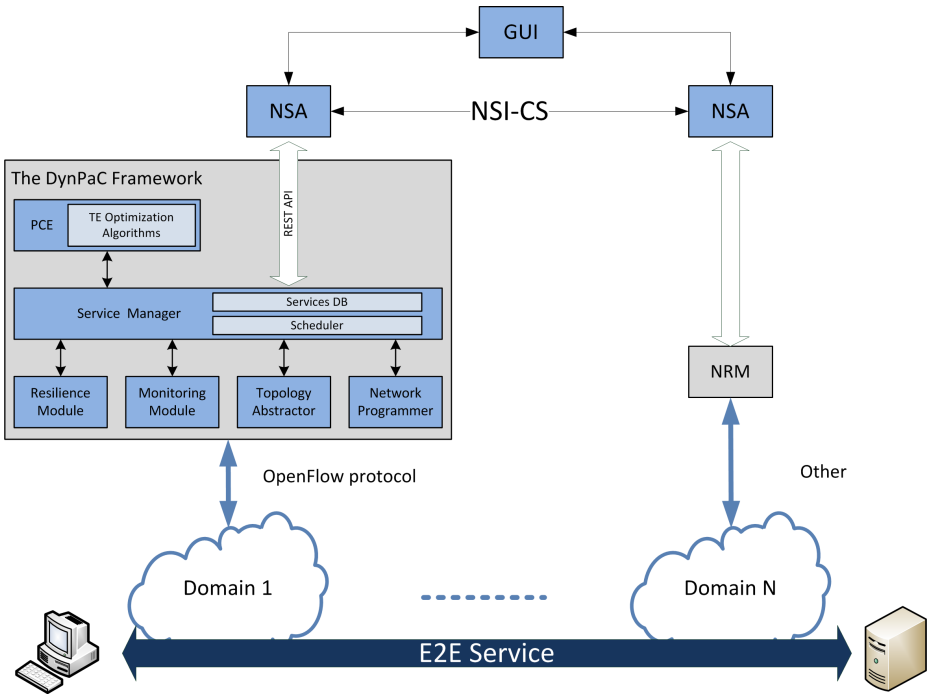
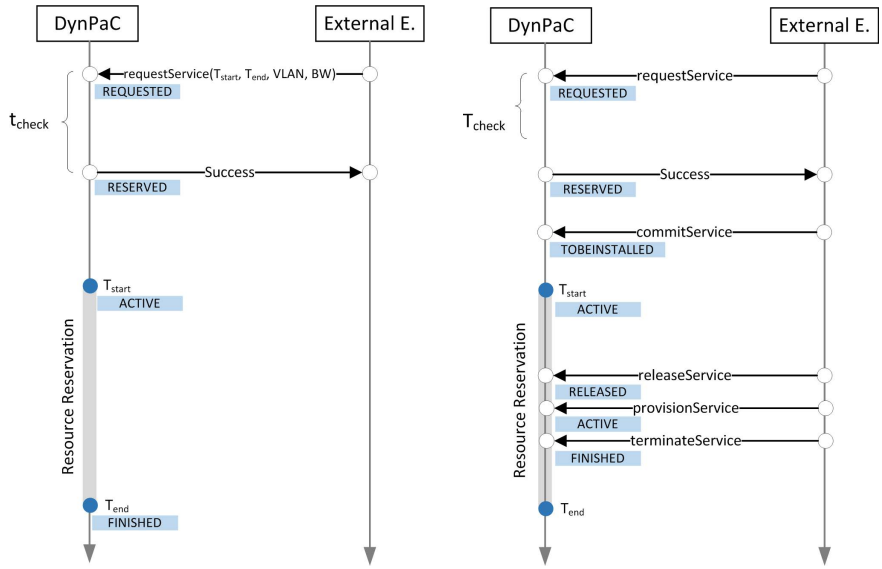


Figure 4.3.: Integration of the DynPaC framework as NRM for multi-domain BoD service provisioning.

operational mode that allows DynPaC to act as the Network Resource Manager (NRM) of OpenFlow domains, as depicted in Figure 4.3.

In NSI-CS, two main elements are involved in the E2E service provisioning. On the one hand, the Network Services Agent (NSA) are in charge of handling the communication with the other NSA peers and of negotiating the E2E services' setup, activation/deactivation, modification and tear down. They also exchange topology information and thus handle an abstracted form of the domain topology that is suitable for advertisement. In other words, the NSAs are in charge of the inter-domain aspects of the service provisioning. On the other hand, the NRM is the NSI-CS component ultimately responsible for managing the transport resources. It thus needs to be able to model the detailed domain topology, maintain a resource utilization calendar and calculate intra-domain paths based on calendar and topology constraints.

In order to be able to use the DynPaC application as an NRM, the NSA needs to be able to map the internal NSI-CS states within the



(a) Synchronous. (b) Asynchronous.

Figure 4.4.: DynPaC operational modes.

DynPaC work-flow. NSI-CS allows for advance reservations, which is a concept already supported by DynPaC, but it also supports the concept of multiple provisioning/releasing cycles within a reservation, which is the process of building or tearing down the actual circuit without canceling the booking of resources. This feature is very useful for the cases where there is a planned network downtime and the user needs to be aware of the exact times when the circuit is unavailable, without setting off monitoring alarms.

As a consequence, to be compliant with the NSI-CS work-flow, the framework has been extended with an additional operational mode, called *ashynchronous operational mode*, in which the network devices are updated upon external request. Figure 4.4 depicts the life-cycle of the services for the two operational modes currently supported in the DynPaC framework: synchronous and asynchronous.

In a nutshell, in the asynchronous operational mode the external element acts as the master, whereas the DynPaC framework acts as the slave regarding the services' life-cycle management. That is, all the changes in the services' life-cycle is directly handled by the external element. On the other hand, in the synchronous operational mode, the

service manager is still in charge of the services' life-cycle management. Figure 4.4 depicts the two operational modes available in the DynPaC framework, and how the service state varies in both cases upon reception of the REST API calls.

First, in the synchronous operational mode, the external element makes a new service reservation request specifying the service start and end times, the bandwidth and the VLAN tags that will be used to isolate the circuit. At that moment, the service is in *REQUESTED* state. Once DynPaC checks that there are enough resources in the network to accept the service, it acknowledges that the service has been accepted to the external element and switches the state of the service to *RESERVED*. Later, when the service start time reaches, the service is automatically installed in the network, and marked as *ACTIVE*. Similarly, once the service end time reaches, the service is automatically uninstalled from the network and marked as *FINISHED*.

Second, in the asynchronous operational mode, the external element makes the service reservation and receives the acknowledgement from the DynPaC framework as before. The main difference is that once the service start time reaches, the service is not automatically installed in the network. It is necessary an additional state that is only reachable by means of an explicit request from the external element. That way, the services evolve from the *RESERVED* state towards *TOBEINSTALLED* state upon the reception of the *commitService* message. Once the service is in that state, it can be installed in the network at the moment of its start time. In addition, as mentioned before, the asynchronous operational mode also supports the release of the network resources upon the external element's request by means of the *releaseService* message. Upon reception of that request, DynPaC puts the service in *RELEASED* state, where the service is uninstalled from the network but the network resources that the service was consuming remain unchanged. Similarly, DynPaC can re-install the service upon the external element's request by means of the *provisionService* message, where the resources that the service was consuming previously are utilized to continue with the service provisioning.

In summary, the DynPaC framework exposes a REST API originally designed to support the reservation or removal of generic L2 service

demands. Furthermore, it also provides the means to retrieve information about the already reserved services and about the edge nodes with their corresponding ports and VLANs. In addition to these calls, the REST API supports the communication with the NSAs. More specifically, the REST API calls have been designed to support the explicit service provisioning and the release of the resources of an active service. With these API calls, the NSA is able to map the internal NSI-CS states to the appropriate REST API calls so that the work-flow proceeds as specified by the NSI-CS standard [218].

5 Support for advance reservations

> "'Who controls the past, ran the Party slogan, controls the future: who controls the present controls the past."
>
> — George Orwell

As stated in [87], the next generation of TE solutions will leverage the benefits of SDN. All in all, SDN can overcome the long-convergence times of legacy routing protocols thanks to its support for *out-of-band* control of the network elements. In addition, SDN enables the utilization of more realistic traffic splitting techniques that leverage the higher granularity to make forwarding decisions at the network elements. Furthermore, it supports the utilization of dedicated elements to perform the path computation using real-time network state information.

However, as outlined in Chapter III, were a comprehensive study about advance reservation systems has been conducted, there are no proposals taking advantage of these benefits, not even the more recent solutions already based in SDN. Overall, the classical approach to deal with advance reservations consists of per-link resource management and the utilization of C-SPF over pruned topologies.

In such a scenario, this thesis presents a novel advance reservation solution that leverages the centralized nature of SDN. The logically control plane of SDN, which allows to make routing decisions taking into account the overall state of the network, facilitates the adoption of new approaches to tackle the problem of advance reservations.

As such, this chapter presents a novel solution for advance reservations that handles network resources atomically, instead of per-link basis. First, the reservation model used in the DynPaC framework is presented. Second, the algorithm used for the computation of the paths is described. Then, the two data structures that make possible the management of the network resources over time are introduced, namely *Network Snapshots* and *Network Snapshots Tree (NSTree)*. Finally, the operations supported by the NSTree are described in detail.

5.1 RESERVATION MODEL

This section describes the reservation model used in the DynPaC framework, and the timing conditions that need to be fulfilled to accept new service reservation demands into the system.

5.1.1 DEFINITION

In the DynPaC framework, a new service reservation request between two end-points for a certain amount of time is defined as follows:

Definition 1. *Every new service reservation request is identified by the source and destination end-points (node and port), the start time at which the service needs to be provisioned and the end time at which the service needs to be terminated. In addition, the service reservation request is also characterized by the peak bandwidth that will be available through the E2E circuit and the desired failure recovery policy, which can be either path protection or path restoration. It is, therefore, a STSD reservation model.*

5.1.2 TIMING CONSTRAINTS

Each new service reservation request requires the execution of two different phases: *check* and *update*. Firstly, it is necessary to check whether there are enough resources in the network to satisfy the service demand or not (i.e., bandwidth, VLANs, etc.). This phase takes t_{check} time and results in the new service demand being accepted (see Figure 5.1) or rejected (see Figure 5.2).

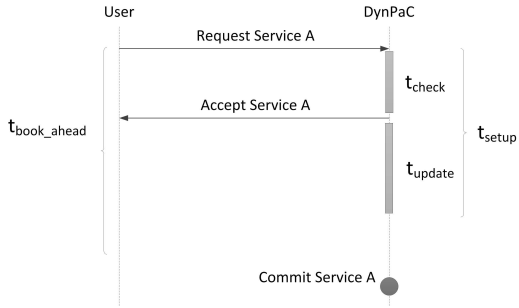


Figure 5.1.: Service is accepted by the DynPaC framework.

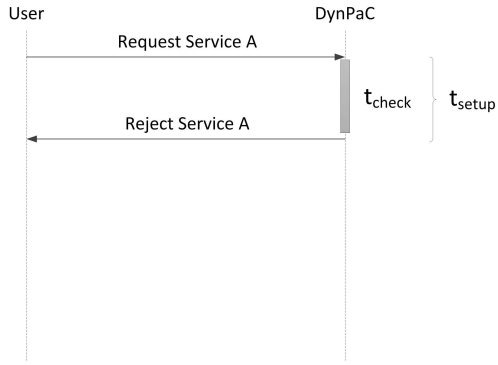


Figure 5.2.: Service A is rejected by the DynPaC framework due to a lack of resources.

Secondly, the optimal path is selected among all the available paths satisfying the connectivity requirement and the necessary data structures are updated, which takes t_{update} time. The second phase is only executed for the accepted services, being p_A the probability of a service being accepted by DynPaC, also known as the *Service Acceptance Ratio (SAR)*. The SAR is one of the most commonly used parameters to evaluate advance reservation systems, since it measures the amount of service reservation requests that are accepted by the system compared to the total amount of service reservations that are requested to the system, as described in equation 5.1:

$$SAR = \sum service_{accepted} \ / \ \sum service_{requested} \tag{5.1}$$

The time required to execute both phases is known as t_{setup}, and sets the time that needs to pass between consecutive service requests in order

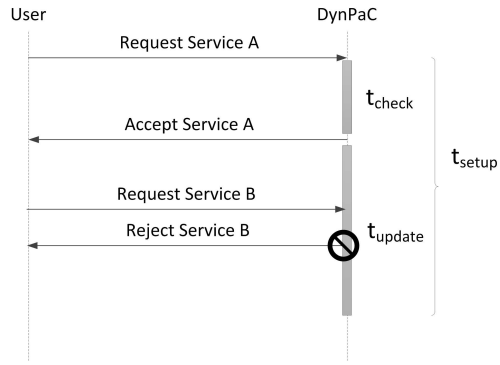


Figure 5.3.: Service B is rejected by DynPaC because it is blocked for new requests.

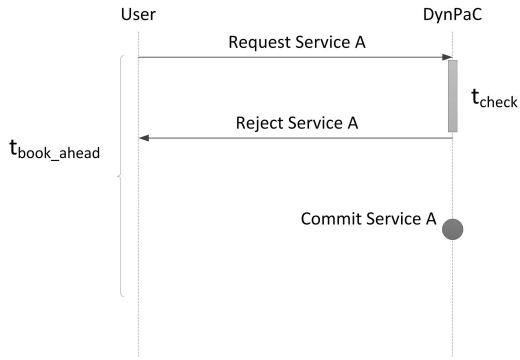


Figure 5.4.: Service A is rejected by DynPaC because the book-ahead interval is not respected.

to guarantee that the system is stable and that the data structures represent the network state correctly. Furthermore, t_{setup} also delimits the time at which the requested service can be committed to the network, which is commonly referred to as the *book-ahead interval* (t_{book_ahead}) [180]. By definition, no service requests will be accepted in the system if no t_{setup} has passed since the last service request, as depicted in Figure 5.3 and no t_{book_ahead} time exists between the time at which the request was received and the start time of the service, as illustrated in Figure 5.4. The relationship between t_{check}, t_{update}, t_{setup} and t_{book_ahead} is defined by equations 5.2 and 5.3.

$$t_{setup} = t_{check} + t_{update} \tag{5.2}$$

$$t_{setup} \leq t_{book\_ahead} \tag{5.3}$$

Since the t_{setup} for a given service depends on whether the service has been accepted or not, we define the average t_{setup} as follows:

$$t_{\langle setup \rangle} = t_{check} + p_A * t_{update} \tag{5.4}$$

5.2 PATH COMPUTATION ALGORITHM

As mentioned before, having an advance reservation solution that leverages the centralized nature of the SDN control plane allows per-network resource management, as opposite to the classical per-link approach. This characteristic of the advance reservation solution presented in this thesis makes possible the utilization of path pre-computation strategies. In a nutshell, the approach followed on the DynPaC framework consists of the pre-computation of a set of possible paths for all the pairs of source and destination nodes. Then, once a particular service reservation is requested, the possible paths are checked against the available resources, until a valid path is found.

It is worth noting, that prior to the path pre-computation, it is necessary to retrieve and abstract the network topology. In that regard, once the topology is retrieved, an undirected graph is formed. This approach allows to simplify the path computation, since in this solution, the service requests are considered symmetrical and bidirectional. That is, for a given service the same bandwidth is reserved for both directions of the communication.

This section describes both algorithms, the algorithm used at the *pre-computation phase* and the second algorithm used at the *on-demand phase*.

5.2.1 PRE-COMPUTATION PHASE

The first phase on the computation of the optimal path for a given service reservation request consists of the computation of the set of possible paths

that will be used to establish the connectivity circuits between particular source and destination nodes. The most basic approach would consist of the computation of all the possible paths between all the possible combinations of source and destination nodes to have the highest possible number of alternative paths to test. However, this approach is not feasible, since obtaining all the possible paths between two nodes is a well-known NP-hard problem.

In order to be computationally feasible, the path pre-computation algorithm needs to compute only a subset of the possible paths in the network. Although this approach makes the utilization of path pre-computation feasible, it also presents some drawbacks that need to be taken into account. First, by pruning the list of possible paths, the DynPaC algorithm can reject a service reservation request even in the case of available resources, that is, the request can result in a *false negative*. This is a typical problem when using heuristics. Notwithstanding, as it will be demonstrated in Chapter VIII, the SAR gets improved with two possible paths, but no significant enhancement is achieved with a higher path diversity. Second, the path pre-computation phase will depend on the use case and the topology of the network, since the better the selection of the possible paths is, the lower the amount of false negatives will be. For the particular case of BoD, the path pre-computation algorithm has been customized taking into account two requirements particular to the use case.

First, services are provided E2E between two end-points, where an end-point is represented either by a combination of node and port, or by a combination of node, port and VLAN tag. In terms of path computation, what this requirement represents is that only the possible paths between pairs of ingress or egress nodes must be computed. It is unnecessary to go through every combination of nodes, since only the ones facing users or external administrative domains can be selected in the service reservation requests. For example, Figure 5.5 depicts the ingress and egress nodes in a network. In that case, in the Network Domain A, the combination of nodes, which is $\binom{n}{k}$, being k the number of nodes being selected from a set of n nodes, gets reduced from $\binom{7}{2} = 21$ to $\binom{3}{2} = 3$.

Second, it is desirable to consume the least possible amount of network resources to provide a connectivity service. Although in a communication

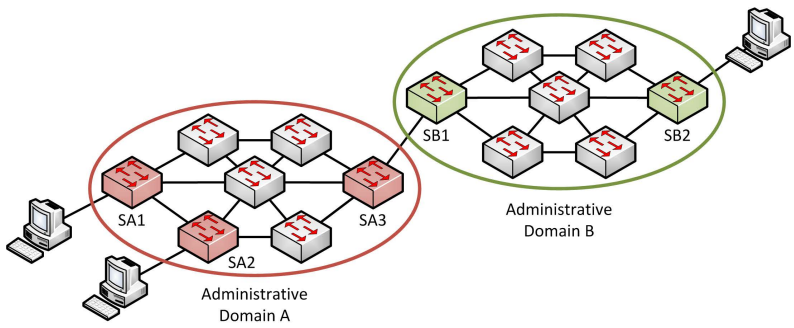


Figure 5.5.: Ingress and egress nodes facing users or other domains in a network.

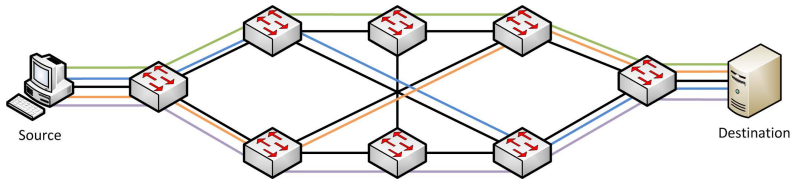


Figure 5.6.: Selection of paths of less than n_{MAX} hops.

network with n nodes a path of $n-1$ hops (also known as traversing edges) is a feasible path, it is not desirable in terms of resource consumption. As a consequence, the algorithm has been designed to only accept the paths with a maximum number of hops, defined as n_{MAX}. The procedure to obtain all the possible paths from 1 to n_{MAX} hops consists on executing a variation of the BFS [219] algorithm that does not keep track of the visited nodes. The algorithm is executed on a loop for every n in the range 1 to n_{MAX}, and all the paths of the same length are stored in an indexed map named *pathMap* to facilitate the future retrieval of the paths of certain lengths per source and destination pair.

Figure 5.6 illustrates a network of 8 nodes with $n_{MAX} = n/2 = 4$ and the paths that the path pre-computation algorithm would compute, which in this case is 4 instead of the 14 available paths with non-repeating edges. Algorithm 1 describes in a simplified manner the procedure followed to compute the set of possible paths to be used in the on-demand path computation phase described in the next subsection.

The structure of the pathMap is detailed below:

```
1 public class PathMap {
        Map<PairOfNodes, PathsList> pathMap;
        private class PairOfNodes {
```

Algorithm 1 Path Pre-Computation Algorithm

```
1: function PathPreComputation(topology)  ▷ Computes a set of possible paths
   between pairs of source and destination nodes
2:     nodes ← filterEgressNodes(topology)
3:     for all pair of nodes do
4:         for all i ← 1, n_MAX do
5:             possiblePaths ← computePaths_i(topology, srcNode, dstNode)
6:         end for
7:         pathLists[i] ← possiblePaths
8:     end for
9:     pathMap[srcNode, dstNode] ← pathLists
10: end function
```

```
                Node source;
                Node destination;
6           }
            private class PathsList {
                Map<Integer, List<Paths>> pathList;
            }
}
```

5.2.2 ON-DEMAND PHASE

The second phase of the path computation consists on the randomization of the list of the possible paths provided by the PCE for a given pair of source and destination nodes. As mentioned before, the PCE in the pre-computation phase generates a map in which it stores a list of possible paths per each combination of ingress and egress nodes, arranged taken into account the number of hops. Figure 5.7 depicts the content of the pathMap that is computed by the PCE in the pre-computation stage for the specific network topology included in the example.

When a new service reservation occurs, the service manager requests the list of possible paths to the PCE, by simply specifying the source and destination nodes. Then, the PCE checks the pathMap and returns the list of possible paths for that pair of nodes. It is worth noting that this approach results in the PCE providing identical path lists for services originating and terminating in the same nodes. As a consequence, there is an increasing possibility of congesting some network resources

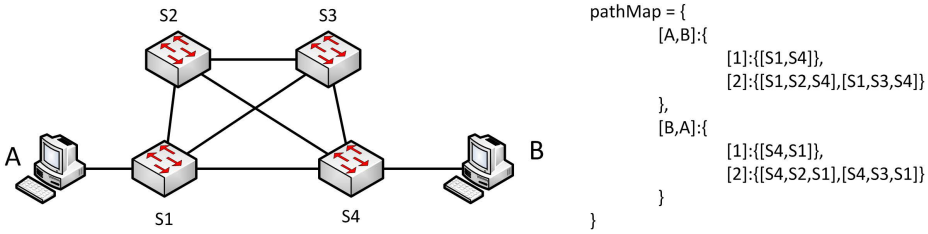


Figure 5.7.: Structure to store the pre-computed paths.

while others remain underutilized. These kinds of algorithms are usually referred to as *greedy algorithms*, since they always try to provide the best path to every request, which in turn results in the congestion of those paths.

In order to solve the *greedy algorithm* problem, the algorithm used in the on-demand phase randomizes the list of possible paths so that requests with the same source and destination nodes are assigned with a different list of possible paths. Please note that the set of possible paths is the same, what varies is the order in which the paths are listed. The randomization is applied to the set of paths of the same length, in order to prioritize at all times the shortest paths. This is precisely the reason why the pathMap arranges the list of paths per each source and destination pair using blocks of paths of the same length. The procedure to obtain the randomized list of possible paths is described in Algorithm 2.

Using the network described in Figure 5.7, the following example illustrates the output of the On-Demand Path Computation Algorithm

Algorithm 2 On-Demand Path Computation Algorithm

```
1: function OndemandPathComputation(srcNode, dstNode)   ▷ Provides an
   ordered list of possible paths between the given source and destination nodes
2:     pairOfNodes ← [srcNode, dstNode]
3:     listPaths ← pathMap(pairOfNodes)
4:     pathsList ← []
5:     for i ← 1, n_MAX do
6:         paths ← randomizePathList(listPaths.get(i))
7:         pathsList.append(paths)
8:     end for
9:     return pathsList
10: end function
```

for two consecutive path computation requests using the same source and destination nodes as parameters.

```
Request 1: OndemandPathComputation(A,B)
Output:
pathsList = {
        [S1,S4],
        [S1,S2,S4],
        [S1,S3,S4]}

Request 2: OndemandPathComputation(A,B)
Output:
pathsList = {
        [S1,S4],
        [S1,S3,S4],
        [S1,S2,S4]}
```

5.3 DATA STRUCTURES FOR ADVANCE RESERVATIONS IN SDN

This section presents the data structures that make possible the support for advance reservations in the DynPaC framework. The solution is compliant with the reservation model described in Section 5.1 and satisfies the objectives presented in Chapter I. It is based on the utilization of two data structures and a set of algorithms specifically designed to support advance reservations in SDN. Firstly, the *network snapshot* data structure is used to represent the network state in terms of resource consumption at a specific period of time. Secondly, the *NSTree* data structure provides the means to arrange multiple network snapshots making possible the efficient reservation of new service requests.

5.3.1 NETWORK SNAPSHOT

One of the main contributions of the proposed solution is the utilization of *network snapshots*, which are defined as follows:

Definition 2. *A network snapshot represents the state of the network during a specific period of time in terms of network resource consumption.*

It is identified by a unique identifier and its start and end times. A network snapshot stores the following information:

- *Identificator (ID): Unique identifier.*
- *Start Time (sT): Time at which the services that coexist in the network snapshot and that are not already installed must be installed in the network elements.*
- *End Time (eT): Time at which the services that coexist in the network snapshot and that do not continue in the next network snapshot must be removed from the network elements.*
- *Concurrent Services: Connectivity services to be provided simultaneously during a specific period of time.*
- *Consumption Vector: Vector that summarizes the resource availability in terms of available bandwidth per network edge in a given network snapshot. Depending on the use case additional information such as VLAN tag availability can be stored.*
- *Optimal Paths: A map that stores the optimal path for a given service in a given network snapshot.*
- *Relocation Alternatives: A set of alternatives to rearrange the services into alternative paths.*

In other words, a network snapshot represents the available resources in the whole network and provides information about the concurrent services during a specific period of time. This approach is possible thanks to the centralized nature of SDN. It differs from the classical approach in advance reservation systems (see Chapter III), where the network resource consumption is kept for each link of the network independently. As a result of that approach, each link can have different resource consumption variations over time, depending on the flows carried by each link at each time interval.

On the contrary, the approach presented in this thesis makes easier the utilization of more flexible data structures, without predetermined time slots that limit the time for which the system accepts reservations in advanced and making possible the computation of the paths prior to the reservation itself, resulting in a lower t_{setup}.

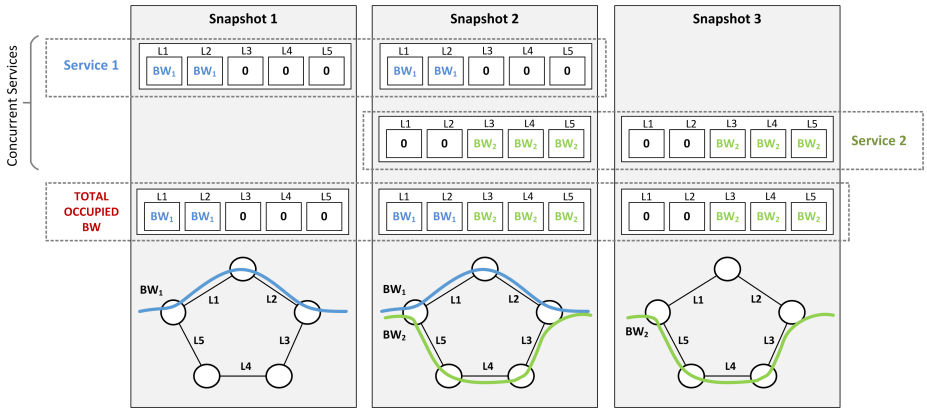


Figure 5.8.: Network resource consumption evolution represented with network snapshosts.

Figure 5.8 depicts the evolution of the network resources consumption when two services are requested, generating three different network snapshots, the first one when only *service 1* exists, the second one when *service 1* and *service 2* coexist and the third one when only *service 2* exists.

5.3.2 NSTREE

It is a characteristic of advance reservation systems the need to keep track of all the bookable resources over time. As a consequence, one of the main problems of these kinds of systems is that checking for available resources needs to be done for each time interval affected by the new service request, in this case, for each network snapshot. In such a scenario, this section presents the NSTree, a custom data structure designed to support efficient service reservations in SDN.

In general, since a new service reservation request can overlap multiple network snapshots, each of them must be analyzed independently. If and only if all the overlapped network snapshots have enough resources to accept the new demand, the requested service is accepted. Since t_{check} impacts directly t_{setup}, the minimization of t_{check} contributes to the minimization of t_{setup}. The NSTree has been designed to prioritize the minimization of t_{check} over the minimization of t_{update}. On the one hand, because the checking phase is executed both for accepted and rejected

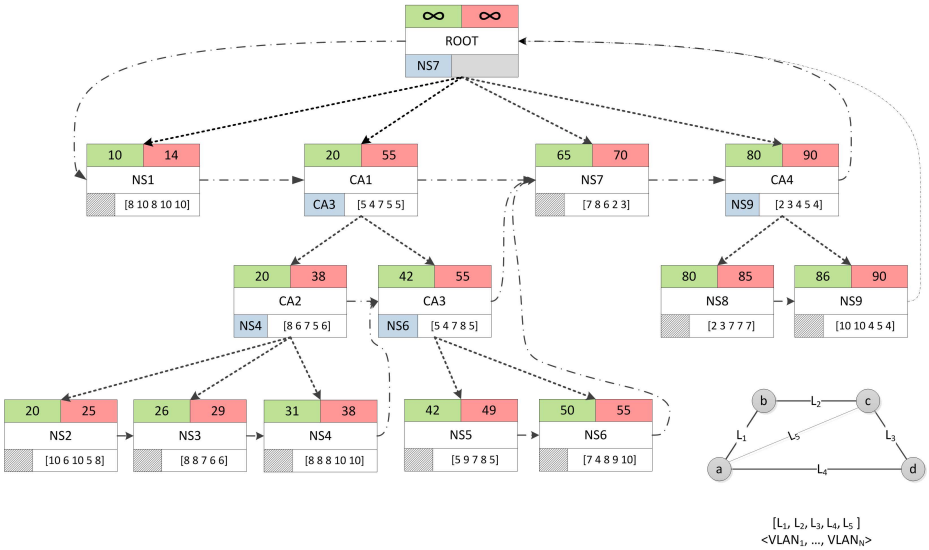


Figure 5.9.: Design of the NSTree data structure.

service requests. On the other hand, because it is desirable to tell the user as soon as possible whether the demand can be satisfied or not.

The easiest way to arrange the network snapshots is using a chronologically ordered list. In that case, the checking phase takes up to $\mathcal{O}(n)$, being n the number of network snapshots in the list. Taking into account that each service request can generate up to $2n + 1$ new network snapshots and that the framework presented in this paper is expected to handle thousands of service requests per month [220], it is of uttermost importance to reduce the time of the checking stage.

The NSTree presented in this section arranges the network snapshots in a way that reduces the amount of network snapshots being analyzed at the check phase, with a minimal penalization in terms of memory consumption due to the utilization of auxiliary nodes. The structure of the NSTree is depicted in Figure 5.9.

5.3.2.1 NODE TYPES

The NSTree is comprised of three different types of nodes, namely *root*, *Network Snapshot (NS)* and *Common Ancestor (CA)*. Each node in the NSTree keeps track of the information specified in the network snapshot definition of the previous section. In addition, in order to arrange the

different nodes and make feasible the traversal of the NSTree, each node keeps track of the following information:

- *Next Node:* A pointer to the next node in the NSTree.
- *Parent:* A pointer to the parent of the node.
- *Children:* List of pointers to the child nodes of the given node.
- *Second Pointer Node (SPN):* A pointer to a node in the NSTree that allows to split the NSTree in two halves.

The following subsections describe the different types of nodes in detail, where it is also specified the default values utilized for a subset of the fields of the network snapshot data structure.

- *Root*

An auxiliary node that represents the root of the tree. By definition, only a single node of this type can exist in the NSTree. To ease computation, the root is considered to have an start time and an end time equal to infinite (which in practice takes the highest value of the data type used to represent time). This allows to use the root node as a regular node in the NSTree and facilitates the insertion of the first and the last nodes in the NSTree. As opposite to the NSs and CAs, it does not keep track of any resource consumption or concurrent services. In summary, the root node is characterized as follows (please note that only the most relevant fields are included):

- *ID:* ROOT
- *sT:* ∞
- *eT:* ∞
- *Concurrent Services:* Not Applicable (NA)
- *Consumption Vector:* NA
- *Optimal Path:* NA
- *Parent:* NA
- *Relocation Alternatives:* NA

- *Network Snapshot*

A node in the tree that represents a network snapshot. It is identified by its start and end times and keeps track of the resources that are available in the period of time that it represents. As specified in Section 5.3.1, the resource availability is represented through a *consumption vector*, where each index at the consumption vector provides information about the amount of available bandwidth in a specific link. Each NS points to the next node in chronological order, which can be either another NS or a CA. In addition, it also points to its parent node, which in this case can be the root node or a CA, but never an NS. A particularity of the NS nodes is that they do not have any children nor an SPN, since they are always leafs in the NSTree. In summary, the NS nodes are characterized as follows (please note that only the most relevant fields are included):

- *ID*: NS + <No>
- sT: start time of the concurrent services
- eT: end time of the concurrent services
- *Children:* NA
- *SPN:* NA

- *Common Ancestor*

An auxiliary node that aggregates a set of consecutive children nodes (which can be both NS or CA nodes). It represents a wider time interval where a set of common resources are ensured to be available through the whole time interval, regardless of network snapshot variations. It is identified by a start time, equal to the minimum start time among the start times of its children nodes and an end time, equal to the maximum end time among the end times of its children nodes.

It contains a list with its children nodes arranged in chronological order and keeps track of the consumed resources with a consumption vector built from the minimum values of the consumption vectors of all its children nodes. As the NS nodes, it has a pointer to the next node in chronological order, which can be both a CA or an NS. More precisely,

the next node is the next node of its last child in chronological order. In addition it also has a pointer to its parent node, which can be either the root or a CA node and a pointer to the SPN that splits the sub-tree in two halves of equal amount of NS nodes. It is worth noting that the CA does not keep a list of the optimal paths nor of the relocation alternatives, since the same service can be provided through different paths in NS nodes aggregated under the same CA. In summary, the CA nodes are characterized as follows (please note that only the most relevant fields are included):

- *ID:* CA + <No>
- *sT:* start time of the first child
- *eT:* end time of the last child
- *Concurrent Services:* NA
- *Consumption Vector*: computed using Algorithm 3
- *Optimal Path:* NA
- *Relocation Alternatives:* NA

As a remark, the utilization of SPNs allows a more efficient traversal of the NSTree. Each node in the NSTree of type *root* or *CA* contains two pointers that split the tree in two halves, each with the same amount of nodes of type *NS* (in case of an odd number of NS nodes, the first half is always the biggest one). The first pointer points to the first NS node in chronological order and the second pointer points to the node ensuring

Algorithm 3 CA consumption vector computation algorithm

```
1: function CAConsumptionVector(CA)
2:     vector ← []                                   ▷ Empty array
3:     children ← children of the CA
4:     for i ← 1, number of links do
5:         vector[i] ← min_{∀c∈children} vector_c[i]
6:     end for
7:     propagate consumption vector upwards
8:     return vector
9: end function
```

the same amount of nodes of NS type in the two halves, that is, the SPN. In order to determine at which node the algorithms in charge of handling the reservation or removal of the services must start, it is only necessary to store the start time of the SPN. This way, by simply comparing the start time of the new service reservation and the start time of the SPN it can be ensured that only the correct half is analyzed.

5.3.2.2 NODE GENERATION PROCESS

Every time a new service reservation is done, the NSTree needs to be updated to reflect the new resource availability for future requests, which results in new NS nodes being added or updated at the NSTree. In addition, the generation of new NSs may require the generation of CA nodes for their aggregation. This characteristic of the NSTree makes necessary the evaluation of consecutive nodes, in order to determine if they have to be aggregated into a new CA or an existing one. As a consequence, the algorithms described in Section 5.4 have been designed to operate on a per-NSTree node basis.

Every time a new service reservation is requested, it is compared chronologically against all the nodes that it overlaps in time. In order to handle this efficiently, a *Divide & Conquer*-like approach is followed, where the original problem is divided into multiple smaller sub-problems[1]. In every step of the process, a subset of the time for which the service reservation has been requested is analyzed, which is referred to as Analyzed Time Slot (aTime). The aTime comprises the period of time between the end time of the Predecessor NSTree Node (pNode) (the node analyzed in the previous step) and the end time of the Analyzed NSTree Node (aNode) (the node being analyzed in the current step). Once the analysis of a given aTime has finished, that period of time is subtracted from the service reservation request, in order to continue analyzing the next node in chronological order. It is worth noting that in every step of the process, the analysis encompasses two well defined phases. First, the lapse of time between the end time of the pNode and the start time of the aNode is tackled, since it requires the generation of a new NS node. Second, the lapse of time corresponding to the duration of the aNode

1 Please note the Divide & Conquer paradigm uses recursion to solve problems, technique that it is not followed here

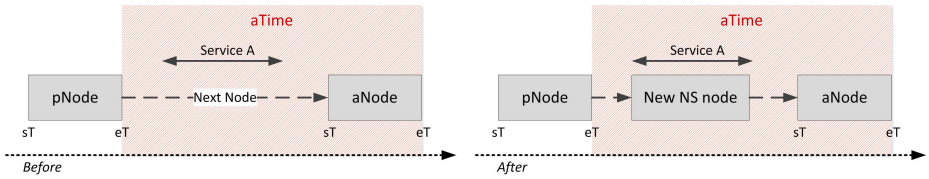


Figure 5.10.: Node generation process case 0: no overlapping.

itself is analyzed, phase that requires updating the aNode in order to update the resource availability to reflect the resources consumed by the new service reservation, and the generation of additional NSs depending on how the service overlaps the aNode.

This approach allows to reduce the node generation or update to five different cases, which differ from each other depending on how the service reservation overlaps the aTime. The five cases considered in this solution are (Case 0) No overlapping, (Case 1) Back overlapping, (Case 2) Partial overlapping, (Case 3) Front overlapping and (Case 4) Full overlapping. The case of post-overlapping[2] is not considered, since it can be seen as a no overlapping case over the next node being analyzed. This is achieved by considering the root as the pNode of the first node in chronological order and the last aNode. Please note that during the same aTime it is possible to have simultaneously a Case 0 and another case from the remaining ones.

Case 0: No overlapping

This case correspond to service requests whose start time is lower than the start time of the aNode, which is depicted in Figure 5.10. In this case, it is necessary to create a new NS, since a gap between the pNode and the aNode represents a period of time in which no services are reserved. That is, the gap represents a period of time where the network is empty and no network resources are being consumed. Therefore, once the *new NS* is created, only the resources consumed by the new service request will exist.

During the analysis of the aTime, it can occur to have a service reservation request whose end time is higher than the start time of

2 A post overlapping will correspond with a situation in which the service reservation ends after the aNode

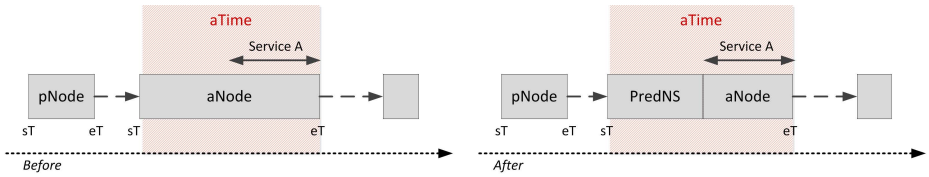


Figure 5.11.: Node generation process case 1: back overlapping.

the aNode. When this happens, the period of time related to the no overlapping case is subtracted from the service reservation and the aTime is updated, in order to continue with the analysis, since a Case 3 or a Case 4 can occur.

Case 1: Back overlapping

When the service reservation request's start time is higher than the aNode start time, the aNode needs to be split in two, as depicted in Figure 5.11. First, a new NS is created equal to the aNode but with an end time equal to the start time of the service request minus one. This node is named Predecessor Network Snapshot (PredNS) and represents the part of the original aNode in which the service did not overlap. Second, the aNode is shortened, with a new start time equal to the start time of the service request. In this case, since a new service needs to coexist with the previously reserved services in that node, the resources are updated, in order to reflect the new situation. Once the nodes are created or updated, the pointers that allow to traverse the NSTree in chronological order are updated. The PredNS becames the next node of the pNode, while it points to the aNode. Please note that when the service reservation's end time is higher than the aTime's end time, the analysis continues with the next node in the NSTree. This involves the aNode being considered the new pNode and the next NSTree node in chronological order being treated as the new aNode, resulting in the aTime being updated to reflect the new analyzed time span.

Case 2: Partial overlapping

The case of partial overlapping reflects the case in which the service request fully overlaps the aNode but the start time of the service is higher

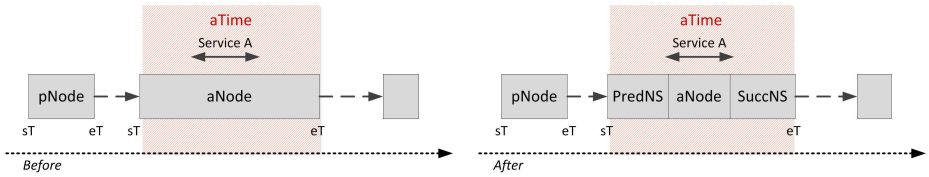


Figure 5.12.: Node generation process case 2: partial overlapping.

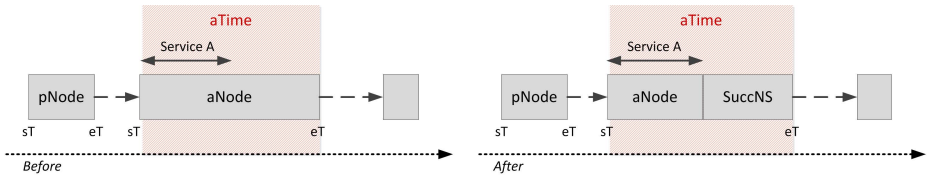


Figure 5.13.: Node generation process case 3: front overlapping.

than the start time of the aNode and the end time of the service is lower than the end time of the aNode, as depicted in 5.12. This situation results in the aNode being split in three nodes. The first one is the PredNS, and it is a copy of the aNode but with the end time equal to the service start time minus one. The second node is the aNode itself, whose start and end times are updated to match the ones of the service. This is the only node in which the service request needs to be added in order to reflect the new resource availability. Finally, as in the case of the PredNS, the Successor Network Snapshot (SuccNS) is a copy of the aNode prior to the insertion of the new service, with a start time equal to the service end time plus one. As in the previous case, the pointers are updated in order to ensure the chronological ordering.

Case 3: Front overlapping

In this case, the start time of the service reservation request corresponds with the start time of the aNode, as illustrated in Figure 5.13. However, the end time of the service reservation request is lower than the end time of the aNode, which results in the original aNode being split in two. On the one hand, the original aNode is shortened, with an end time equal to the one of the service reservation request. After this, the service is added to the aNode, and the resources of this node updated. On the other hand, the remaining part of the aNode that is not overlapped by the service

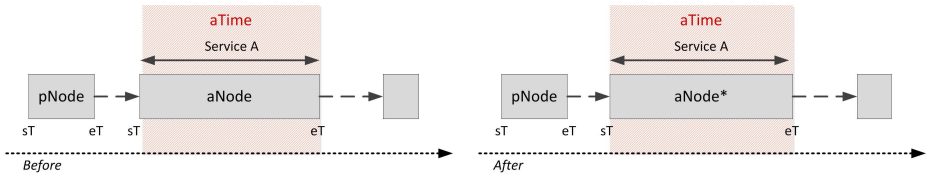


Figure 5.14.: Node generation process case 4: full overlapping.

reservation conforms a new node, referred to as SuccNS, which maintains the same resource availability that the aNode previous to the addition of the new service. Of course, the pointers that allow to traverse the NSTree in chronological order are updated, in such a way that now the aNode points to the SuccNS and the latter one points to the node that the aNode was pointing prior to the split.

Case 4: Full overlapping

The final case corresponds with the case in which the service reservation fully overlaps the aNode, with the particularity that both the service and the aNode have identical start and end times, as shown in Figure 5.14. In this case, the service is added to the node, and the start and end times of the aNode and the pointers remain unchanged.

5.3.2.3 NODE AGGREGATION POLICY

The utilization of the CA nodes imposes a penalization in memory and they are only useful when they allow to reduce the amount of NSs being analyzed when a new service reservation is requested. Hence, the amount of CA nodes that are created in the NSTree needs to be kept to the minimum in order to improve the t_{setup}. For instance, considering a binary balanced NSTree, where each CA would be the combination of two child nodes, the amount of CA nodes generated would be very big and not necessarily beneficial. The criteria used to aggregate multiple NSTree nodes into a CA depends on the nature of the connectivity service being provided and the use case. As such, this section presents the node aggregation policy used in the advanced reservation system presented in this thesis.

In advance reservation systems, particularly in the ones used in the REN environment, services are usually requested for a long period of time. As a result, it is highly probable for service reservations separated less than the average duration of the service requests to be simultaneously overlapped by new demands. Therefore, and taking into account this particularity of advance reservation systems in the REN environment, it has been decided to use a time-based aggregation policy. Two consecutive nodes, either two NSs, two CAs or an NS and a CA are paired together into the same CA as defined below:

Definition 3. *Two consecutive nodes can be aggregated into a CA if and only if there is less than δ between the end time of the former node and the start time of the latter node.*

Therefore, δ defines the minimum amount of time that needs to exist between two consecutive nodes not to be aggregated into a common CA. With such a strategy, every time a new node is created in the NSTree, it is necessary to check whether a rearrangement of the NSTree is necessary or not. That is, it is necessary to check if new CAs have to be created or if existing ones have to be updated with new child nodes.

The procedures used for the rearrangement of the NSTree always consider the separation between the pNode and the new node for the formation of the CAs. The aggregation is performed when there is at most δ between the end time of the pNode and the start time of aNode. The methods utilized for the rearrangement of the NSTree are described in Algorithm 4, which differentiates the following cases.

Case A: Consecutive Network Snapshots aggregated into the same Common Ancestor

As will be later explained in Section 5.4, when a new NS is created, it inherits the parent of its pNode. As such, it is possible to have two consecutive nodes separated less than δ automatically aggregated into a common CA[3]. As depicted in Figure 5.15, when a new NS is created, identified as A, if the parent of the predecessor NS identified as P is already a CA, the new NS is automatically aggregated into the CA, with

3 Please note that in order to inherit the a parent CA it is check if the time span between the nodes is less than δ

Algorithm 4 NSTree rearrangement method

```
1: function RearrangeTree(pNode, aNode)
2:     if pNode.eT + δ > aNode.sT then
3:         if Case B then
4:             pNodeBis ← copy(pNode)
5:             pNode ← upgrade2CA(pNode)
6:             pNode.addChildren(pNodeBis, aNode)
7:         end if
8:         if Case C then
9:             ca ← pNode.getParent()
10:            ca.addChildren(aNode)
11:        end if
12:        if Case D then
13:            pNodeBis ← copy(pNode)
14:            pNode ← upgrade2CA(pNode)
15:            ca ← aNode.getParent()
16:            pNode.addChildren(pNodeBis, ca.getChildren()
17:        end if
18:        if Case E then
19:            ca1 ← pNode.getParent()
20:            ca2 ← aNode.getParent()
21:            ca1bis ← copy(ca1)
22:            ca ← upgrade2CA(ca1)
23:            ca.addChildren(ca1bis, ca2)
24:        end if
25:    end if
26: end function
```

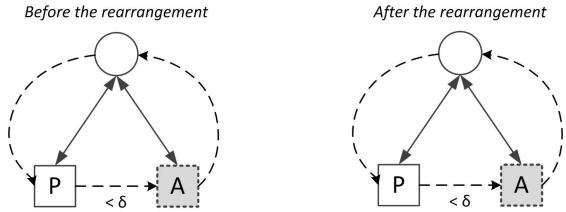


Figure 5.15.: Rearrangement Case A: Two consecutive NS aggregated into the same CA

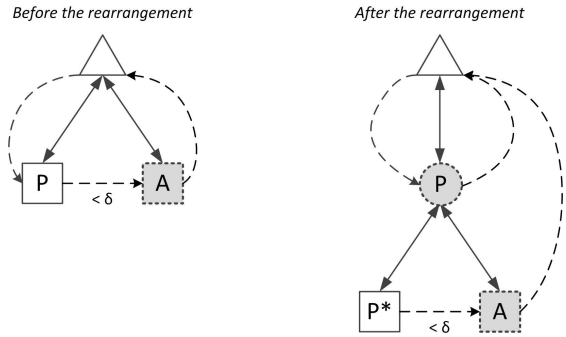


Figure 5.16.: Rearrangement Case B: Two consecutive NS children of the root node.

no further rearrangements required if the time lapse between A and P is shorter than δ.

Case B: Consecutive Network Snapshots children of the root node

In this case, as depicted in Figure 5.16, the pNode marked as P and the new node identified as A are both children of the *root* node. When less than δ exists between the two nodes, it is necessary to create a new CA in order to aggregate them. The approach followed in this solution is to upgrade the pNode to became a CA, which allows to keep the pointers that guarantee the chronological ordering. Then, a copy of the pNode identified as $P*$ in the figure is aggregated as first child of the CA, while the new node is aggregated as the second child.

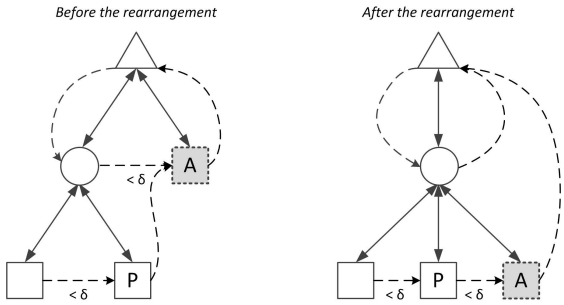


Figure 5.17.: Rearrangement Case C: Predecessor Network Snapshot aggregated into a Common Ancestor.

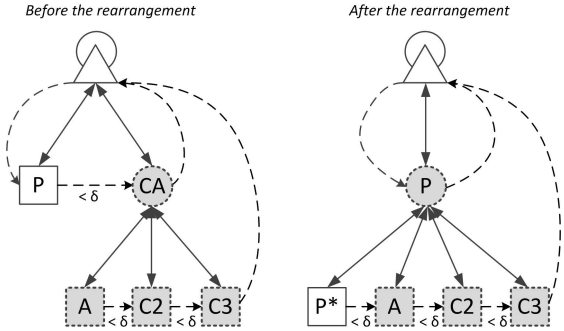


Figure 5.18.: Rearrangement Case D: New node aggregated into a Common Ancestor.

Case C: Predecessor Network Snapshot aggregated into a Common Ancestor

The case when the pNode is aggregated into a CA but the new node is a child of the *root* node is illustrated in Figure 5.17. When there is less than δ between the pNode, marked as P, and the new node marked as A, it is necessary to convert the new node in a child node of the CA. Once that is done, the CA node is updated with the resources consumed by the new node and the pointer to the next node of the CA is retrieved from the new node.

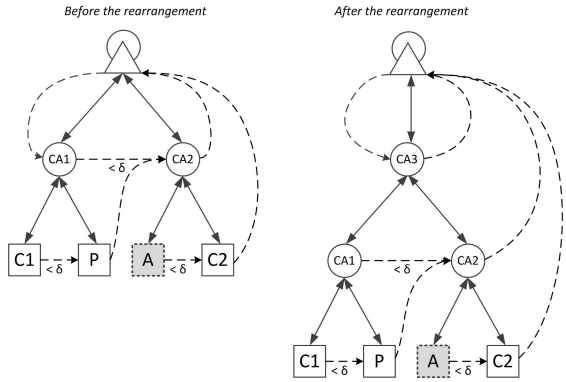


Figure 5.19.: Rearrangement Case E: Consecutive nodes aggregated into different Common Ancestors.

Case D: New node aggregated into a Common Ancestor

In this occasion, the pNode identified as P in Figure 5.18 is a child of the *root* node, while the new node, identified as A is aggregated into a CA. In order to ensure the chronological ordering, it is necessary to upgrade the pNode to become a CA. Then, a copy of the pNode identified as $P*$ is added as the first child of the newly created CA, while the new node and the rest of the children of the initial CA are transfered to the new one. Once all the children have been aggregated into the new CA, the initial CA is removed from the NSTree.

Case E: Consecutive nodes aggregated into different Common Ancestors

Finally, as depicted in Figure 5.19, when the pNode identified as P is aggregated into $CA1$ but the new node identified as A is aggregated into a different CA, identified as $CA2$, it is necessary to create a higher height CA to aggregate both, $CA1$ and $CA2$, identified as $CA3$ in the figure. This situation can occur when there is a Case 0 (no overlapping) followed by a Case 1 (back), Case 2 (partial) or Case 3 (front) overlapping. In this case, the new NS node generated in the Case 0 can be aggregated in the $CA1$ and the aNode has resulted in the generation of the $CA2$ as a consequence of the aNode overlap. Although the most simple approach would be to transfer the child nodes of the $CA2$ to $CA1$, the algorithm

checks whether the number of child nodes in both $CA1$ and $CA2$ is similar or on the contrary, is unbalanced. In the latter case, it is preferable to keep the pNode aggregated in $CA1$ and the aNode aggregated in $CA2$. Notwithstanding, since less than δ exists between pNode and aNode, it is guaranteed that less than δ exists between $CA1$ and $CA2$, therefore being possible the aggregation of these two CA nodes into the higher length $CA3$.

5.4 OPERATIONS SUPPORTED BY THE NSTREE

The set of operations that need to be supported by the NSTree data structure are the *reservation of a new service*, the *removal of a given service* and the *programmatic removal of a network snapshot*, which are detailed below. Please note that the *programmatic installation of a network snapshot* is not described because it does not require any modifications in the NSTree.

5.4.1 RESERVATION OF A NEW SERVICE

In our reservation model, every time a new service reservation is requested, the system must first check in every affected NS whether there are enough resources to satisfy the demand or not, and if positive, update the NSTree accordingly to reflect the new situation. These two phases, *check* and *update*, are described in the following subsections.

5.4.1.1 PHASE I: CHECK

The approach followed to determine whether there are enough resources or not consists of traversing the tree chronologically, starting from the correct half. Every time a node overlaps in time with the service reservation request, it is necessary to check whether the node has enough resources to accept the request or not during the time interval that it represents. This is achieved by comparing the consumption vector of the node with the vector describing the resource consumption of the service using a specific path. Since the PCE provides a set of possible paths for every service request, ordered according the the number of hops, the algorithm checks all the possible paths, until a suitable one is found. Once a suitable

path is found, the time interval analyzed is subtracted from the original service request. This procedure is repeated, until there is no time pending to be analyzed.

The algorithm ends with a negative result when a NS does not have enough resources to provide the service. Otherwise, the algorithm considers that the service demand can be satisfied. The output of the algorithm will be false if the service is rejected and true if the service is accepted. In the last case, the algorithm also returns the predecessor node, which is the last node not affected by the new service reservation request. This approach allows to minimize the NSTree traversal in the next phase.

In order to keep track of the time interval pending to be analyzed we use the Remaining Service Time (RST) variable. The RST is a duple that specifies the sT and the eT of the time interval pending to be analyzed. RST is initialized with the start and end times of the requested service, and the sT is updated every time it is checked that a NS or a CA have enough resources. The details of the first phase are shown in Algorithm 5. It is worth mentioning that prior to the execution of the algorithm a procedure is executed to discard the service reservation requests that do not satisfy the timing constraints presented in Section 5.1 or exceed the resources handled by the advance reservation system (i.e., VLAN tags out of a specific range or bandwidth reservation requests that exceed the maximum bandwidth of the network).

5.4.1.2 PHASE II: UPDATE

Every service reservation request accepted by the advance reservation system requires the update of the NSTree. As mentioned before, the pNode returned by Algorithm 5 is used to start the NSTree traversal that allows us to perform the necessary updates. The algorithm (described in Algorithm 6), uses the RST variable initialized with the service start time and end time to know when to terminate the update phase.

In a nutshell, the algorithm starts by initializing the RST variable and obtaining the aNode, which is the next node in chronological order of the pNode. As long as the RST is not empty, the algorithm will check first if a new NS node needs to be created in the gap between the end time of the pNode and the start time of the aNode. This step corresponds to

Algorithm 5 Check resources algorithm

```
1: function CHECK(service, NSTree)
2:     RST ← (service.sT, service.eT)
3:     aNode, pNode ← root
4:     nodes ← [ ]
5:     predecessorFound ← false
6:     children ← getChildrenSinceRST(aNode)
7:     if !children is empty then
8:         nodes.push(children)
9:     end if
10:    while RST and nodes are not empty do
11:        if !predecessorFound then
12:            pNode ← aNode
13:        end if
14:        for all aNode ← nodes do
15:            if !aNode overlaps RST then
16:                clear RST
17:                predecessorFound ← true
18:            else
19:                if aNode has resources then
20:                    update RST
21:                    nodes.remove(aNode)
22:                    predecessorFound ← true
23:                else
24:                    if aNode is CA then
25:                        children ← getChildren(aNode, RST.sT)
26:                        nodes.push(children)
27:                    else
28:                        return null
29:                    end if
30:                end if
31:            end if
32:        end for
33:    end while
34:    return pNode
35: end function
```

the *no overlapping* case described in Section 5.3.2.2. If positive, a new NS is created and the appropriate NSTree rearrangement is performed, as described in Algorithm 4.

Once the *no overlapping* has been solved and the RST variable has been updated by removing the time interval corresponding to the new NS node, it is necessary to continue with the analysis of the same aNode. Since updating the RST guarantees that no further *no overlapping* is possible until the end time of the aNode, the analysis continues by considering the rest of the cases (1 to 4).

When the RST overlaps the aNode additional nodes will be created, with the exception of the *full overlapping*. As such, the particularities of each of the cases need to be handled properly. On the one hand, in the case of a *back overlapping* a PredNS is created as a copy of the aNode. On the other hand, in the case of a *front overlapping* a SuccNS is created as a copy of the aNode. In case of a *partial overlapping* both nodes are created. In any case, the aNode is left as the only node to be updated by the addition of the new service. Once the nodes have been created, the algorithm for the rearrangement of the NSTree is executed again. It is worth noting that when a SuccNS or a PredNS is created, these can be aggregated in conjunction with the aNode into a CA. Once the overlapping of the aNode has been handled, the RST is updated, to continue with the analysis of the rest of the nodes in the NSTree affected by the new service request.

Figure 5.20 illustrates the case when the aNode is a child of the root node, which requires the upgrade of the aNode and the addition of the PredNS represented as $A-$, a copy of the aNode, represented as $A*$ and the SuccNS represented as $A+$ as children. Please note that the figure is illustrating the case of the *partial overlapping*, for a *back overlapping* or a *front overlapping* the SuccNS or the PredNS will be missing respectively.

Similarly, Figure 5.21 depicts the case in which the aNode is already the child of a CA. In this case the rearrangement is simpler, since only the PredNS and the SuccNS need to be added as children of the same CA. The procedures to perform such rearrangement are described in Algorithm 7.

Once the new nodes have been created and the rearrangement of the NSTree has been performed and the NSTree is balanced, it is necessary to update the affected nodes with the addition of the new service. This means that not only the NSs need to be updated, but also the CAs of

Algorithm 6 NSTree update for accepted services algorithm

```
1: function NSTreeUpdateAccept(service, NSTree, pNode)
2:     RST ← (service.sT, service.eT)
3:     aNode ← pNode.getNext
4:     solvedNodes = [ ]
5:     while RST is not empty do
6:         if aNode is solved then
7:             update RST
8:             continue with next node
9:         else
10:            if Case 0 then
11:                create newNS
12:                add service to newNS
13:            else
14:                if Case 1 then
15:                    create PredNS from aNode
16:                end if
17:                if Case 2 then
18:                    create PredNS from aNode
19:                    create SuccNS from aNode
20:                end if
21:                if Case 3 then
22:                    create SuccNS from aNode
23:                end if
24:                update aNode
25:                rearrangeTree()
26:                add service to aNode                ▷ Includes Case 4
27:                balanceTree()
28:                pNode ← aNode
29:                aNode ← aNode.nextNode
30:            end if
31:            update RST
32:        end if
33:    end while
34: end function
```

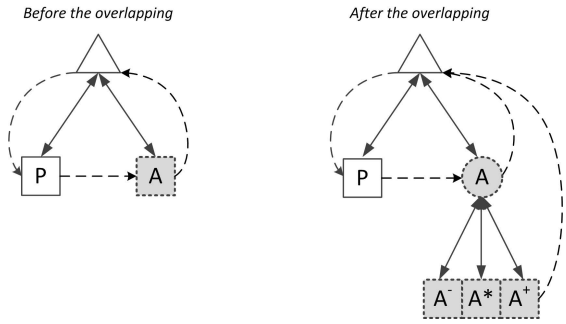


Figure 5.20.: Creation of a CA from a NS with root parent.

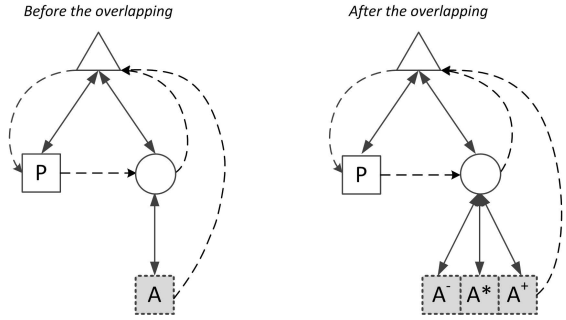


Figure 5.21.: Creation of a CA from a NS with CA parent.

which the NSs are descendant of. The procedure used to add the service is described in Algorithm 8.

In order to ensure that the services are not being switched constantly from one path to a different path when the network snapshot changes, the algorithm uses two strategies. Firstly, it always tries to assign the most stable path for a given service request. That is achieved by using the aggregated utilization information contained at the CAs as long as it is possible. Therefore, it is necessary to go from the NS being updated to the CA with the highest height and add the service using the CA consumption vector. If there is a common solution, all the children nodes affected by the service request are marked as solved after propagating the solution for the service downwards.

It is worth noting, that once the solution is spread, a set of relocation alternatives can be generated for each NS, which will be different since a different set of concurrent services exists in each of them. The relocation alternatives are generated to support the relocation of flows

Algorithm 7 Create a new CA algorithm

```
1: function CreateCA(PredNS, node, SuccNS)
2:     parent ← node.parent
3:     if parent is not CA then
4:         parent ← newCA
5:         node.parent ← parent
6:         newCA.addChildren(node)
7:     end if
8:     PredNS.parent ← parent
9:     SuccNS.parent ← parent
10:    parent.addChildren(PredNS)
11:    parent.addChildren(SuccNS)
12:    return parent
13: end function
```

into alternative paths, as it will be further explained in Chapter VI. Secondly, it tries to propagate the solution that is already valid for the same service in the pNode into the newly created NSs. Please note that in Case 0 the newly requested service is the only service that will exist in the NS created as a consequence of the *no overlapping*, and therefore, no pNode needs to be sent as parameter.

As mentioned before, after adding the service to all the possible nodes, it is necessary to update the consumption vector of the CAs affected by the update. In order to facilitate this task, when a possible path is found for the CA with the highest possible height, that CA and the other CAs previously identified in the Step 1 are added to a queue. That queue is later used to update in order, from the lowest height to the highest height the aggregated consumption vectors. The computation of the consumption vector of a given CA is described in Algorithm 3. As in the case of the *check* phase, the algorithm is executed per each node until the RST is empty.

5.4.2 REMOVAL OF A SERVICE

The second operation that the NSTree supports is the removal of a service reservation. This operation can affect multiple network snapshots, depending on the duration of the service, with a running time of $\mathcal{O}(n)$, being n the number of network snapshots. The procedure is described in

Algorithm 8 Add Service to a NS algorithm

```
1: function AddService(RST, aNode, pNode)
2:     CAstack.add(aNode)
3:     parent ← aNode.getParent()
4:     while parent.overlaps(RST) do                    ▷ Step 1
5:         CAstack.add(parent)
6:         parent ← parent.getParent()
7:     end while
8:     while node ← CAstack.peek() do
9:         path ← node.getOptimalPath()
10:        if path exists then
11:            propagate solution to descending NSs
12:            generateRelocationAlternatives()
13:            update CAs
14:        else
15:            CAstack.pop()
16:        end if
17:    end while
18:    returnlist of solved nodes
19: end function
```

Algorithm 9, which stops when a network snapshot is found that does not contain the service in its concurrent services list.

It needs to be taken into account that the removal of a service reservation can also affect the NSTree structure itself, resulting in the removal of not only NS nodes, but also CAs. As a consequence, every time an NS node is updated, it is necessary to check whether more concurrent services exist or not. If the service being removed is the only concurrent service in the analyzed NS node, this results in the deletion of that NS node.

On the other hand, it is also necessary to check whether consecutive NS nodes need to be merged or not. After the removal of a service, it is possible to have two consecutive NS nodes, the pNode and aNode, with the same concurrent services. We refer to this situation as pNode being soft equivalent to aNode. Meaning that the same concurrent services exist, but they do not have necessarily the same resources reserved. In that case, it is possible to remove aNode just by extending the duration of the pNode until the end time of the aNode.

Once the aNode has been removed or merged with the pNode, it should be checked whether the parent CA needs to be removed or not. More

precisely, when the parent CA is left with a single child, the CA node needs to be replaced by that child.

Finally, once the NSTree has been rearranged and the corresponding NS and CA nodes have been removed, it is necessary to progress the release of the network resources upwards. That is, the resources released at the aNode need to be released in the necessary parent CAs.

5.4.3 PROGRAMMATIC REMOVAL OF A NETWORK SNAPSHOT

The third operation supported by the NSTree is the programmatic removal of a network snapshot. This operation occurs when an active network snapshot reaches its end time, that is, when the network snapshot expires, and it is described in Algorithm 10. At that moment, it is necessary to remove the NS node representing the finalized network snapshot, and update any CA directly or indirectly aggregating that NS node. As in the case of a service removal, if by removing the NS node the CA is left with a single child, the CA is replaced by the child.

By definition, the expiration of a network snapshot necessarily involves the removal or the insertion of at least one service in the network. However, it is also possible to have services that remain unchanged when transitioning from a network snapshot to another. As a consequence, every time a network snapshot expires, it is necessary to perform the operations:

- Check which services need to be relocated onto alternative paths and trigger the necessary re-programming of the network devices involved.
- Remove the services that do not longer exist in the next network snapshot.
- Install the new services.

Algorithm 9 NSTree update for remove services algorithm

```
1: function NSTreeUpdateRemove(service, NSTree)
2:     aNode ← root.firstChild
3:     pNode ← root
4:     stop ← false
5:     while !stop do
6:         if service exists in aNode then
7:             RemoveService(service, aNode, pNode)
8:             if service does not exists in aNode.nextNode then
9:                 stop ← true
10:            end if
11:        end if
12:        pNode ← aNode
13:        aNode ← aNode.nextNode
14:    end while
15: end function
16: function RemoveService(service, aNode, pNode)
17:    parent ← aNode.getParent
18:    aNode.releaseResources()
19:    if aNode is empty then
20:        remove aNode
21:    else
22:        if aNode.softEqual(pNode) then
23:            aNode.merge(pNode)
24:        end if
25:        if aNode.softEqual(aNode.nextNode) then
26:            aNode.merge(aNode.nextNode)
27:        end if
28:    end if
29:    if parent has single child then
30:        fromCa2NS(parent)
31:    end if
32:    while parent is CA do
33:        update parent consumption vector
34:        parent ← parent.getParent
35:    end while
36: end function
```

Algorithm 10 Programmatic removal of a network snapshot algorithm

```
1: function ProgrammaticRemovalOfNS(NS)
2:     parent ← NS.getParent()
3:     if parent is root then
4:         if parent.getChildren().size > 1 then
5:             root.nextNode ← NS.nextNode
6:         else
7:             root.nextNode ← null
8:         end if
9:     else
10:        if parent.getChildren().size == 2 then
11:            fromCa2NS(parent)
12:        end if
13:    end if
14:    parent.removeChildren(NS)
15: end function
```

6 Optimization Techniques

> "Premature optimization is the root of all evil (or at least most of it) in programming."
>
> — Donald Knuth

Even if the path computation algorithm has been designed to avoid some resources to be under-utilized while others remain over-utilized, there are other optimization techniques that can be applied to improve the network resource consumption and increase the SAR. As such, this chapter presents two optimization techniques that exploit the flow-based nature of SDN technologies, namely *flow relocation* and *flow disaggregation*. First, the flow relocation technique leverages the high programmability of SDN to dynamically relocate services into alternative paths. Second, the flow disaggregation technique is used to split original services into more granular sub-services easier to relocate, taking advantage of the higher granularity available to forward the traffic at the OpenFlow devices.

6.1 FLOW RELOCATION

Usually, core networks present a high degree of redundancy and resource over-provisioning to deal with unexpected network failures in an efficient manner. As a result, it is assumed that multiple paths exist between the same pair of source and destination nodes. Furthermore, these paths may be equally good in terms of QoS metrics, like hop count (the constraint

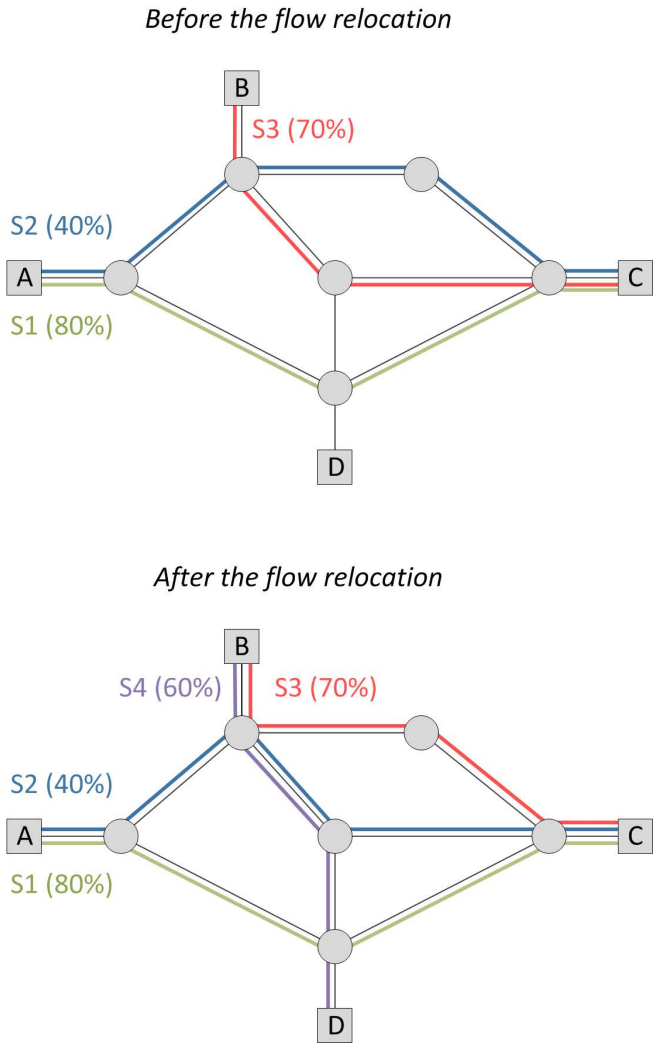


Figure 6.1.: Flow relocation.

that we use in our path computation algorithm), or other metrics such as latency, cost, etc. However, even if load balancing techniques are used to optimize the network resource consumption, the utilization of routing algorithms can lead to the rapid congestion of the optimal paths. Moreover, having different pairs of source and destination nodes may cause uneven network resource utilization. As a result, service requests may be rejected unnecessarily, therefore reducing the SAR of the system [70].

Algorithm 11 Flow relocation algorithm

```
 1: procedure FlowRelocation(service)
 2:     NS2BeUpdated ← []
 3:     while Check do
 4:         if NS has not resources then
 5:             relocationAlternatives ← NS.alternatives()
 6:             while relocationAlternatives is not empty do
 7:                 alternative ← relocationAlternative.peek()
 8:                 vector ← consumptionVector(alternative)
 9:                 if vector has resources for new service then
10:                     NS2BeUpdated.add(NS, vector, alternative)
11:                     break
12:                 else
13:                     relocationAlternatives.pop()
14:                 end if
15:             end while
16:         end if
17:         if Service can be accepted in all NSs then
18:             for all NS in NS2BeUpdated do
19:                 NS.consumptionVector ← NS2BeUpdated.get(NS).getVector()
20:                 NS.optimalPaths ← NS2BeUpdated.get(NS).getAlternative()
21:                 updateCA()
22:             end for
23:         end if
24:     end while
25: end procedure
```

In some cases, the rejection of a service reservation request may be a consequence of the order in which the prior services were requested. Since the algorithm always tries to provide the best paths, inevitably some resources become unavailable faster than others. Notwithstanding, a different distribution of the services across the network may solve this problem.

In such a scenario, the solution presented in this thesis includes a mechanism to relocate the flows into alternative paths, called flow relocation, which is described in Algorithm 11. This strategy is possible thanks to the utilization of network snapshots, which allow the association of the same service request to different paths through time.

The flow relocation algorithm leverages the fact that relocation alternatives are generated when a service is accepted in a NS. When during the check phase a service cannot be accepted into the NS with the

current arrangement of services, it takes the list of relocation alternatives and checks whether with any of them the acceptance of the service is possible. Since in order to accept the service all the affected network snapshots must be analyzed, the relocation alternative for a given NS is stored in a list, and it is only applied if the remaining NSs are also able to allocate the service.

Figure 6.1 illustrates a network with three already accepted services being provided through the shortest paths. When a new service demand is received, and given that the current service distribution does not allow the acceptance of this new service, the flows B and C are relocated onto alternative paths, making possible the acceptance of the new service D. It is worth noting that the flow relocation mechanism is embedded in the solution in the *Service Manager*, and it is automatically implemented when a service reservation demand is processed.

However, the utilization of the flow relocation mechanism entails some challenges. First, although it may be necessary, it is undesirable to move the services from one path to another. Even if OpenFlow provides the means to achieve this with a minimum packet loss thanks to the priority-based forwarding, which allows to install the alternative path with a higher priority prior to the removal of the previous one, it needs to be assumed that there will always be some losses. This is precisely why in Algorithm 8 the paths are first computed using the information from the CAs and then propagated towards the NSs. This way, the same path is valid in consecutive network snapshots, hence, reducing the flow relocation.

On the contrary, there are other situations in which the relocation of a service may be done without any packet loss. That is the case of services scheduled for the future that have not been installed in the network. In such a case, when all the paths for the newly requested service are not valid, a relocation alternative may be tested. This occurs first at the *check* phase, since the consumption vector always reflects the available resources for a given choice of paths. In case there is no room for the service, once a NS node without the necessary resources is found, the relocation of the services is tried, prior to the rejection of the service request. In addition, the relocation of backup path neither entails any

packet loss, since resources are reserved but the associated paths are not effectively installed in the network.

The flow relocation mechanism can also differentiate among different types of services. There are some scenarios, though, in which the flow relocation may only be applied to specific service types. For instance, the system may differentiate between low priority services such as best effort services which can be relocated and high priority services that due to specific constraints as high availability or minimum latency do not accept any flow relocation.

6.2 FLOW DISAGGREGATION

The second technique used in DynPaC to increase the SAR is the flow disaggregation. In a nutshell, flow disaggregation splits an original service into a set of smaller sub-services that are easier to relocate. Therefore, this technique is tightly coupled to the previous one. As an example of this technique, Figure 6.2 represents the case in which the service D is only accepted in the network once the service B is disaggregated into two smaller sub-flows easier to relocate.

Traffic splitting has always been used in TE. However, as pointed out in the Section 2 of Chapter I, the traditional approaches present some limitations related to unrealistic traffic splitting ratios and packet reordering. Taking into account those limitations, the flow disaggregation mechanism presented in this thesis takes into account the real-time traffic pattern information that is available through the REST API. Furthermore, it leverages the high granularity achievable at the OpenFlow forwarding devices to split the original service into a less disruptive and more realistic way.

The flow disaggregation mechanism, which is described in Algorithm 12, uses the interface towards the *traffic analyzer* to retrieve information about the traffic pattern of the services. As such, once a service is about to be discarded after the execution of the flow relocation mechanism, the service manager gets the information of the services currently provided in the network. This last point is important, since the flow relocation mechanism relies on the traffic pattern, it can only be applied to services actually running in the network.

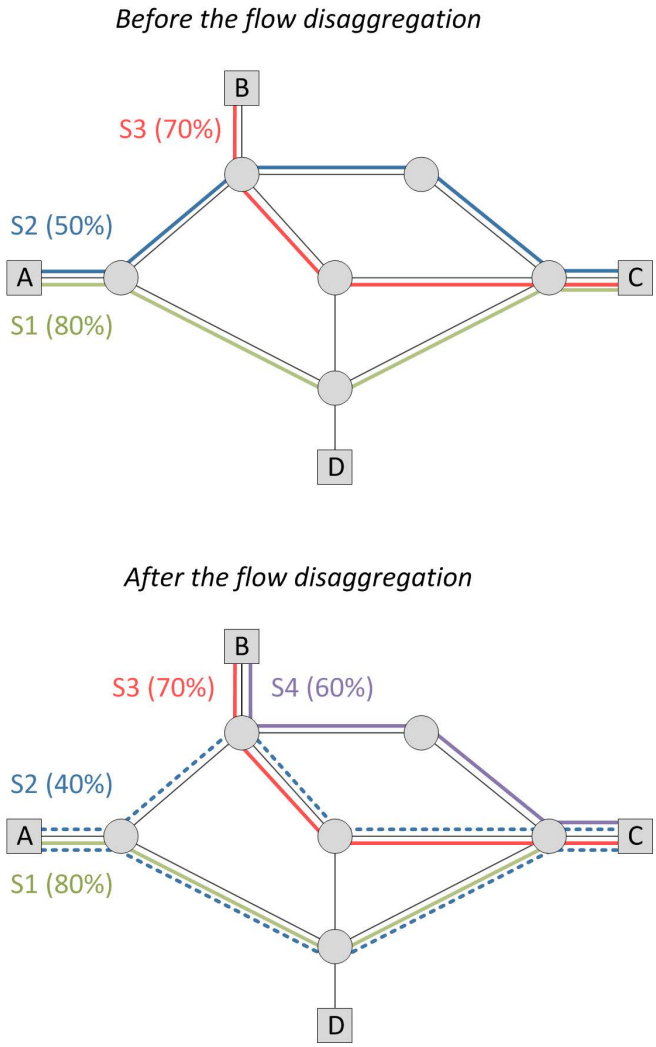


Figure 6.2.: Flow disaggregation.

Once it retrieves the list of services that can be disaggregated, the flow disaggregation mechanism removes from the consumption vector of the current NS the resources that are consumed by these services. Please note that this step does not remove the services from the network elements. Then, it checks whether the new service can be accepted in the current NS. If negative, the service is automatically rejected. If positive, the algorithm generates a set of disaggregation alternatives taking into account the set

of services that can be disaggregated and the disaggregation information provided by the traffic analyzer for each of them.

Then, it checks if any of the disaggregation alternatives can be accepted in the current NS. It is worth noting, that in order to consider a disaggregation alternative valid, all the sub-services associated with all the services contained in the disaggregation alternative must be accepted in the NS. Then, if a disaggregation alternative is found and the service can be accepted in the remaining NSs that it affects to, the current NS is updated with the new consumption vector and optimal paths, so as the affected CAs, and the new service is accepted.

Finally, once the new service has been accepted for a given disaggregation alternative, the network elements are updated to remove the original flow entries associated to the original services and install the new flow entries of the sub-services.

Algorithm 12 Flow Disaggregation Algorithm

```
1: procedure FlowDisaggregationAlgorithm(service)
2:     selectedAlternative ← null
3:     if currentNS has not resources then
4:         disaggServices ← TrafficAnalyzer.getServices()
5:         for all serviceD in disaggServices do
6:             currentNS.consumptionVector.remove(serviceD)
7:         end for
8:         if service can be accepted in currentNS then
9:             disaggregationAlternatives ← generateDAs(disaggServices)
10:            for alternative in disaggregationAlternatives do
11:                if alternative is valid then
12:                    selectedAlternative ← alternative
13:                    break
14:                end if
15:            end for
16:            if selectedAlternative is not null then
17:                if all remaining NSs can accept new service then
18:                    currentNS.loadAlternative(alternative)
19:                    Updatenetworkelements
20:                    accept new service
21:                end if
22:            end if
23:        end if
24:    else
25:        reject new service
26:    end if
27: end procedure
```

Part IV

VALIDATION

7 Functional Validation

> "No matter how slick the demo is in rehearsal, when you do it in front of a live audience, the probability of a flawless presentation is inversely proportional to the number of people watching, raised to the power of the amount of money involved."
>
> — Mark Gibbs

This chapter presents the functional validation conducted to establish that the DynPaC framework satisfies the objectives presented in Chapter I.

It is worth noting that the implementation of the DynPaC framework is part of the efforts conducted by the pan-European REN GÉANT and the homonym H2020 project to improve its current BoD service provisioning tool. As a consequence, in addition to a set of experiments demonstrated in international conferences, the tests conducted in a laboratory owned by GÉANT are described, which were used to validate the DynPaC framework against a set of requirements proposed by their network operations team.

7.1 SUITABILITY OF THE DYNPAC FRAMEWORK FOR TE AND OPTIMIZATION TECHNIQUES

The first demonstration described in this section took place in the European Workshop of Software-Defined Networks of 2015 held in Bilbao [221]. This section describes the objectives of the demonstration, the

scenario that was used and the description of the experiments that were performed.

OBJECTIVE

The main objective of this demonstration was to to prove that the DynPaC framework is able to support the provisioning of resilient L2 services with bandwidth guarantees. More precisely, the functional objectives of this demonstration are listed below:

- Demonstrate the capability of the framework to program the network devices at the specific times at which the service reservations must start or expire.
- Demonstrate the creation of different network snapshots depending on how the service reservations overlap with each other.
- Demonstrate the capability of the framework to install a backup path for the affected services already provisioned in the case of a link failure.
- Demonstrate the capability of the framework to relocate already installed services into alternative paths, in order to support the acceptance of a new service request that otherwise would be rejected.
- Demonstrate the capability of the framework to retrieve information from an external traffic analyzer and use that information to disaggregate an original service into multiple sub-services easier to relocate. As in the previous case, the goal is to accept a new service demand that without this mechanism would be rejected.

SCENARIO

This section describes the scenario that was used to demonstrate the functionalities supported by the DynPaC framework.

In order to implement the DynPaC framework, the ODP network operating system was selected, since it was the most powerful SDN

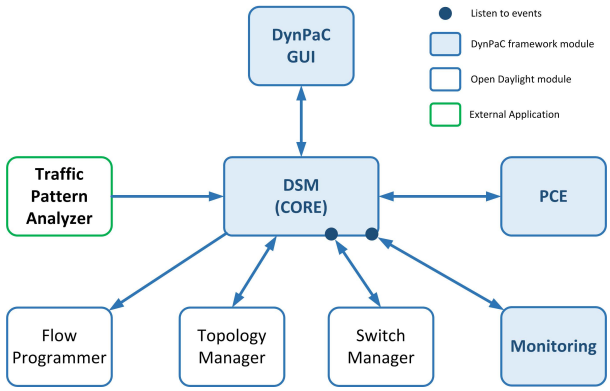


Figure 7.1.: Implementation of the DynPaC framework with ODP.

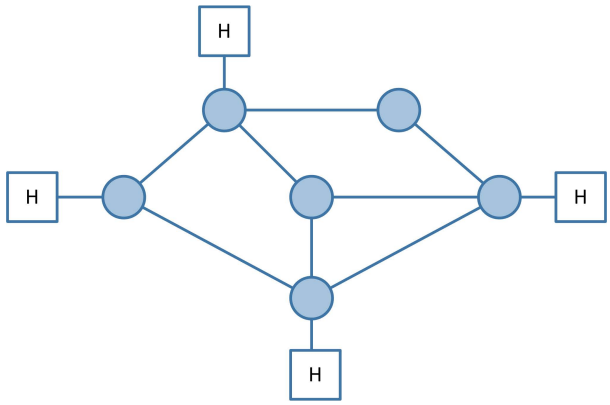


Figure 7.2.: Topology used for the EWSDN 2015 demonstration.

controller at that time. Figure 7.1 depicts the architecture of this first implementation of the DynPaC framework, which re-uses the Topology Manager, the Switch Manager and the Flow Rules Manager modules already included in the ODP Hydrogen release. In addition, ODP was extended with four custom modules and a REST API, where the logic of the advance reservation mechanism and the techniques to improve the network resource utilization were implemented.

Please note that in this first release of the DynPaC framework, the architecture does not correspond exactly with the one presented in Chapter IV. This is the consequence of a continuous refactoring process of the software, which has allowed to improve the architecture of the framework. In addition, the network snapshot database used in this version operated in linear time, and the PCE did not implement

Figure 7.3.: Video of the EWSDN 2015 demonstration.

the path pruning techniques described in Chapter V. However, this implementation represents the starting point of the DynPaC framework, and the algorithms presented in this thesis have been designed to increase the performance of the solution for its implementation on a real deployment (scheduled for the end of 2017).

The demonstration was run on an Ubuntu 14.04 virtual machine with 4 GB of RAM and two dedicated cores of an Intel CORE i7. The network was emulated using mininet, where the topology illustrated in Figure 7.2 comprised of four nodes and six OVS switches with OpenFlow 1.0 support was used. As mentioned before, the controller selected for this first release of the DynPaC framework was the Hydrogen release of ODP.

DESCRIPTION OF THE EXPERIMENTS

A video with the experiments conducted for this demonstration is available through the following QR code (see Figure 7.3). The three experiments that were conducted are described below:

- **Resilience:** this first experiment shows how the resilience mechanism works in order to guarantee the service provisioning even in the case of a link failure. It shows the primary path programmed into the network devices and how in case of failure, the flow entries associated to the primary path are deleted while the flow entries associated to the backup path are installed.

- **Advance reservations:** the second experiment deals with advance reservations. It demonstrates how the DynPaC framework generates the different network snapshots to represent the network state over time, each of which considers the different combinations of paths

valid for the concurrent services. This experiment consists of the reservation of a service that generates a first snapshot, and how the request of three different services from the same source to the same destination but with different start and end times generate three additional snapshots. This approach allows to reserve the same resources for the three additional services, since they do not overlap in time.

- **Flow disaggregation:** the third experiment shows how the disaggregation algorithm works. This experiment consists of the request of three services to the DynPaC framework that make impossible the provisioning of a fourth service keeping the current network condition. When the service manager detects that there are no available resources to accept the service request, it uses the flow disaggregation algorithm to find a set of sub-services in which an already installed service can be divided to ease its relocation, freeing enough resources in the network to accept the new service.

The three experiments were successful, and demonstrated the feasibility of the DynPaC framework to support TE in SDN, and the viability of utilizing the flow relocation and flow disaggregation techniques for network resource optimization.

7.2 MULTI-DOMAIN E2E INVOLVING OPENFLOW DOMAINS

As mentioned in Section IV, DynPaC includes a REST interface that allows its interconnection with external elements, such as orchestrators and multi-domain agents. The demonstration that is described here took place at the Supercomputing Conference of 2014 [222].

OBJECTIVE

As presented in [223], the connection-oriented multi-domain OpenFlow is based on the utilization of an OpenFlow controller as the NRM. This approach is consistent with the foundations of the NSF, since the NSI-CS service remains technology agnostic, whereas the management of the intra-

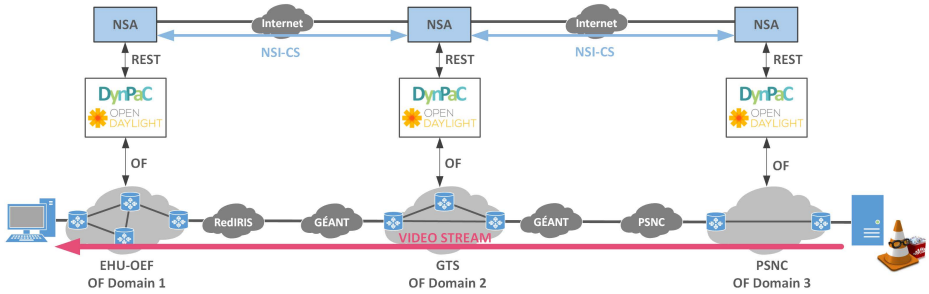


Figure 7.4.: Demo scenario for multi-domain OpenFlow based on NSI CONTEST and DynPaC results.

domain topology and the setup of the paths inside the domain rely on the OpenFlow controller. Furthermore, the NSI-CS protocol is flexible enough to express services in terms of OpenFlow matching fields by simply adding information to the Service Termination Point (STP) description. It is worth remarking that the implementation of the connection-oriented multi-domain OpenFlow was possible thanks to the work carried out by two GÉANT Open Call projects: NSI CONTEST and DynPaC.

In such a context, the main goal of this demonstration was to prove the suitability of the DynPaC framework as NRM for OpenFlow domains.

SCENARIO

In order to prove the feasibility of the proposal, the following setup was implemented and demonstrated at the Supercomputing 2014 conference held in New Orleans. The demo scenario consisted of three domains interconnected using RedIRIS, GÉANT and PIONIER [224] networks.

As depicted in Figure 7.4, one of the domains was located at the University of the Basque Country (UPV/EHU), formed by 4 NEC IP8800 switches with OpenFlow 1.0. The second domain was a testbed created via the GÉANT Testbed Services (GTS) [225], where three OVSs were instantiated in virtual machines provisioned in Amsterdam, Bratislava and Copenhagen. Finally, a third domain was deployed at PSNC, the Polish NREN, formed by 2 Juniper MX80 switches. Each of the domains was controlled by a DynPaC controller acting as the NRM of that domain. Finally, a client host was attached to the UPV/EHU domain and a server host to the PSNC domain.

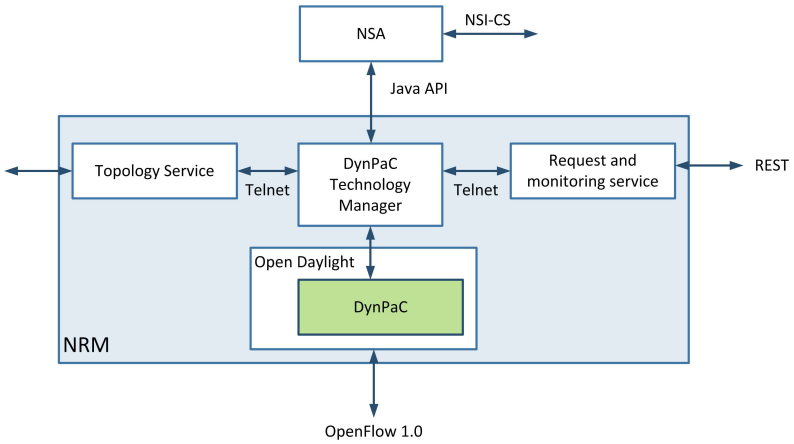


Figure 7.5.: Modules of a NRM for OpenFlow domains based on DynPaC.

Figure 7.5 describes all the involved modules and interfaces for the establishment of multi-domain L2 connectivity services. On the one hand, the NRM provides the discovery and provisioning functions to the NSA. Moreover, through the Topology Service module, it exchanges information about the available STPs with the rest of the Network Services Interface (NSI) domains. It also provides a REST interface for the integration with the GUI. Finally, it uses the same version of the DynPaC framework that was described in the previous section for the provisioning of the L2 connectivity circuits in each of the OpenFlow domains.

On the other hand, the NRM is comprised of the DynPaC Technology Manager that interconnects with the NSA using Java APIs. It uses DynPaC for network resource discovery and local configuration to identify the STPs of each domain. Then, it shares the NSI domain's topology with other domains and gets the actual global NSI topology. It also redirects the client requests during the E2E connection, generates the valid NSI XML request messages and uses the NSI-CS protocol to make the multi-domain reservations. As mentioned before, the DynPaC Technology Manager in each domain gets the local reservation requests and triggers the service provisioning using the DynPaC REST interface. Finally, it also monitors the status of the reservation request.

Figure 7.6.: Video of the Supercomputing 2014 demonstration.

DESCRIPTION OF THE EXPERIMENT

During the demonstration, a user requested a service using the GUI, specifying the source and destination nodes, and the start and end times. With that information, a script was launched that requested a VLAN service from the client located in UPV/EHU to the server located in PSNC. The reservation was requested first to the NSA located at PSNC, available through a public IP address. After that initial request, the NSA of PSNC made the same request to the GTS's NSA, request that finally reached the NSA of the UPV/EHU domain. With this procedure, it was checked whether it was possible to deploy the requested service in all the involved domains from the given source to the given destination or not.

Once the service request was accepted by all the domains, each NSA communicated with the DynPaC framework of its corresponding domain to request a path between the STPs of that domain, in order to provide the requested E2E service.

At the start time of the service reservation, the DynPaC framework automatically installed the necessary flows in the flow tables of all the involved nodes within its domain. Once the path was established from the client to the server, the establishment of the multi-domain L2 connectivity circuit was demonstrated by streaming a video from a VLC server located at the PSNC server. A video of this demonstration is available through the following QR code (see Figure 7.6).

7.3 MULTI-DOMAIN E2E INVOLVING HETEROGENEOUS DOMAINS

Currently, the pan-European REN GÉANT offers the BoD service through the AutoBAHN provisioning tool [24], which allows to request short term, high capacity and E2E connectivity services through a web interface. Furthermore, AutoBAHN is compliant with the NSI-CS protocol, which makes possible the establishment of multi-domain connections. However, the AutoBAHN provisioning tool is not currently prepared to operate over software-defined networks.

In such a scenario, the Joint Research Activity of the GÉANT 4 H2020 project focused on *Future Network Services* has implemented a solution based on the DynPaC framework to provide multi-domain BoD services using SDN technologies that is compliant with the NSI-CS protocol. This approach allows to clear the pace for a migration towards SDN technologies that will allow to improve TE and QoS, while the strategic connections with other RENs are kept intact thanks to the NSI-CS support.

The SDN-based BoD service provisioning was demonstrated at the IEEE Conference of Network Softwardization 2016, which was held in Seoul. It is worth mentioning, that this demonstration was awarded with the *Best Demo Award* of the conference. In addition, this demonstration was also presented in the GÉANT Symposium 2016, and in the Terena Networking Conference 2016. As in the previous cases, this section presents the objective of the demonstration, the scenario that was used and the description of the demonstration.

OBJECTIVE

All in all, the main objective of this demo is to show how the proposed solution is able to provide multi-domain BoD services involving heterogeneous transport technologies, how the QoS constraints such as rate limiting are enforced at the SDN domains and how network resilience is supported satisfying the required QoS constraints.

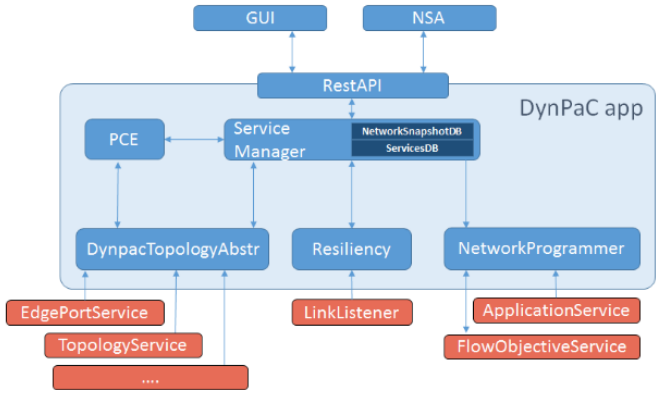


Figure 7.7.: Implementation of the DynPaC framework with ONOS.

SCENARIO

As mentioned before, in multi-domain scenarios, the DynPaC framework is used as the NRM of OpenFlow domains. In the NSF, the NRM is the element in charge of the intra-domain path computation, whereas the NSA controls the multi-domain connection through the NSI-CS protocol. The NSAs are also in charge of the topology information exchange, which in this case is achieved by connecting each NSA to a central repository where the intra-domain topology information is stored.

It is worth mentioning that, as described in [226], the DynPaC framework went through a refactoring process after the demonstration presented in the section before. The second release of the framework, which was the one used for this demonstration, has been implemented as an application running at the ONOS controller, which provides a basic set of services and features, such as device, link or host discovery. The details of the implementation used for this demonstration are depicted in Figure 7.7, where the red modules are ONOS modules and the blue modules are custom DynPaC modules. The set of algorithms and data structures utilized in this second release are the same ones that were used in the ODP implementation.

As depicted in Figure 7.8, the scenario used for the demonstration consists of two OpenFlow-enabled domains interconnected through the GÉANT legacy BoD domain, which is an MPLS domain. The first OpenFlow domain is located at the GÉANT's Cambridge premises and it

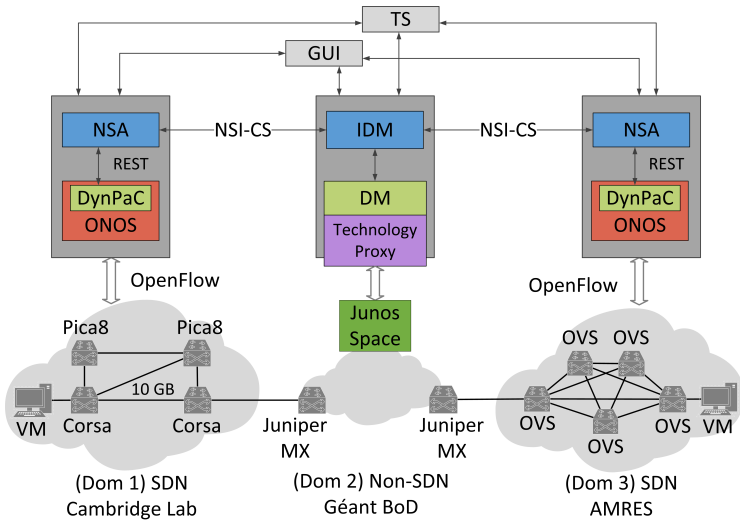


Figure 7.8.: Scenario for multi-domain BoD service provisioning involving three heterogeneous domains.

consists of two Corsa whiteboxes with rate limiting capabilities and two Pica 8 virtual switch instances. The second OpenFlow domain is located at the AMRES premises, the Serbian NREN, and it consists of five OVSs interconnected to each other using a full mesh network. All the OpenFlow devices used in both domains are OpenFlow 1.3 capable.

DESCRIPTION OF THE DEMONSTRATION

This demonstration consisted of the request of two BoD services from a client located at the Cambridge premises to a server located at the AMRES premises. The services were requested using the GÉANT BoD portal, where the following information was provided:

- Source domain.
- Source STP.
- Destination domain.
- Destination STP.
- Start time.

- End time.
- Ingress VLAN.
- Egress VLAN.
- Bandwidth.
- Resilience mechanism (Gold or Regular).

In order to demonstrate the resilience capabilities of the DynPaC framework, the following two services were requested.

- *Gold service:* a 50 Mbps service with a guaranteed backup path.
- *Regular service:* a 20 Mbps service without a guaranteed backup path.

Once the services' start time reached, the elements involved in the three domains were successfully programmed, and the services appeared as provisioned in the BoD portal. At that moment, it was possible to inject traffic in both services using *iperf*, and to check with *speedometer* how the rate limiting was successfully enforced at the Cambridge domain's Corsa whiteboxes.

Finally, in order to demonstrate the resilience capabilities of the DynPaC framework, the link interconnecting the Corsa whiteboxes was disabled, to force the installation of the backup path of the Gold service. At that moment, it was possible to see that the OpenFlow switches at the Cambridge domain were re-programmed and how the traffic exchanged using the Gold service continued without any disruption, involving in this case the Pica 8 switches, whereas the traffic exchanged using the Regular service stopped. Please note that although it is possible to have Regular services with restoration capabilities, it was decided not to compute any backup path to demonstrate that resilience is a feature that can be handled on a per service basis.

A video of the demonstration is available through the following QR code (see Figure 7.9).

Figure 7.9.: Video of the NetSOFT 2016 demonstration.

7.4 GÉANT REQUIREMENTS TESTS

As mentioned before, the development of the DynPaC framework has been included as one of the use cases in the GÉANT 4 project, in order to use it as the NRM of OpenFlow domains in the future SDN-based BoD service [226]. As such, this section presents the requirements that the framework should satisfy, the scenario that was used to conduct the tests and the obtained results.

7.4.1 REQUIREMENTS

The first step in the deployment of the future SDN-based BoD service consisted of the specification of the functional requirements that the SDN application acting as an NRM should satisfy. This process involved GÉANT network operators, users and BoD, SDN and NSI-CS specialists, and the output is summarized in Table 7.1.

First, the automated network *topology discovery* was appointed as a must have feature. Currently, AutoBAHN lacks this functionality; the topology discovery is performed manually, which can be error prone. Second, the *identification of the STPs* was also designated as a must have feature. As mentioned before, in NSI-CS, the STPs represent the points that the users can use to start or terminate a service. They consist of a port, and optionally, of the range of available VLANs in that port. The identification of the STPs is absolutely necessary because they are the only network resources visible from the NSAs. Therefore, the BoD solution must be able to identify the edge nodes of the network domain, and only expose to the NSA the ports and VLANs that the users can use in each of them. In this regard, the SDN application must also provide

Table 7.1.: Requirements for an SDN-based NRM.

Requirement	Description
Topology discovery	The NRM must be able to discover the intra-domain network topology automatically.
STP identification	The NRM must be able to identify the STPs of the intra-domain topology, as a set of of ports and the associate VLAN range.
Network updates	The NRM must be able to detect changes in the intra-domain topology and react upon those changes to provide resilience.
L2 VLAN circuit provisioning	The NRM must be able to install an end-to-end L2 VLAN circuit across the nodes that conform the path, since it is the type of circuit provided by most BoD provisioning tools.
VLAN Translation	The NRM must be able to support VLAN translation in order to optimize the resources utilization.
Stateful PCE	The NRM must be able to compute the paths taking into account the state of the network and the available resources (VLANs and bandwidth).
Time-dependent TED	The NRM must keep track of how the resources' consumption evolves over time in order to support the scheduling functionality.
Advance reservations	The NRM must support advance service reservations in order to optimize the resources utilization; therefore, the associated circuits must only be provisioned from the service start time to the service end time.
Rate limiting	The NRM must be able to add, modify and delete per-flow meters in the network devices in order to enforce the SLA subscribed between Géant and the user in terms of maximum bandwidth.
Interface towards an NSA	The NRM must have a well-defined interface to communicate with the NSA. This interface must support the advertisement of STPs (port and available VLAN range) and provide the means to check, add, remove, modify, provision and release a service reservation.

an *interface towards the NSA*, to expose the aforementioned STPs, to support the request of new service demands from the NSA and to permit the NSA to handle the life-cycle of the services.

It was also agreed that the solution must also be able to detect *network updates*. Since one of the limitations that must be solved in AutoBAHN is the lack of failure recovery capabilities, the SDN-based BoD solution must be able to automatically detect network changes. This feature is necessary to guarantee that in case of failure, the SDN application implementing the NRM for the SDN domain will be able to keep provisioning the services through alternative paths. Furthermore, the SDN application requires updated topological information in order to compute the optimal paths and guarantee the SLAs subscribed with the users.

In the provisioning of the BoD service, the SLA subscribed with the user requires the SDN application to have an up-to-date knowledge of the network resources consumed over time. BoD services are reserved for a certain period of time, to be delivered at the present moment or in the future. Therefore, the solution must support *advance reservations* and the automatic programming of the network devices at the specified times.

As a consequence, this requires an *stateful PCE* with a *time-dependent TED* to keep track of the resources consumed or reserved by the scheduled services. This would allow the computation of the optimal paths for the new service demands, taking into account the available bandwidth and VLANs in the network.

In addition, the BoD service must ensure that the user is able to obtain the requested bandwidth at all times. As a consequence, the network devices must be able to limit the bandwidth associated to each flow independently, to enforce the QoS subscribed between GÉANT and the users. The network devices must support *rate limiting*, and the NRM must have the capability to add, delete and modify the meters.

Finally, the multi-domain BoD service provisioning requires *L2 VLAN circuits* in each of the domains involved. Moreover, as supported in AutoBAHN, the SDN-based solution must be able to handle *VLAN translation*. This feature not only eases the interconnection between domains, as different VLANs can be use in each domain, but it also improves the network resources utilization.

7.4.2 SCENARIO

The scenario used to conduct the functional validation was deployed at GÉANT's premises, and it is illustrated in Figure 7.10. The list below summarizes the equipment that was used:

- **Corsa whiteboxes:** Two Corsa 6400 whiteboxes with rate limiting capabilities. Corsa is a novel SDN equipment manufacturer that uses customized FPGAs to construct pure OpenFlow datapaths with OpenFlow 1.3 support. The Corsa whiteboxes support a wide range of pipelines, that are delivered to satisfy the needs of specific use cases. As such, Corsa provided a custom pipeline to support the BoD use case, including support for VLAN translation and rate limiting.

- **Pica8:** In order to increase the redundancy of the network, two virtual instances of a Pica8 were included in the deployment. The Pica8 virtual switches support OpenFlow 1.3, but do not have rate limiting capabilities.

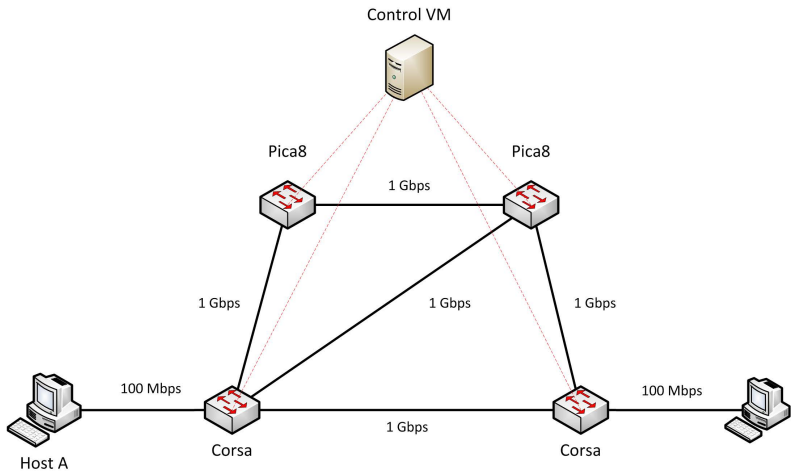


Figure 7.10.: Cambridge laboratory deployment for functional validation.

- **Host VMs:** Two virtual hosts with 1 GB of RAM were created in order to inject and receive traffic. The two hosts were attached to the Corsa whiteboxes, since they are the only devices in the network with rate limiting capabilities.
- **Controller VM:** A virtual machine with 4 GB of RAM was deployed in order to run the ONOS controller, the DynPaC application and the NSA of the domain.

7.4.3 RESULTS

This section summarizes the results of the experiments conducted in the Cambridge laboratory in order to test the suitability of the DynPaC framework to act as the NRM of the future SDN-based BoD service. On the one hand, Table 7.2 summarizes the test results, marked with ✓ if the test was successful and marked with ✗ if the test failed, and identifies which of the requirements proposed by GÉANT it validates. On the other hand, the remaining tables of this section present the procedures used to conduct the tests and the obtained result.

Table 7.2.: Summary of the functional validation tests.

Test No.	Topology Discovery	STP Identification	Interface to NSA	Network Updates	Advance Reservation	Stateful PCE	Time TED	Rate Limiting	VLAN circuit	VLAN translation
1	✓									
2					✓	✓	✓		✓	
3					✓	✓	✓		✓	✓
4					✓	✓	✓	✓	✓	
5				✓						
6			✓							
7		✓								

Table 7.3.: Test 1 - DynPaC topology abstraction.

DynPaC topology abstraction	
Procedure	1. Install DynPaC and check that the topology is retrieved at start time and that the graph is abstracted correctly.
Input	Number of hosts, devices and links and how they are interconnected.
Pass / Fail Criteria	Pass criteria: The topology is abstracted correctly and the maximum consumption vector is computed correctly. Fail criteria: (1) The topology is not retrieved. (2) The topology is retrieved but the maximum consumption vector is not correctly computed.
Expected output	1. Check at ONOS that all the devices are detected. 2. Check at ONOS that all the hosts are detected. 3. Check at ONOS that all the links are detected. 4. Activate the DynPaC application. 5. Check that DynPaC abstracts the topology correctly and that the maximum consumption vector represents the maximum available bandwidth in each undirected link.
Observations	No remarkable observations
Status	Success

Table 7.4.: Test 2 - Programmatic VLAN circuit establishment.

Programmatic VLAN circuit establishment	
Procedure	1. Reserve Service A with VLAN 10 from startTime to endTime. 2. Check at the ONOS controller that the flow entries have been installed in the flow tables of the Corsa whiteboxes at startTime. 3. Generate tagged traffic and check that the traffic is correctly received. 4. Check at the ONOS controller that the flow entries have been removed from the flow tables of the Corsa whiteboxes at the endTime. 5. Check that the traffic is no longer received at the destination node.
Input	Ports and switches where the two hosts are connected.
Pass / Fail Criteria	Pass criteria: The tagged traffic is received at the destination node only from startTime to endTime. Fail criteria: (1) The tagged traffic is not received at the destination node from startTime to endTime. (2) The traffic is received before the startTime. (3) The traffic is received after the endTime.
Expected output	1. Check at ONOS that the flow tables of the Corsa whiteboxes are empty before the startTime. 2. Check at ONOS that the flow entries for the VLAN 10 have been installed in the Corsa whiteboxes at startTime. 3. Generate tagged traffic using iperf at the source node. 4. Check at the destination node that the tagged traffic is received correctly. 5. Check at ONOS that the flow entries for the VLAN 10 have been removed from the Corsa whiteboxes at endTime. 6. Check at the destination node that the tagged traffic is no longer received.
Observations	No remarkable observations.
Status	Success

Table 7.5.: Test 3 - Programmatic VLAN circuit establishment with VLAN translation.

Programmatic VLAN circuit establishment with VLAN translation	
Procedure	1. Reserve Service A with an ingress VLAN 10 and egress VLAN 20 from startTime to endTime. 2. Check at the ONOS controller that the flow entries have been installed in the flow tables of the Corsa whiteboxes at startTime. 3. Generate traffic with the VLAN 10 and check that the traffic is correctly received with the VLAN 20. 4. Check at the ONOS controller that the flow entries have been removed from the flow tables of the Corsa whiteboxes at the endTime. 5. Check that the traffic is no longer received at the destination node.
Input	Ports and switches where the two hosts are connected.
Pass / Fail Criteria	Pass criteria: The traffic is received at the destination node only from startTime to endTime and the VLAN tag has been changed. Fail criteria: (1) The tagged traffic is not received. (2) The traffic is received before the startTime or after the endTime. (4) The traffic is received with the VLAN 10.
Expected output	1. Check at ONOS that the flow tables of the Corsa whiteboxes are empty before the startTime. 2. Check at ONOS that the flow entries for the VLAN 10 have been installed in the Corsa whiteboxes at startTime and an action to translate the VLAN is included. 3. Generate tagged traffic using iperf at the source node with the VLAN 10. 4. Check at the destination node that the tagged traffic is received correctly with the VLAN 20. 5. Check at ONOS that the flow entries for the VLAN 10 have been removed from the Corsa whiteboxes at endTime. 6. Check at the destination node that the tagged traffic is no longer received.
Observations	No remarkable observations.
Status	Success

Table 7.6.: Test 4 - Rate limiting.

Rate limiting	
Procedure	1. Reserve Service A with 10 Mbps of peak bandwidth with VLAN 10. 2. Reserve Service B with 20 Mbps of peak bandwidth with VLAN 20. 3. Check at the ONOS controller that the flow entries have been installed in the flow tables of the Corsa whiteboxes. 4. Check at the Corsa whiteboxes the created meters. 5. Generate traffic for both VLANs and check that it is limited at the Corsa whiteboxes.
Input	Ports and switches where the two hosts are connected.
Pass / Fail Criteria	Pass criteria: The traffic of each flow is correctly limited. Fail criteria: The traffic of each flow is not correctly limited.
Expected output	1. Check at ONOS if there is a flow entry for the VLAN 10 whose output is a meter. 2. Check at ONOS if there is a flow entry for the VLAN 20 whose output is a meter. 3. Check at the Corsa whiteboxes that the referenced meter is the correct one. 4. Inject traffic for each VLAN using two iperf clients. 5. Check at the destination host that the traffic of each VLAN is limited to the specified values.
Observations	In order to be able to operate with meters at the Corsa devices it has been needed to implement a driver at the ONOS controller specific to the pipeline utilized in the use case. This driver was developed by Corsa with the input of the GN4-1 project members.
Status	Success

Table 7.7.: Test 5 - Port status notification.

Port status notification	
Procedure	1. Tear down one of the ports of the Corsa whitebox. 2. Check if the controller receives a notification about the new port status. 3. Restore the port at the Corsa whitebox. 4. Check if the controller receives a notification about the new port status.
Input	None.
Pass / Fail Criteria	Pass criteria: The ONOS controller is notified about the new port status. Fail criteria: The ONOS controller is not notified about the new port status.
Expected output	1. Check if after tearing down the port the switch sends the OFPT_PORT_STATUS message. 2. Check if ONOS receives correctly the port status notification. 3. Check if after restoring the port the switch sends the OFPT_PORT_STATUS message. 4. Check if ONOS receives correctly the port status notification.
Observations	No remarkable observations.
Status	Success

Table 7.8.: Test 6 - Interface towards NSA.

Interface towards NSA	
Procedure	1. Request a services using the REST interface. 2. Check if the service has been requested correctly.
Input	API calls of the REST interface.
Pass / Fail Criteria	Pass criteria: The service is requested correctly. Fail criteria: The service is not requested correctly.
Expected output	1. Use the REST API to request a new service. 2. Check at the ONOS log if the request has been received correctly..
Observations	No remarkable observations
Status	Success

Table 7.9.: Test 7 - STP discovery.

STP discovery	
Procedure	1. Start ONOS and install the DynPaC application. 2. Check if DynPaC identifies correctly the set of STPs.
Input	Ingress and egress nodes and ports in the topology.
Pass / Fail Criteria	Pass criteria: The STPs are correctly identified by DynPaC. Fail criteria: The STPs are not correctly identified by DynPaC.
Expected output	1. Start ONOS and activate the DynPaC application. 2. Check using the DynPaC CLI if the STPs are correctly identified.
Observations	No remarkable observations
Status	Success

7.4.4 CONCLUSIONS

In summary, it can be concluded that the DynPaC framework satisfies the requirements identified by the GÉANT's network operations teams for the future SDN-based BoD service with multi-domain capabilities. Through a set of demonstrators in international conferences, it has been proven that the framework is able to provide resilient L2 services where the maximum reserved bandwidth is enforced through meters at the OpenFlow devices. The framework can operate as a stand-alone application for a single domain, or act as a NRM in multi-domain scenarios thanks to the REST API that it exposes, which can be further used by other kinds of orchestrators [227]. In addition, the framework has been tested in a laboratory with real OpenFlow devices, in order to check if the solution would work in the future SDN-based BoD deployment.

8 Performance Evaluation

> "I am putting myself to the fullest possible use, which is all I think that any conscious entity can ever hope to do."
>
> — HAL 9000 (2001: A space odissey)

This chapter presents the performance evaluation conducted to validate the suitability of the DynPaC framework and the data structures and algorithms it relies on to handle advance reservations. As such, a set of experiments have been designed to evaluate the performance of the solution in terms of the time required to accept or reject a new service reservation request (t_{setup}) and the SAR, since these are the most common parameters used to evaluate advance reservation systems.

First, the impact of the number of alternative paths on the network resource utilization that is achieved and the setup time are evaluated using different topologies, which has allowed to study the suitability of the path computation strategy. Second, the impact of the number of links in the network has been measured, to study its impact on the setup time and check whether the topology abstraction used in DynPaC is efficient or not to handle advance reservations. In addition, an analysis of the effect of the requested peak bandwidth has been conducted, to evaluate the effectiveness of the check and update algorithms at the time of accepting a new service reservation depending on its size. Finally, the effect that the aggregation of multiple NSs into CAs has in the admission control process has been analyzed, where multiple values of δ have been used and

the setup time and the SAR have been measured in order to state the suitability of the node aggregation policy.

8.1 ANALYSIS OF THE IMPACT OF THE NUMBER OF ALTERNATIVE PATHS IN THE SETUP TIME AND THE RESOURCE UTILIZATION

As described in Chapter V, the path computation is handled by means of two different algorithms. Firstly, the path pre-computation algorithm generates a set of alternative paths. Secondly, the on-demand path computation algorithm randomizes the set of alternative paths taking into account their hop number in order to enhance the network resource utilization. With such a strategy, the first experiment aims to state the impact of the number of alternative paths generated on the path pre-computation phase on the achieved network resource utilization, so as in the admission control process.

For this experiment, five different topologies have been used: four full mesh topologies of 5, 6, 7 and 8 nodes respectively and the topology of the ESNet network, which consists of 22 nodes and 66 links. In each topology, a set of concurrent services for the same period of time have been requested. With this approach the analysis gets simplified, since a single network snapshot is created where all the requested services coexist with each other.

It must be taken into account that for each topology the number of concurrent services has been selected to ensure the over-subscription of the overall network capacity. This way, it has been possible to study whether the number of alternative paths has an impact or not in the overall network resource utilization.

In order to conduct the tests, an Ubuntu 14.04 virtual machine with 6 GB of RAM and an Intel CORE i7 vPro processor has been used. The DynPaC framework has been implemented as an application running in ONOS 1.7 and the network has been emulated using mininet, where the switches are OVSs with OpenFlow 1.3 support. As a consequence, unlike when simulation tools are utilized, the experiments may have been affected by several factors of the deployment setup, such as the performance of the machine that has been used to run the experiments,

the utilization of a virtual machine, both the host and the guest operating systems or the performance of the ONOS controller, where the DynPaC application runs concurrently with multiple applications. In such an environment, although the obtained results are consistent, there are several values that excel when compared to the remaining values, which according to the author of this thesis, is a consequence of the setup that has been used.

8.1.1 FULL MESH NETWORKS

The first experiment has been conducted over a 5 node full mesh network, where 200 concurrent services with random source and destination nodes to ensure an even utilization of the network resources, and with a peak bandwidth ranging from 300 Mbps to 1 Gbps on steps of 50 Mbps have been requested. Each test has been repeated 30 times for four different values of n_{MAX}, which limits the number of hops that the generated alternative paths can have.

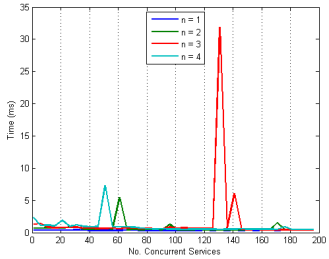
The obtained results are depicted in Figure 8.1, where the following information is presented: (1) the median values of the on-demand path computation time ($t_{ondemand}$) (see Figure 8.1a), (2) the check time (t_{check}) (see Figure 8.1b), (3) the update time (T_{update}) (see Figure 8.1c), (4) the overall setup time (t_{setup}), computed as the sum of the on-demand path computation time the check time and the update time, as expressed in equation 8.1 (see Figure 8.1d), (5) the number of rejected services (see Figure 8.1e), (6) the SAR (see Figure 8.1f), (7) the overall available bandwidth (see Figure 8.1g) and (8) the probability of rejecting a given service (see Figure 8.1h) depending on the number of concurrent services.

$$t_{setup} = t_{ondemand} + t_{check} + t_{update} \tag{8.1}$$

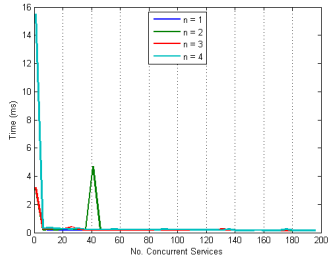
The same experiment has been repeated for the remaining full mesh topologies. On the one hand, for the 6 node full mesh network, depicted in Figure 8.2, 350 concurrent services have been requested, while for the 7 node full mesh network, depicted in Figure 8.3, 400 concurrent and for the 8 node full mesh network 600 concurrent services have been requested

respectively. In all the cases a peak bandwidth ranging from 300 Mbps to 1000 Gbps on steps of 50 Mbps have been used[1].

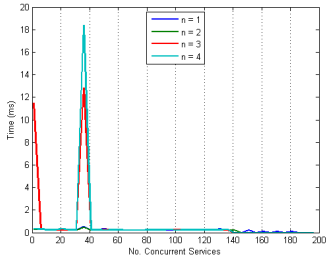
1 Please note that the time required to accept or reject the first requested service is higher since the NSTree has to be created.



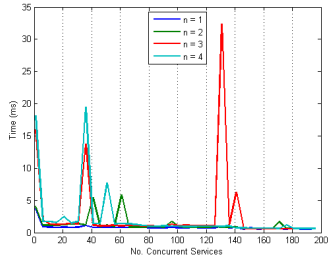
(a) On-demand path computation time.



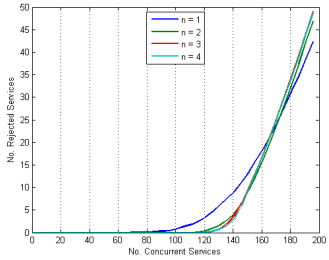
(b) Check time.



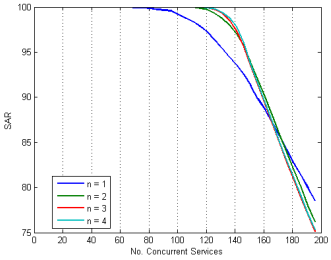
(c) Update time.



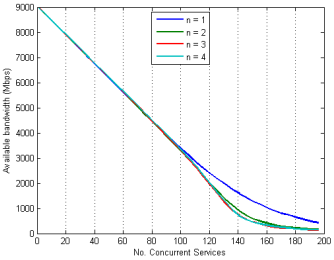
(d) Setup time.



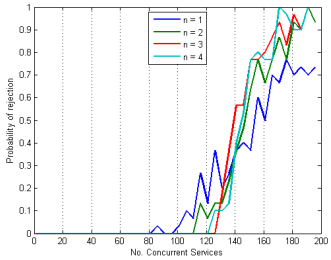
(e) No. Rejected services.



(f) Service Acceptance Ratio.

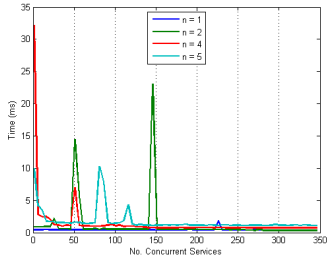


(g) Overall available bandwidth.

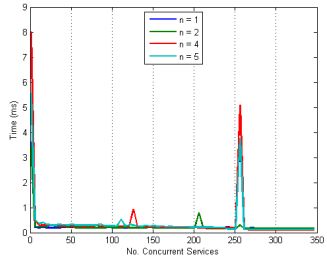


(h) Prob. service rejection.

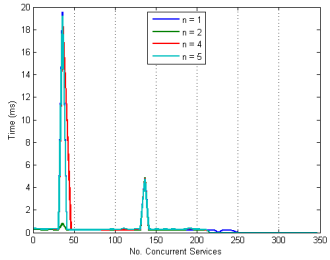
Figure 8.1.: Effect of the path diversity in a 5 node full mesh network.



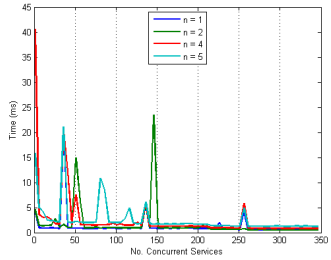
(a) On-demand path computation time.



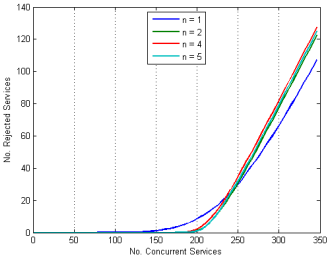
(b) Check time.



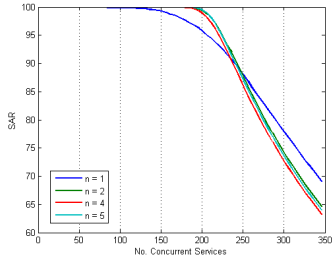
(c) Update time.



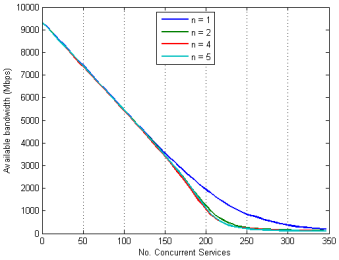
(d) Setup time.



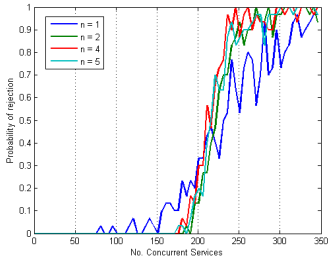
(e) No. Rejected services.



(f) Service Acceptance Ratio.

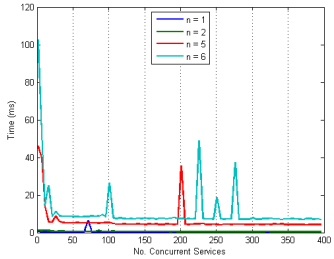


(g) Overall available bandwidth.

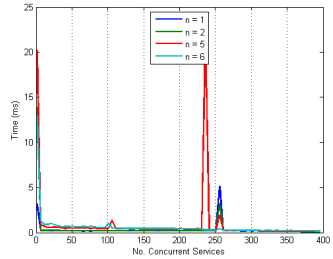


(h) Prob. service rejection.

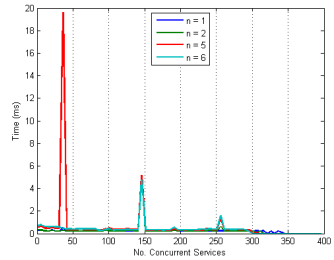
Figure 8.2.: Effect of the path diversity in a 6 node full mesh network.



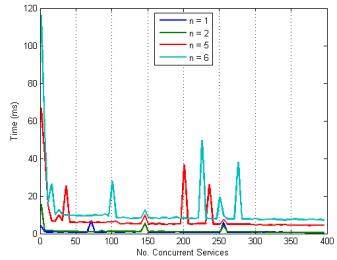
(a) On-demand path computation time.



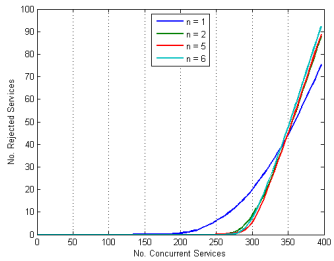
(b) Check time.



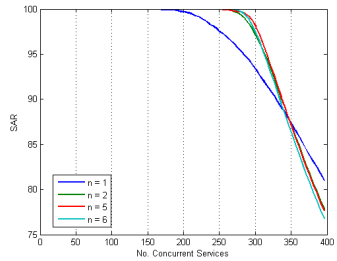
(c) Update time.



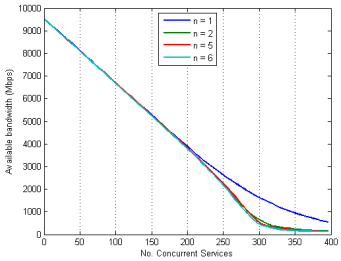
(d) Setup time.



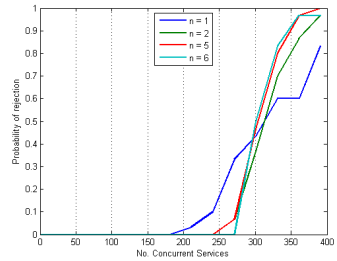
(e) No. Rejected services.



(f) Service Acceptance Ratio.

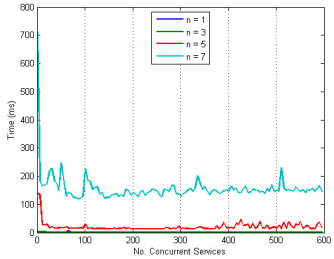


(g) Overall available bandwidth.

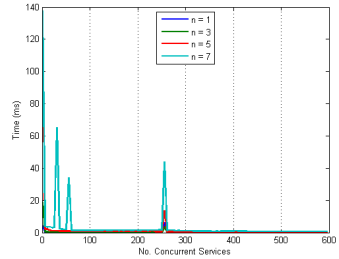


(h) Prob. service rejection.

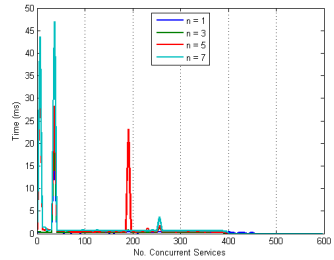
Figure 8.3.: Effect of the path diversity in a 7 node full mesh network.



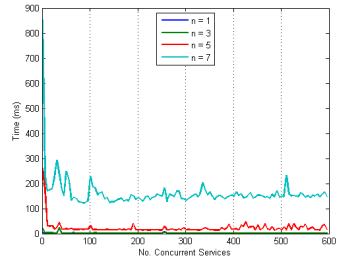
(a) On-demand path computation time.



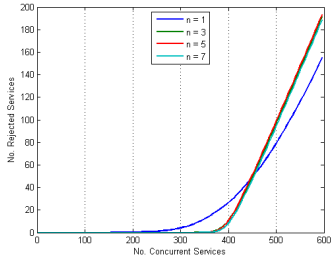
(b) Check time.



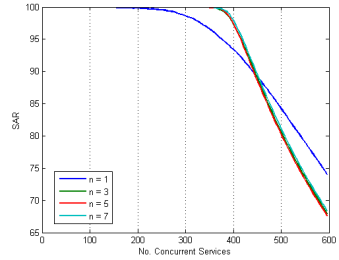
(c) Update time.



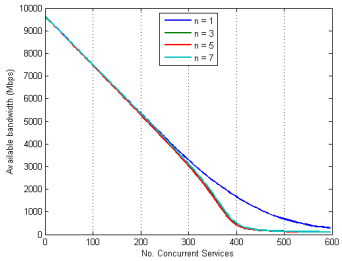
(d) Setup time.



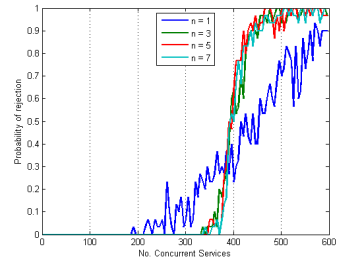
(e) No. Rejected services.



(f) Service Acceptance Ratio.



(g) Overall Available Bandwidth.



(h) Prob. service rejection.

Figure 8.4.: Effect of the path diversity in a 8 node full mesh network.

Table 8.1.: Summary of the results obtained for the full mesh topologies.

Topology	n_{MAX}	Mean t_s (ms)	95% Conf	Min t_s (ms)	Max t_s (ms)	Ser (95%)	SAR (%)
Mesh 5	1	0,88	0,12	0,76	1,00	163	88,36
	2	1,20	0,23	0,98	1,43	141	97,76
	3	1,63	0,42	1,22	2,05	137	98,80
	4	1,89	0,46	1,43	2,36	138	98,80
Mesh 6	1	1,05	0,20	0,84	1,25	243	89,79
	2	1,29	0,26	1,03	1,55	208	98,76
	4	1,92	0,34	1,58	2,26	204	98,65
	5	2,66	0,52	2,15	3,18	205	99,20
Mesh 7	1	0,98	0,10	0,87	1,08	347	88,07
	2	1,21	0,15	1,06	1,36	287	98,92
	5	6,70	0,71	6,00	7,41	287	99,56
	6	10,94	1,38	9,56	12,32	284	99,61
Mesh 8	1	0,92	0,10	0,83	1,02	462	88,20
	3	2,08	0,41	1,66	2,49	377	99,48
	5	22,50	2,08	20,41	24,58	376	99,62
	7	158,63	4,84	153,79	163,47	380	99,62

8.1.1.1 DISCUSSION

Table 8.1 summarizes the values obtained for the different topologies analyzed in this section. On the one hand, the average median t_{setup} expressed in ms, where the 95% confidence interval and the minimum and maximum values are also included are represented depending on the value of n_{MAX}. On the other hand, the number of concurrent services at which a 95% of the network resource utilization is achieved is indicated (Ser (95%) in the table), so as the SAR, expressed as a percentage.

Regarding the t_{setup}, it is worth mentioning that it is clearly affected by the number of alternative paths that are computed. This is the result of all measured times, $t_{ondemand}$, t_{check} and t_{update}, being affected by the number of alternative paths. Notwithstanding, as it is appreciated specially in Figures 8.4b and 8.4c, the impact on the t_{check} and t_{update} is minimal, within the fraction of the milliseconds, compared to the impact in the $t_{ondemand}$. The reason is on the kind of operation that is performed in each phase. On the one hand, both t_{check} and t_{update} require comparison operations on the array used to represent the consumption vector, which are not computationally intensive as it will be further explained in Section 8.2. On the other hand, the on-demand path computation time is heavily affected by the number of alternative paths, given the fact

that the randomization process needs to be applied over the whole set of alternative paths. Therefore, the higher the number of alternative paths is the higher the $t_{ondemand}$ is.

It is worth mentioning that the lower times achieved in the t_{setup} when the number of concurrent services increases is the result of the increasing number of rejected services. This is a consequence of splitting the admission control process in two different phases. Given that for the rejected services the update procedure is not executed, the t_{setup} decreases. As it is appreciated in Figure 8.4c, the number of concurrent services at which the value of t_{setup} decreases correlates with the number of concurrent services at which the services start to be rejected (see Figure 8.4e).

As appreciated both in the graphics and in Table 8.1, the network resource utilization gets improved when the n_{MAX} is higher than one. This can be concluded by looking at the concurrent service number at which the 95% of the network utilization is achieved. In general, the optimization of the network resource utilization is achieved earlier when n_{MAX} is higher than one, resulting in a higher SAR at that precise moment. That is, the probability of the services being accepted in the network increases, resulting in an increase of around the 10% in the SAR to achieve the desired network resource utilization.

The enhancement that is achieved in terms of network resource utilization thanks to the pre-computation of multiple alternative paths in an 8 node full mesh network is also depicted in Figure 8.5. In the figure, the network resource utilization that is achieved depending on the service rejection probability is represented as a percentage of the overall network capacity that is used. As illustrated, by a certain rejection probability, the network resource utilization that is achieved when values of n_{MAX} higher than one are used is always better than the one achieved with a single path alternative.

However, there is a point in which the number of rejected services is equal regardless of the value of n_{MAX}. This is appreciated in the figures where the number of rejected services is depicted, it corresponds with the concurrent service number at which all the lines intersect. From that moment, in the case of n_{MAX} equal to 1, the SAR obtained is better than the SAR obtained for higher values of n_{MAX}. This is a consequence

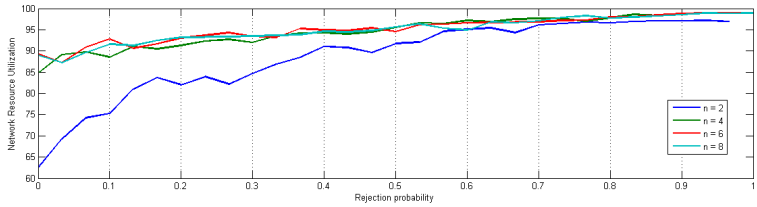


Figure 8.5.: Overall network resource utilization for a 8 node full mesh topology.

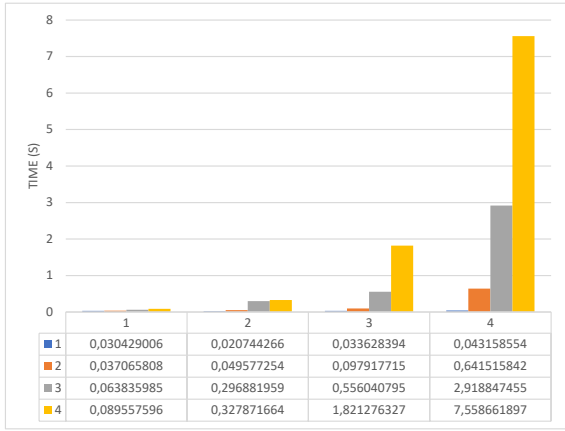


Figure 8.6.: Path pre-computation time.

of having almost a 100% of probability of rejecting new services when higher values of n_{MAX} are utilized, since the network resource utilization is already optimized. Therefore, the values obtained from that moment are not representative, although the author of this theses has decided to depict them to illustrate the phenomena.

Nevertheless, there is no appreciative improvement when more than two hop path alternatives are utilized. This is consistent with the research conducted by Microsoft for the design of SWAN [129], where they concluded that for their use case, a set of 15 alternative paths were sufficient to achieve the optimal network resource utilization. Given the fact that with the variation of the n_{MAX} is hard to state at which number of path alternatives the highest resource utilization is achieved, a more fine grained evaluation will be performed with the ESNet topology.

Notwithstanding, taking into account the low impact achieved for these kinds of network in terms of resource utilization when more than two hops' alternative paths are computed, it is worth studying the impact that generating such alternatives has in the solution. As illustrated in Figure 8.6, the time required to compute all the possible paths up to n_{MAX} hops increases exponentially. Even if the number of alternatives is limited, computing all the alternatives up to n_{MAX} takes a considerable amount of time, which is dependent on the number of nodes and links in the network. In fact, the time required to compute all the alternatives for a 10 node full mesh network takes an unacceptable time for the requirements of the system, and the amount of alternatives generated also results in an unacceptable memory consumption.

Therefore, it can be concluded that taking into account that higher number of alternative paths impacts negatively on the memory consumption of the solution, the amount of alternative paths should be kept to the minimum. The amount of alternative paths should be tailored to suit the requirements of the use case, and take into account the resilience mechanisms implemented by the solution. In this regard, it is worth noting that no resilience mechanisms have been utilized to conduct the tests, in order to simplify the analysis. Nevertheless, it is worth remarking that the t_{setup} in all the cases is in the order of the milliseconds. Taking into account that the establishment of these kinds of circuits in RENs can even require several days, it is a remarkable improvement.

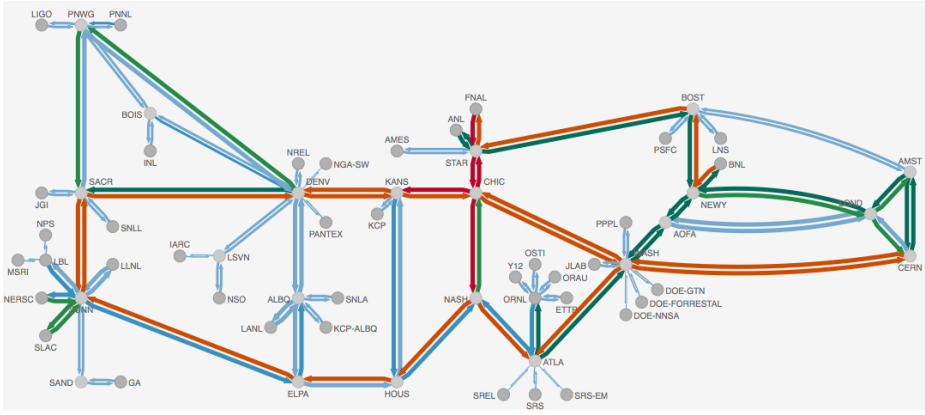
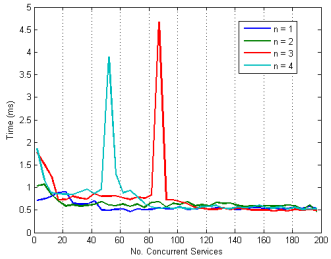


Figure 8.7.: Topology of the ESNet network.

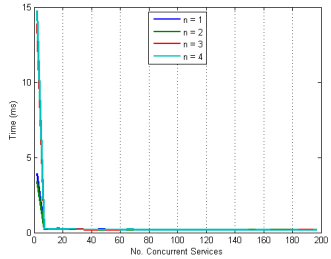
8.1.2 ESNET NETWORK

Since with the variation of n_{MAX} the set of alternative paths does not increase linearly, an additional constraint has been added to the path pre-computation algorithm used for the ESNet topology, which is depicted in Figure 8.7. In this case, the algorithm finds the set of K-shortest paths that do not exceed the maximum n_{MAX} hop number.

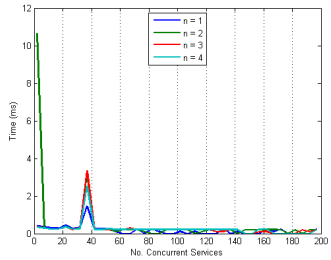
In this case, the tests have been conducted for different values of K, that is, of alternative paths. On the one hand, Figure 8.8 depicts the cases where the set of alternative paths ranges from 1 to 4, whereas Figure 8.9 illustrates the cases where the set of alternative paths has been limited to 1, 10, 100 and 1000. For each test, 200 services have been requested, with a peak bandwidth ranging from 300 Mbps to 1 Gbps in steps of 50 Mbps.



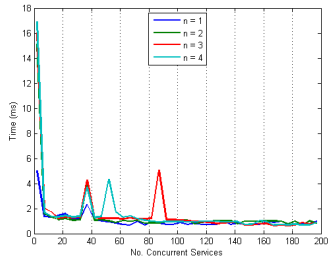
(a) On-demand path computation time.



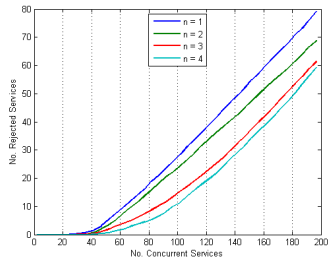
(b) Check time.



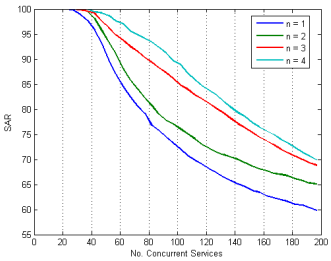
(c) Update time.



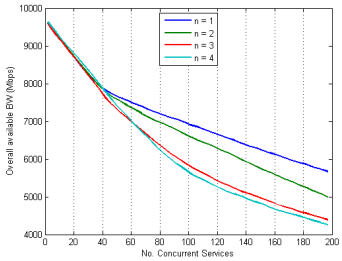
(d) Setup time.



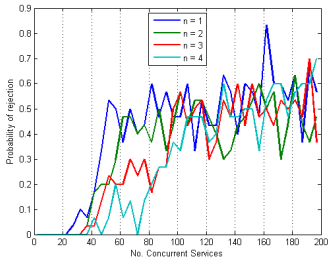
(e) No. Rejected services.



(f) Service Acceptance Ratio.

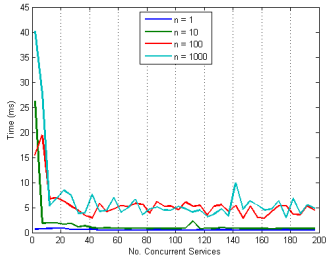


(g) Overall available bandwidth.

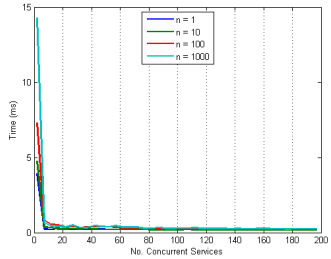


(h) Prob. Service Rejection

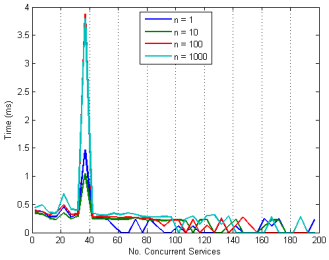
Figure 8.8.: Effect of the path diversity in the ESNet network.



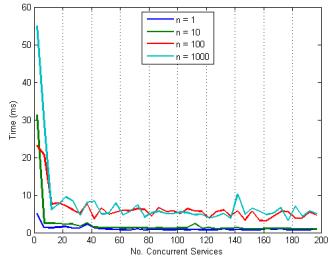
(a) On-demand path computation time.



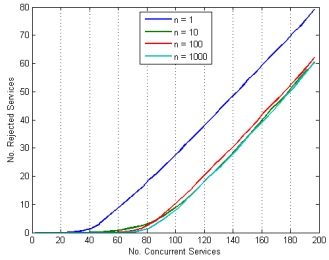
(b) Check time.



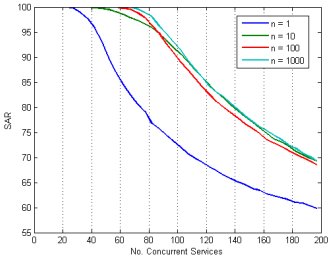
(c) Update time.



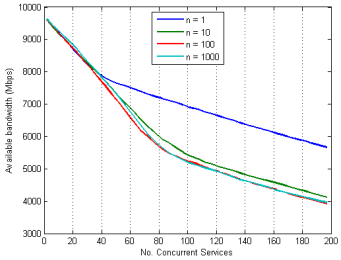
(d) Setup time.



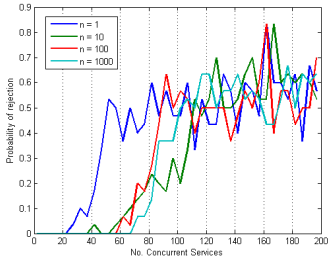
(e) No. Rejected services.



(f) Service Acceptance Ratio.



(g) Overall available bandwidth.



(h) Prob. Service Rejection

Figure 8.9.: Effect of the path diversity in the ESNet network.

Table 8.2.: Summary of the results obtained for the ESNet topology.

Topology	n_{MAX}	Paths	Mean t_s (ms)	95% Conf	Min t_s (ms)	Max t_s (ms)	Ser. (40%)	SAR (%)
ESNet	11	1	1,05	0,14	0,91	1,19	171	62,19
	11	2	1,13	0,17	0,96	1,29	138	70,46
	11	3	1,24	0,19	1,05	1,43	95	86,84
	11	4	1,34	0,25	1,08	1,61	89	92,23
	11	10	1,72	0,37	1,35	2,09	81	96,25
	11	100	6,28	0,83	5,44	7,11	73	98,84
	11	1000	8,29	2,04	6,24	10,34	75	99,34

8.1.2.1 DISCUSSION

Table 8.2 presents the average median t_{setup} expressed in ms, where the 95% confidence interval and the minimum and maximum values are also represented depending on the value of the number of alternative paths. On the other hand, the number of concurrent services at which the 40% of the network resource utilization is achieved is indicated (the selection of this value is later explained), so as the SAR, expressed as a percentage.

As it happened in the case of the full mesh network, the utilization of more alternative paths makes possible to achieve the optimal network resource utilization earlier, and with a lower penalty in terms of rejected services. It is worth remarking, that in this case, the difference between utilizing one alternative path or four results in almost a 30% of higher SAR, which is a considerable improvement. Notwithstanding, although the utilization of more than four alternative paths continues to improve the SAR, the effect is not that prominent. This is the consequence of the topology that has been used, where only the points that connect ESNet with other RENs have been included as source and destination end-points. Given the low level of path diversity that exists in these nodes, the links get oversubscribed rapidly, while the inner links of the topology remain under-utilized. This is also the reason of selecting the 40% of resource utilization as the optimization objective, since in RENs like GÉANT, the inter-REN traffic is usually around that percentage.

As in the case of the full mesh networks, when analyzing the network resource utilization that is achieved for a given rejection probability, the benefit of pre-computing multiple path-alternatives is demonstrated. Figure 8.10 depicts the overall network resource utilization that is achieved for a given service rejection probability depending on the number

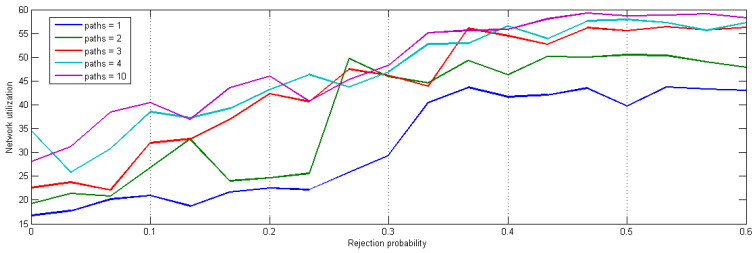


Figure 8.10.: Overall network resource utilization in the ESNet topology.

of alternative paths, which increases with the number of alternative paths that are used.

All in all, the obtained results are very promising, taking into account that the average median setup time remains in the order of the milliseconds when applied to a real REN's topology. This results validate the suitability of the advance reservation mechanism for its utilization in RENs.

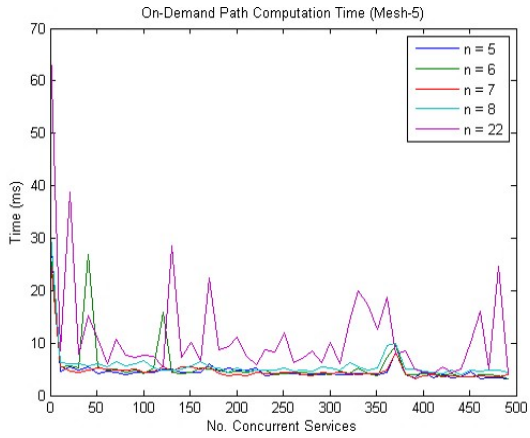


Figure 8.11.: Impact of the number of nodes in the setup time.

8.2 ANALYSIS OF THE IMPACT OF THE NUMBER OF LINKS ON THE SETUP TIME

As mentioned in Chapter V, the setup time considers both the check time and the update time, at least for the accepted services. All in all, these two times are dependent on the number of network links in the network, since in order to check or update the values of the nodes in the NSTree, it is necessary to operate over the consumption vector. In a nutshell, the consumption vector represents the aggregated available bandwidth for each unidirectional link in the network graph.

The consumption vector is implemented as a one dimensional array, therefore, the check or update operations are dependent on the size of this array. Notwithstanding, the utilization of a consumption vector that summarizes the availability of the network resources allows to perform this operations in $\mathcal{O}(k)$, being k the number of unidirectional links in the network graph. On the one hand, the checking phase only requires the comparison between the consumption vector computed for the alternative path being tested and the consumption vector of the given node in the NSTree. On the other hand, the update phase only requires the subtraction of the consumption vector computed for the alternative path being tested and the consumption vector of the given node in the NSTree.

Figure 8.11 depicts the values obtained for the first 500 concurrent services requested for the four full mesh networks presented in the

previous section and the ESNet network. The number of links is $n * (n - 1)/2$ for the full mesh topologies, where n is the number of nodes and 66 in the case of the ESNet topology.

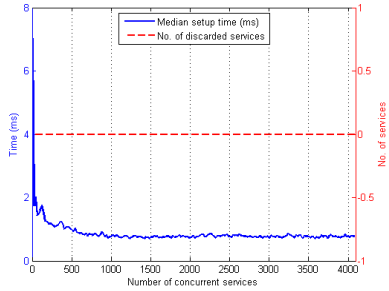
As it can be deducted, for the full mesh networks, the setup time is constant regardless of the already accepted services in the network snapshot, although the value gets slightly affected by the size of the array. On the other hand, for the ESNet topology there is a higher variation on the setup time. This is a consequence of the lower SAR achieved with the ESNet topology, since the update phase is only executed for the accepted services. Notwithstanding, in all the cases the median setup time remains below the 30 ms, which as mentioned before, satisfies the requirements of an advance reservation system in a RENs.

8.3 ANALYSIS OF THE IMPACT OF THE REQUESTED BANDWIDTH ON THE SETUP TIME AND THE NUMBER OF REJECTED SERVICES

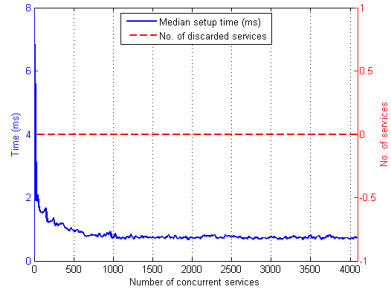
This experiment aims to demonstrate how the algorithms used in the DynPaC framework behave independent on the amount of bandwidth requested for the given services. In that regard, a set of tests have been conducted using service requests for a single network snapshot with random source and destination nodes to ensure a uniform distribution, as depicted in Figure 8.12, where the 8 node full mesh network has been used. The selected peak bandwidth values to conduct this experiment are 1 Mbps (see Figure 8.12a), 10 Mbps (see Figure 8.12b), 100 Mbps (see Figure 8.12c), 1 Gbps (see Figure 8.12d), 10 Gbps (see Figure 8.12e) and random values of bandwidth between 10 Mbps and 500 Mbps (see Figure 8.12f).

Please note that the number of concurrent services has been limited taking into account the amount of bandwidth requested by the services, since for higher values of bandwidth, lower concurrent services can be accepted in the network. For each case, the setup time as an output of the number of concurrent services is provided. It is worth remarking that for all the cases, the setup time remains below the 2 ms.

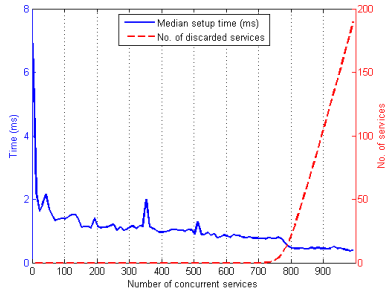
Nevertheless, as depicted in the graphics the setup time gets decreased for higher values of concurrent services as a consequence of a higher amount of services being rejected, since for the rejected services, the update phase is not executed. Please note that the higher setup time obtained for the first service reservation is the consequence of creating the NSTree.



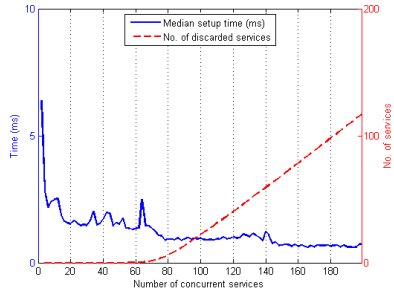
(a) 1 Mbps.



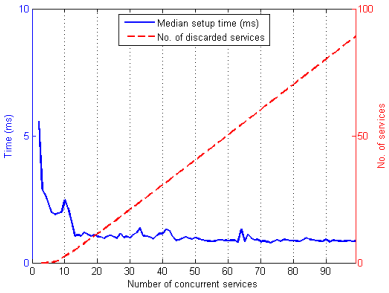
(b) 10 Mbps.



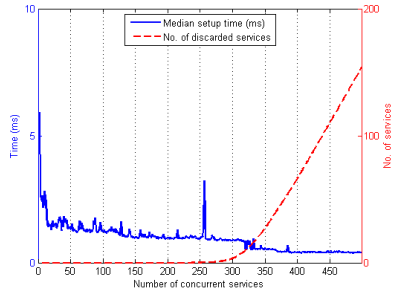
(c) 100 Mbps.



(d) 1 Gbps.



(e) 10 Gbps.



(f) Random bandwidth.

Figure 8.12.: Effect of the requested bandwidth.

8.4 ANALYSIS OF THE IMPACT OF THE AGGREGATION POLICY

One of the main contributions of this thesis is the NSTree, a hierarchical data structure where the network snapshots are aggregated into common ancestors depending on time constraints. As a consequence, this section evaluates the impact of the aggregation policy in the performance of the system. The experiment evaluates the impact of the selected δ on the performance of the solution, where δ refers to the time that needs to exist between two nodes in the NSTree to be aggregated into a common CA. More precisely, four different values of δ have been used: none, one day, one week and one month[2].

For this experiment, the ESNet topology has been used, in order to asses the suitability of the aggregation policy in a real REN's topology, where 500 services have been requested. The requested services have been generated using the following parameters:

- Random bandwidth from 100 Mbps to 2 Gbps.
- Random start time and random end time, ensuring a duration between 1 day and 4 weeks.
- Random source and destination nodes to ensure a uniform distribution of the service requests.

First of all, even if the value of δ is modified, the number of network snapshots that are generated remain constant (see Figure 8.13c). This is the expected behavior, since the variation of the δ does not affect how the network snapshots are generated, which only depend on the service start and end times and their relative overlapping.

Of the analysis of the graphics illustrated in Figure 8.13 can be deducted that, on the one hand, there is no significant impact on the check time (see Figure 8.13a). Even if the number of generated CAs varies (see Figure 8.13d), the only operation that is performed in each of the nodes is the comparison of the consumption vectors, since there is no need to create or update the nodes in the NSTree.

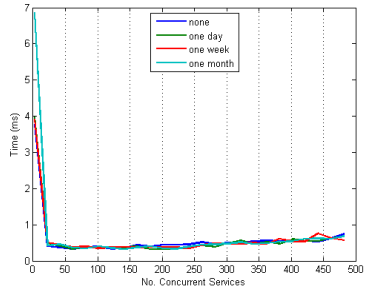
2 Please note that for the *none* case, CAs are still created as a consequence of the overlapping cases.

On the other hand, the effect on the update phase can be seen in Figure 8.13b). It must be taken into account that in the update phase, a lower number of CAs imposes a higher number of calls to the add service algorithm presented in Chapter VI. As such, for the case where no δ is specified, and therefore, a lower number of CAs are generated, the time required to update the NSTree is higher.

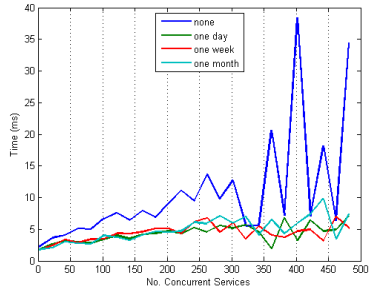
Regarding the number of rejected services (see Figure 8.13e), there is a minimal difference for the different values of deltas. This is the expected behavior, since the rejection of a service ultimately depends on the resources available in each network snapshot. The generation of CA reduces the time required to state whether a service can be accepted or not, but it does not have an impact on the fact that a service reservation will only be accepted if there are enough resources in the network. For the concurrent service number 500, the number of rejected services varies between 12 and 20, which translates as a difference of the 0,016%. This is a value that is not statistically significant, and the result of the random request of resources.

The same happens with the overall network utilization (see Figure 8.13f). In the end, the generation of CAs eases the admission control process, but ultimately, the network has a set of determined resources, and the utilization of the same service reservation pattern results in the same overall network utilization.

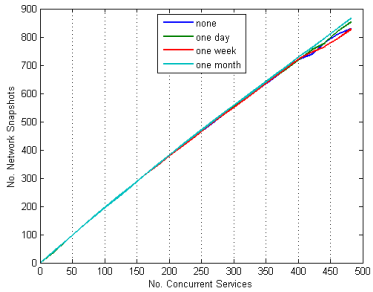
As it can be concluded, the utilization of a hierarchical structure, where the CAs aggregate multiple NSTree nodes and advertise a set of common resources that are available through a wider time span, results in a lower t_{update}. However, the effect of the aggregation is not very prominent in the case of the t_{check}, given the simple operations that are performed in this phase. Moreover, the introduction of this strategy does not have an impact in the number of network snapshots that are generated, neither in the number of rejected services nor in the overall network resource utilization, since these parameters are independent on the number of CAs.



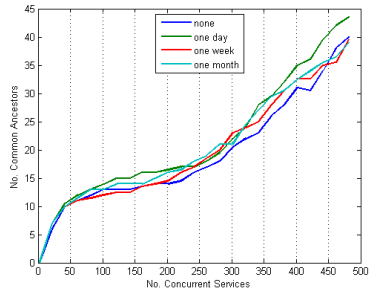
(a) Check time.



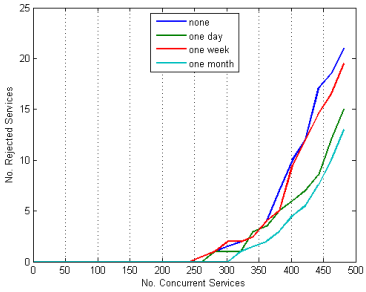
(b) Setup time.



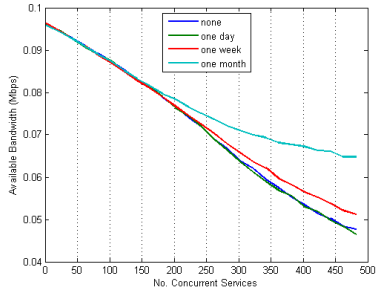
(c) Number of NS nodes.



(d) Number of CA nodes.



(e) Number of rejected services.



(f) Overall available bandwidth.

Figure 8.13.: Effect of the value of delta.

Part V

CONCLUSIONS

9 Conclusions and future work

> "The voyage was long. By the end of it, Val had finished the first volume of her history of the bugger wars and transmitted it by ansible, under Demosthenes' name..."
>
> — Orson Scott Card

9.1 INTRODUCTION

This chapter summarizes the main contributions and conclusions of this PhD thesis, and presents a comprehensive list of publications where the results of this research effort have been disseminated and the projects that have funded this work.

In a nutshell, the main contribution of this PhD thesis is a generic framework for TE in SDN called DynPaC with support for advance reservations that enables the introduction of novel TE strategies that leverage the benefits of SDN. During the four years in which this work has been carried out, the DynPaC framework has evolved from a prototype implemented in ODP to be an application running in ONOS. More importantly, by the end of 2017, the pan-European REN GÉANT will launch the pilot of the SDN-based BoD service. In this pilot, DynPaC will be used to control a set of Corsa whiteboxes distributed across Europe, providing access to this new service to real users from the research and academic community. By the end of the GN4-2 project, which is the

project actually co-funding this research work, it is expected to have the DynPaC application on a Technology Readiness Level (TRL) 8.

Furthermore, during the last year, both GÉANT and the University of the Basque Country have been included as official contributors to the open source ONOS project. In this regard, the Open Networking Laboratory in charge of the development of ONOS has agreed to include the DynPaC framework as an official application. Furthermore, some of the functionalities included in the DynPaC framework are planned to be included in future releases of the core ONOS controller.

Moreover, I would like to emphasize that the output of this PhD thesis is not only theoretical, but also practical. All the algorithms, data structures and functionalities have been conceived to create a real product, viable for its utilization not only in RENs but also in other scenarios. And this is, according to this PhD candidate, the most important outcome of this research effort.

9.2 CONTRIBUTIONS

As demonstrated throughout this document, the DynPaC framework is able to support advance reservations in SDN domains, and satisfies the set of requirements identified for the future BoD service of RENs. The following list summarizes the most relevant contributions of this research work:

1. **Analysis of current solutions for TE in SDN**: An initial in-depth analysis of current TE solutions based on SDN technologies has proved the impact of the protocols operating at the D-CPI interface in TE. Furthermore, taking that study as basis, a representative set of TE solutions that rely on D-CPI protocols have been evaluated, in order to identify their strengths and limitations regarding their support for TE and their applicability to different use cases, making special emphasis on their suitability for the RENs. As a result of this evaluation, it has been possible to detect the need of a generic framework for TE in SDN. Moreover, it has also been possible to identify the set of features that the framework should provide.

2. **Analysis of the impact of the data structures to support advance reservations**: Although the framework proposed in this PhD thesis aims to support a wide range of use cases, it has been designed to support advance reservations, since it is a desired feature in RENs and represents one of the most challenging aspects of TE. As such, an analysis of classic and current approaches to support these kinds of reservations has been conducted. More precisely, the analysis has been focused on the impact that the data structures utilized to handle time constraints has on the performance of the advance reservation solution. This study has allowed to identify a set of characteristics that the data structures used to handle advance reservations in the DynPaC framework should consider.

3. **Design of a generic framework for TE in SDN:** As stated before, one of the main contributions of this research work is a generic framework for TE in SDN, named DynPaC. The framework follows a modular approach, facilitating the introduction of new functionalities and the upgrade of the existing ones. In addition, it has been designed to leverage the logically centralized control plane of SDN and its high programmability, providing the means to include novel TE strategies.

4. **Available resource management on a per network basis:** Another relevant contribution of this PhD thesis is the design of a data structure to handle the time constraints of advance reservations on a per network basis, as opposed to the classical approaches where resource management is handled on a per link basis. The DynPaC framework relies on *network snapshots*, which allow to represent a period of time during which the network resource consumption stays in a stable state, consequence of having a determined set of concurrent services reserved for that specific period of time.

5. **Hierarchical aggregation of network snapshots to reduce the time complexity of the solution:** A novel data structure to handle the time constraints in an efficient manner has been designed. The NSTree arranges the network snapshots hierarchically, aggregating different network snapshots into common ancestors

depending on their relative separation in terms of time. In addition, this approach allows to ensure the utilization of the same path for a given service during wider time intervals, since the paths are computed first using the available resources at the common ancestors that aggregate multiple network snapshots. This increases the network stability and reduces the packet loss, since the SDN devices are not constantly programmed to install the new paths and remove the old ones. Furthermore, this also improves the time required to update the NSTree nodes, since instead of computing a path for each network snapshot, the common solution is propagated downwards the data structure until the network snapshots.

6. **Failure recovery capabilities:** The DynPaC framework has been designed to support a wide range of failure recovery approaches. On the one hand, DynPaC supports service protection, where the resources that would be consumed by the backup path are also reserved. On the other hand, it also supports service restoration, where a backup path can be computed on-demand for a service affected by a network failure by simply re-requesting the service using the updated network topology.

7. **TE optimization techniques to improve the network resource utilization:** In order to improve the network resource utilization a set of TE techniques have been proposed, namely flow relocation and flow disaggregation. Both techniques seek to accept in the network new service reservation requests that with the actual set of reservations would be rejected, by preempting the already accepted services into alternative but equally valid paths. On the one hand, flow relocation allows to assign different paths for both already installed and reserved services. On the other hand, flow disaggregation is applied to services actually installed in the network, which are divided into smaller and easier to relocate sub-services thanks to the information provided by an external entity about their traffic pattern.

8. **Support for multi-domain capabilities:** Given the relevance that multi-domain connectivity services have in RENs, the DynPaC framework has been designed to be compliant with the NSF

framework. DynPaC supports an asynchronous operational mode that allows an external entity, in this particular case an NSA, to handle the life-cycle of the services. In addition, the REST interface designed to facilitate the communication with the NSA can also be used for its integration with other kinds of orchestrators.

9. **Performance evaluation of the advance reservation mechanism:** In order to state the suitability of the data structures and the associated algorithms designed to support advance reservations in SDN, a performance evaluation has been conducted. The results show that the DynPaC framework is able to handle thousands of service reservations requests for different periods of time, and that the admission control mechanism is able to compute the optimal paths in the order of the milliseconds. Taking into account that current BoD services usually involve network operators manually configuring network devices and that the process, especially in the multi-domain case, can take even days, this is a considerable improvement.

9.3 DISSEMINATION OF THE RESULTS

The contributions and results of this PhD thesis have been disseminated in a varied range of journals and international conferences and workshops. As such, the following subsections present the list of associated publications, so as the list of projects in which this PhD work has been carried out.

9.3.1 PUBLICATIONS IN INTERNATIONAL JOURNALS

At the moment of writing this PhD thesis, a single article has been published in an indexed journal. Notwithstanding, this PhD candidate is currently working on a set of articles to present the algorithms and their performance evaluation to be submitted in the near future. Nevertheless, it is worth remarking that the published article has been accepted in the most relevant journal in the area of Telecommunications and Computer networks, the IEEE Communications surveys and tutorials. In addition, the DynPaC framework and its multi-domain capabilities have also been

published as selected papers of the Terena Networking Conference, which is one of the most prestigious conferences in the REN environment.

- **A survey on the contributions of Software-Defined Networking to Traffic Engineering.** [87]
  - *Journal:* IEEE Communication Surveys and Tutorials
  - *Year:* 2017
  - *Impact factor:* 9.22 (Q1)
  - *Citations:* 1
- **DynPaC: Dynamic and Adaptive Traffic Engineering for SDNs.** [Selected paper] [228]
  - *Journal:* Selected papers of Terena Networking Conference 2015 (ISBN: 978-90-77559-25-3)
  - *Year:* 2015
- **Multi-domain Software Defined Networking: Exploring possibilities.** [Selected paper] [223]
  - *Journal:* Selected papers of Terena Networking Conference 2014 (ISBN: 978-90-77559-24-6)
  - *Year:* 2014

9.3.2 PUBLICATIONS IN PROCEEDINGS OF INTERNATIONAL CONFERENCES

The DynPaC framework has been published in the proceedings of multiple international conferences of special relevance to the topic of this PhD thesis, as listed below:

- **Towards an SDN-based bandwidth on demand service for the European research community.** [226]
  - *Conference:* International Conference on Networked Systems
  - *Year:* 2017
- **Multi-domain bandwidth on demand service provisioning using SDN.** [229]

  - *Conference:* IEEE Conference on Network Softwarization
  - *Year:* 2016
  - *Citations:* 3

- **DynPaC: A Path Computation Framework for SDN.** [230]
  - *Conference:* European Workshop on Software Defined Networks
  - *Year:* 2015
  - *Citations:* 6

9.3.3 ORAL COMMUNICATIONS

During the development of this PhD thesis, this PhD candidate has also participated in the following set of international events as speaker:

- **SDN-enabled AutoBAHN. Envisioning future BoD services.** [231]
  - *Conference:* Terena Networking Conference
  - *Year:* 2016

- **Adapting the SDN-based BoD application to the particularities of the Corsa HW datapath.** [232]
  - *Conference:* GÉANT Symposium
  - *Year:* 2016

- **Demonstration of the multi-domain BoD application.** [233]
  - *Conference:* GÉANT Symposium
  - *Year:* 2016

- **A solution for connection-oriented multi-domain SDN based on NSI.** [234]
  - *Conference:* Terena Networking Conference
  - *Year:* 2015

- **Demonstration of connection-oriented multi-domain OpenFlow.** [222]

  - *Conference:* International Conference on Supercomputing
  - *Year:* 2014

9.3.4 OTHER PUBLICATIONS RELATED TO THIS PHD THESIS

This PhD thesis is the result of many years of research in SDN. As such, the following list summarizes a comprehensive list of publications, including indexed journals and international conferences, that although they do not present the research work conducted for this PhD thesis, are relevant to the field:

- **Empowering GÉANT deployments with ONOS brigades.** [235]
  - *Conference:* Terena Networking Conference
  - *Type:* Oral communication
  - *Year:* 2017
- **GÉANT SDX - SDN based Open eXchange Point.** [236]
  - *Conference:* IEEE Conference on Network Softwarization
  - *Type:* Demonstrator
  - *Year:* 2016
  - *Citations:* 1
- **An architecture for dynamic QoS management at Layer 2 for DOCSIS access networks using OpenFlow.** [237]
  - *Journal:* Computer Networks
  - *Year:* 2016
  - *Impact factor:* 1.446 (Q2)
- **Selfdeploying Service Graphs over ELwUD (EHU-OEF Lightweight UNIFY Domain).** [227]
  - *Conference:* IEEE Conference on Network Softwarization
  - *Type:* Demonstrator
  - *Year:* 2015

- **Integrating complex legacy systems under OpenFlow control: The DOCSIS use case.** [238]
  - *Conference:* European Workshop on Software Defined Networks
  - *Type:* Oral communication
  - *Year:* 2014
  - *Citations:* 3
- **FlowNAC: Flow-based network access control.** [239]
  - *Conference:* European Workshop on Software Defined Networks
  - *Type:* Oral communication
  - *Year:* 2014
  - *Citations:* 24
- **The EHU-OEF: An OpenFlow-based Layer-2 experimental facility.** [240]
  - *Journal:* Computer Networks (Q2)
  - *Year:* 2014
  - *Impact factor:* 1.22 (Q2)
  - *Citations:* 10
- **Deploying a virtual network function over a software defined network infrastructure: experiences deploying an access control VNF in the University of Basque Country's OpenFlow enabled facility.** [Selected paper] [241]
  - *Conference:* Terena Networking Conference
  - *Type:* Oral communication
  - *Year:* 2014
- **Enriched slices in EHU-OEF empowered by NFV: the Access Control NFV use case.**
  - *Conference:* Open Networking Summit 2014
  - *Type:* Demonstrator

  - *Year:* 2014

- **Access Control NFV enforcement at the EHU-OpenFlow enabled facility.** [242]
  - *Conference:* IEEE Globecom Industrial Forum & Exhibits
  - *Type:* Demonstrator
  - *Year:* 2013

- **Solución de virtualización de nivel 2 basada en prefijos para plataformas de experimentación.** [243]
  - *Conference:* Jornadas Técnicas RedIRIS 2012
  - *Type:* Oral communication
  - *Year:* 2012

- **Implementing layer 2 network virtualization using OpenFlow: Challenges and solutions.** [244]
  - *Conference:* European Workshop on Software Defined Networks
  - *Type:* Oral communication
  - *Year:* 2012
  - *Citations:* 33

9.3.5 PARTICIPATION IN PROJECTS

This PhD thesis is the result of the work conducted as a researcher associated to the I2T Research Group of the University of the Basque Country (UPV/EHU) from 2013 to 2017. During that period of time, the PhD candidate has participated in the following national and international projects related to the research topic of this PhD thesis.

- Despliegue seguro de servicios con Redes Definidas por Software y Virtualizacion de Funciones de Red (S&N-SEC).
  - *Funding entity:* Spanish ministry of Economy and Competitiveness.
  - *Participants:* UPV/EHU

  - *Duration:* 1-1-2014 to 31-12-2016
  - *Main researcher:* Eduardo Juan Jacob Taquet and Juan José Unzilla Galán.
  - *Number of participating researchers:* 8
  - *Project budget:* 105.391,00 €

- H2020-EINFRA-2014-2 GN4 – Phase 2.
  - *Funding entity:* European Community (H2020(
  - *Participants:* ASNET-AM, ACOnet, AzRENA, BASNET, Belnet, BREN, CARNet, CyNET, CESNET, EENet, DANTE, TERENA, RENATER, GRENA, DFN, GRNET, NIIFI, HEAnet, IUCC, GARR, SigmaNet, LITNET, RESTENA, MARnet, University of Malta, RENAM, MREN, SURFnet, NORDUnet, PSNC, FCCN, RoEduNet, eARENA, AMRES, SANET, ARNES, RedIRIS, SWITCH, ULAKBIM, JANET, URAN.
  - *Duration:* 01-05-2016 to 31-12-2018.
  - *Main researcher:* Eduardo Juan Jacob Taquet
  - *Number of participating researchers:* : 3 (UPV/EHU)
  - *Project budget:* 86.953,25 €

- H2020-EINFRA-2014-2 GN4 – Phase 1.
  - *Funding entity:* European Community (H2020)
  - *Participants:* ASNET-AM, ACOnet, AzRENA, BASNET, Belnet, BREN, CARNet, CyNET, CESNET, EENet, DANTE, TERENA, RENATER, GRENA, DFN, GRNET, NIIFI, HEAnet, IUCC, GARR, SigmaNet, LITNET, RESTENA, MARnet, University of Malta, RENAM, MREN, SURFnet, NORDUnet, PSNC, FCCN, RoEduNet, eARENA, AMRES, SANET, ARNES, RedIRIS, SWITCH, ULAKBIM, JANET, URAN.
  - *Duration:* 01-04-2015 to 31-03-2016
  - *Main researcher:* Eduardo Juan Jacob Taquet
  - *Number of participating researchers:* 4 (UPV/EHU)

  - *Project budget:* 53.600 €

- Dynamic Path Computation Framework (DynPaC), Open Call project associated to INFRA-2013-1.2.1 – GÉANT Multi-Gigabit European Research and Education Network and Associated Services (GN3plus).
  - *Funding entity:* European Community (FP7)
  - *Participants:* ASNET-AM, ACOnet, AzRENA, BASNET, Belnet, BREN, CARNet, CyNET, CESNET, EENet, DANTE, TERENA, RENATER, GRENA, DFN, GRNET, NIIFI, HEAnet, IUCC, GARR, SigmaNet, LITNET, RESTENA, MARnet, University of Malta, RENAM, MREN, SURFnet, NORDUnet, PSNC, FCCN, RoEduNet, eARENA, AMRES, SANET, ARNES, RedIRIS, SWITCH, ULAKBIM, JANET, URAN, EHU
  - *Duration:* 24-10-2013 to 23-03-2015
  - *Main researcher:* Eduardo Juan Jacob Taquet
  - *Number of participating researchers:* 2 (UPV/EHU)
  - *Project budget:* 148.320,00€ UPV/EHU)

- GÉANT Multi-Gigabit European Research and Education Network and Associated Services (GN3plus).
  - *Funding entity:* European Community (FP7)
  - *Participants:* ASNET-AM, ACOnet, AzRENA, BASNET, Belnet, BREN, CARNet, CyNET, CESNET, EENet, DANTE, TERENA, RENATER, GRENA, DFN, GRNET, NIIFI, HEAnet, IUCC, GARR, SigmaNet, LITNET, RESTENA, MARnet, University of Malta, RENAM, MREN, SURFnet, NORDUnet, PSNC, FCCN, RoEduNet, eARENA, AMRES, SANET, ARNES, RedIRIS, SWITCH, ULAKBIM, JANET, URAN
  - *Duration:* 01-4-2013 to 31-09-2015
  - *Main researcher:* Eduardo Juan Jacob Taquet
  - *Number of participating researchers:* 2
  - *Project budget:* 41.800.000,00 (65.145,60 € UPV/EHU)

9.4 FUTURE WORK AND RESEARCH LINES

Finally, given the fact that TE and path computation in SDN is a very important topic, as demonstrated by the growing interest of the industry in this regard, further work will be conducted in the future. Although there has been a lot of progress beyond the current state of the art and the obtained results are very promising, we will continue working to make DynPaC production ready. As such, the following list summarizes the future work and research lines:

- **Application of the DynPaC framework to non-OpenFlow networks:** Although one of the main features of the DynPaC framework is that it has been designed to be technology agnostic, its current implementation is focused on OpenFlow networks and their capabilities. Notwithstanding, MPLS is still the most popular forwarding technology among NSPs and RENs. In such a scenario, there are plans to evolve the DynPaC framework to handle the particularities of such networks, and extend it to operate with BGP-LS/PCEP in order to provide a valid solution for those network operators still reluctant to adopt OpenFlow. In that regard, there are also plans to compare the effectiveness of the solution in OpenFlow and MPLS networks, in order to determine the impact of the distributed LDP used in MPLS versus the out-of-band mechanisms available in OpenFlow to control the network elements.

- **Utilization of statistical analysis techniques to identify traffic patterns:** As mentioned before, the DynPaC framework relies on an external element to retrieve the information about the traffic patterns of the already installed services to disaggregate them. Notwithstanding, it has not been specified how that information is obtained. In that regard, there are plans to analyze traffic patterns, for instance, by using machine learning algorithms to find the optimal ways of disaggregating the services.

- **Enhancement of the path computation algorithm:** Although the path pre-computation strategy is valid for the requirements of RENs, there is plenty of room for improvement. After some discussions with the network operations team of GÉANT, further

research will be done to generate better alternative paths taking into account the topology of the network and current traffic loads.

- **Deployment of the SDN-based BoD pilot service:** As pointed out in the previous section, the next step is to deploy the DynPaC framework to provide the BoD service to a selected group of users across Europe. The main idea of this pilot deployment is to gather the feedback from the users, in order to improve the solution to satisfy their needs.

All in all, this is a very relevant research work which has attracted other RENs besides GÉANT and ISPs. As such, the author of this PhD thesis is confident that this research effort will result on the participation of the University of the Basque Country in more research projects related to SDN and the Future Internet. Moreover, it is expected that the future work presented in this section will result in new final degree projects and PhD thesis.

Part VI

APPENDIX

Appendix: Reservations generator script

For the performance evaluation a set of scripts have been used to generate the service reservations and request them to DynPaC using the REST interface. An example of such an script is provided below:

```
import pycurl
from urllib import urlencode
import random
import json
from StringIO import StringIO
import csv
from datetime import date
import commands
import datetime
from sys import argv
import time

arg = argv

def getNodePairs():
        return random.sample(set(nodes), 2)

def getEdgePort(node):
        return random.sample(set(egressPorts[node]),1)[0]

def getDates():
        start_date = datetime.datetime.today().replace(
            day=1, month=7, year=2017, hour=0, minute=0,
            second=0, microsecond=0).toordinal()
        end_date = datetime.datetime.today().replace(day
            =1, month=12, year=2017, hour=23, minute=59,
            second=59, microsecond=0).toordinal()
```

```
            random_startdate = datetime.datetime.fromordinal(
                random.randint(start_date, end_date))
            duration1 = datetime.timedelta(hours=random.
                randint(0,24))
            random_startdate = random_startdate + duration1
            duration = datetime.timedelta(weeks=random.
                randint(0,3), days=random.randint(0,7), hours=
                random.randint(0,24))
            random_enddate = random_startdate + duration
            reservation = (random_enddate - random_startdate)
                .total_seconds()
            dates = []
            dates.append(random_startdate.date().strftime("%d
                /%m/%Y"))
            dates.append(random_enddate.date().strftime("%d/%
                m/%Y"))
            dates.append(random_startdate.time().strftime("%H
                :%M:%S"))
            dates.append(random_enddate.time().strftime("%H:%
                M:%S"))
            dates.append(reservation)
            return dates

    # Retrieve edge nodes and ports

    for i in range (11, 31):
            print "Iteration:" + str(i)
            commands.getoutput('onos-app localhost reinstall!
                 ~/dynpac/target/dynpac-1.0-SNAPSHOT.oar')
            time.sleep(60)

            buffer = StringIO()
            c = pycurl.Curl()
            c.setopt(c.URL, 'http://localhost:8181/controller
                /web/dynpacServices//node-ports')
            c.setopt(c.WRITEDATA, buffer)
            c.perform()
            c.close()

            nodePorts = json.loads(buffer.getvalue())
            egressPorts = {}
```

```
54
        for key in nodePorts.keys():
                if len(nodePorts[key]['ports']) > 0:
                        ports = []
                        for port in nodePorts[key]['ports
                            ']:
59                              if port != 'LOCAL':
                                        ports.append(port
                                            )
                        egressPorts[key] = ports

        nodes = egressPorts.keys()
64
        file_name = 'esnet_th_40_' + str(i) + '.csv'

        with open(file_name, 'wb') as csvfile:

69              wr = csv.writer(csvfile, quoting=csv.
                    QUOTE_ALL)
                wr.writerow(['SERVICE','SOURCE', '
                    DESTINATION', 'BW', 'START', 'END', '
                    DURATION', 'RESULT', 'INIT TIME', 'FP
                    TIME', 'CHECK TIME', 'UPDATE TIME', '
                    PROCESS TIME', 'NS', 'CA','REST TIME',
                     'OCCUPATION'])

                for i in range(1,int(arg[1])+1):
                        [src, dst] = getNodePairs()
74                      inport = getEdgePort(src)
                        outport = getEdgePort(dst)
                        name = "test" + str(i)
                        bandwidth = random.randrange(0,
                            5000, 500)
                        vlan = str(i)
79                      [startDate, endDate, sT, eT,
                            duration] = getDates()


                        body = '{\"serviceType\":\"VLAN
                            \",\"ingressPort\":\"' +
                            inport + '\",\"egressPort\":\"
```

```
    ' + outport + '\",\"endDate
    \":\"' + endDate + '\",\"
    bandwidth\":\"' + str(
    bandwidth) + '\",\"
    vlanIdIngress\":\"' + vlan + '
    \",\"vlanIdEgress\":\"' + vlan
     + '\",\"name\":\"' + name + '
    \",\"startTime\":\"' + sT + '
    \",\"endTime\":\"' + eT + '
    \",\"solicitService\":\"true
    \",\"startDate\":\"' +
    startDate + '\"}'

buffer2 = StringIO()
c = pycurl.Curl()
c.setopt(c.URL, 'http://localhost
    :8181/controller/web/
    dynpacServices//service')
post_data = {'x-page-url': '
    dynpacServices', 'body': body,
     'action':'add', 'srcNodeId':
    src,'dstNodeId':dst}
postfields = urlencode(post_data)
c.setopt(c.POSTFIELDS, postfields
    )
c.setopt(c.WRITEDATA, buffer2)
c.perform()

statusDescriptor = buffer2.
    getvalue().split("#")
descriptor = statusDescriptor[0]
if (len(statusDescriptor) > 1):
        time_stamps =
            statusDescriptor[1].
            split("$")
        initialTime = time_stamps
            [0]
        fpTime = time_stamps[1]
        checkTime = time_stamps
            [2]
```

```
                updateTime = time_stamps
                    [3]
                processTime = time_stamps
                    [4]
                numberNs = time_stamps[5]
104             numberCs = time_stamps[6]


            result_list = []
            result_list.append(name)
109         result_list.append(src)
            result_list.append(dst)
            result_list.append(bandwidth)
            result_list.append(startDate + "
                //" + sT)
            result_list.append(endDate + "//"
                + eT)
114         result_list.append(duration)
            if (c.getinfo(c.RESPONSE_CODE) ==
                200):
                    result_list.append(
                        descriptor)
                    if (len(statusDescriptor)
                        > 1):
                            result_list.
                                append(
                                initialTime)
119                         result_list.
                                append(fpTime)
                            result_list.
                                append(
                                checkTime)
                            result_list.
                                append(
                                updateTime)
                            result_list.
                                append(
                                processTime)
                            result_list.
                                append(
                                numberNs)
```

```
124                                     result_list.
                                            append(
                                            numberCs)
                                else:
                                        result_list.
                                            append("NA")
                                        result_list.
                                            append("NA")
                                        result_list.
                                            append("NA")
129                                     result_list.
                                            append("NA")
                                        result_list.
                                            append("NA")
                                        result_list.
                                            append("NA")
                                        result_list.
                                            append("NA")
                            else:
134                             result_list.append("
                                    failure")
                                result_list.append("NA")
                                result_list.append("NA")
                                result_list.append("NA")
                                result_list.append("NA")
139                             result_list.append("NA")
                                result_list.append("NA")
                                result_list.append("NA")

                            result_list.append(c.getinfo(c.
                                TOTAL_TIME))
144
                            c.close()

                            buffer3 = StringIO()
                            c = pycurl.Curl()
149                         c.setopt(c.URL, 'http://localhost
                                :8181/controller/web/
                                dynpacServices//avg-bandwidth'
                                )
                            c.setopt(c.WRITEDATA, buffer3)
```

```
                c.perform()
                c.close()

154             avg_bw = json.loads(buffer3.
                    getvalue())
                #print type(avg_bw)
                avg = 0
                size = 1
                for s in avg_bw:
159                     avg = avg + float(s)
                        size = size + 1
                bw = avg / size
                result_list.append(bw)
                wr.writerow(result_list)
```

Bibliography

[1] Kate Greene. "TR10: Software-Defined Networking." In: *MIT Technology Review* (2009). Accessed: 2016-05-25. URL: http://www2.technologyreview.com/article/412194/tr10-software-defined-networking/.

[2] *The Top 5 Emerging Technology Trends of 2014*. http://www.avaya.com/usa/perspectives/articles/the-top-5-five-emerging-technology-trends-of-2014. Accessed: 2014-06-24.

[3] *Cisco's SDN solutions*. http://www.cisco.com/c/en/us/solutions/software-defined-networking/overview.html. Accessed: 2016-06-22.

[4] *HP SDN Solutions*. accessed: 2016-05-25. 2015. URL: http://www8.hp.com/us/en/networking/sdn/portfolio.html.

[5] *NEC SDN Solutions*. accessed: 2016-05-25. 2015. URL: http://www.nec.com/en/global/solutions/sdn/.

[6] *Corsa*. Accessed: 2016-05-25. URL: http://www.corsa.com/.

[7] ONF. "OpenFlow Switch Specification 1.5.0." In: *OpenFlow - Open Networking Foundation* (2013).

[8] Nabil Damouny and John Harcourt. *OpenFlow-Enabled Hybrid Cloud Services Connect Enterprise and Service Provider Data Centers*. en-gb. Ed. by Mitch Auster. ONF Solution Brief. accessed: 2016-05-25. 2012. URL: https://www.opennetworking.org/sdn-resources/technical-library#pub.

[9] Suresh Katukam et al. *SDN in the Campus Environment*. en-gb. Ed. by Marc LeClerc. ONF Solution Brief. accessed: 2016-05-25. 2013. URL: https://www.opennetworking.org/sdn-resources/technical-library#pub.

[10] Nick McKeown et al. “OpenFlow: enabling innovation in campus networks.” In: *ACM SIGCOMM Computer Communication Review* 38.2 (2008), pp. 69–74.

[11] David Allan et al. *OpenFlow-enabled SDN and Network Functions Virtualization*. en-gb. Ed. by Michael Zimmerman. ONF Solution Brief. accessed: 2016-05-25. 2014. URL: https://www.opennetworking.org/sdn-resources/technical-library#pub.

[12] D. Awduche et al. *Overview and Principles of Internet Traffic Engineering*. RFC 3272. RFC Editor, 2002.

[13] D. Awduche et al. *Requirements for Traffic Engineering Over MPLS*. RFC 2702. RFC Editor, 1999.

[14] AT&T. *AT&T Research areas*. Tech. rep. accessed: 2016-05-25. 2016. URL: http://www.research.att.com/evergreen/what_we_do/research.html.

[15] *Internet2*. accessed: 2017-05-14. URL: http://www.internet2.edu/.

[16] *ESnet*. Accessed: 2017-05-14. URL: http://www.es.net/.

[17] *Canarie*. accessed: 2017-05-14. URL: https://www.canarie.ca/network/.

[18] *Kreonet*. Accessed: 2017-05-15. URL: http://www.nisn.re.kr/eng/action.do?menuId=50030.

[19] *Sinet*. Accessed: 2017-05-15. URL: https://www.sinet.ad.jp/en/top-en.

[20] *RedIRIS*. accessed: 2017-05-14. URL: https://www.rediris.es/.

[21] *Géant*. accessed: 2017-05-14. URL: https://www.geant.org/.

[22] *Géant Services Portfolio*. accessed: 2017-05-14. URL: https://www.geant.org/Services.

[23] *ESnet's OSCARS with FloodLight*. Accessed: 2016-05-25. URL: https://github.com/hsr/oscars-gui.

[24] *AutoBAHN*. Accessed: 2016-05-25. URL: http://geant3.archive.geant.net/service/autobahn/pages/home.aspx.

[25] L. Zuo and M. M. Zhu. "Concurrent Bandwidth Reservation Strategies for Big Data Transfers in High-Performance Networks." In: *IEEE Transactions on Network and Service Management* 12.2 (2015), pp. 232–247. ISSN: 1932-4537. DOI: 10.1109/TNSM.2015.2430358.

[26] L. Zuo, M. Khaleel, M. M. Zhu, and C. Q. Wu. "On Fixed-Path Variable-Bandwidth Scheduling in High-Performance Networks." In: *2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing*. 2013, pp. 23–30. DOI: 10.1109/GreenCom-iThings-CPSCom.2013.30.

[27] T. Orawiwattanakul, H. Otsuki, E. Kawai, and S. Shimojo. "Dynamic Time Scheduling in Advance Bandwidth Reservation." In: *2012 26th International Conference on Advanced Information Networking and Applications Workshops*. 2012, pp. 885–890. DOI: 10.1109/WAINA.2012.26.

[28] L. Zuo and M. M. Zhu. "Improved Scheduling Algorithms for Single-Path Multiple Bandwidth Reservation Requests." In: *2016 IEEE Trustcom/BigDataSE/ISPA*. 2016, pp. 1692–1699. DOI: 10.1109/TrustCom.2016.0259.

[29] Lina Battestilli et al. "EnLIGHTened computing: An architecture for co-allocating network, compute, and other grid resources for high-end applications." In: *High Capacity Optical Networks and Enabling Technologies, 2007. HONET 2007. International Symposium on*. IEEE. 2007, pp. 1–8.

[30] Michiaki Hayashi, Hideaki Tanaka, and Masatoshi Suzuki. "Advance reservation-based network resource manger for optical networks." In: *Optical Fiber communication/National Fiber Optic Engineers Conference, 2008. OFC/NFOEC 2008. Conference on*. IEEE. 2008, pp. 1–3.

[31] Atsuko Takefusa et al. “G-lambda: Coordination of a grid scheduler and lambda path service over GMPLS.” In: *Future Generation Computer Systems* 22.8 (2006), pp. 868–875.

[32] Alexander Willner et al. “Harmony-advance reservations in heterogeneous multi-domain environments.” In: *NETWORKING 2009* (2009), pp. 871–882.

[33] Sergi Figuerola et al. “PHOSPHORUS: Single-step on-demand services across multi-domain networks for e-science.” In: *Asia-Pacific Optical Communications*. International Society for Optics and Photonics. 2007, pp. 67842X–67842X.

[34] Klara Nahrstedt and Ralf Steinmetz. “Resource management in networked multimedia systems.” In: *Ieee Computer* 28.5 (1995), pp. 52–63.

[35] Anindo Banerjea et al. “The Tenet real-time protocol suite: Design, implementation, and experiences.” In: *IEEE/ACM Transactions on networking* 4.1 (1996), pp. 1–10.

[36] James Roberts and Keqiang Liao. “Traffic models for telecommunication services with advance capacity reservation.” In: *Computer Networks and ISDN Systems* 10.3-4 (1985), pp. 221–229.

[37] Thomas DC Little and Dinesh Venkatesh. “Prospects for interactive video-on-demand.” In: *IEEE multimedia* 1.3 (1994), pp. 14–24.

[38] D. Wischik and A. Greenberg. “Admission control for booking ahead shared resources.” In: *INFOCOM ’98. Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*. Vol. 2. 1998, 873–882 vol.2. DOI: 10.1109/INFCOM.1998.665112.

[39] Xiangfei Zhu and Malathi Veeraraghavan. “Analysis and design of book-ahead bandwidth-sharing mechanisms.” In: *IEEE Transactions on Communications* 56.12 (2008).

[40] Mikael Degermark, Torsten Köhler, Stephen Pink, and Olov Schelen. “Advance reservations for predictive service.” In: *Network and Operating Systems Support for Digital Audio and Video*. Springer. 1995, pp. 1–15.

[41] Jun Zheng and Hussein T Mouftah. "Routing and wavelength assignment for advance reservation in wavelength-routed WDM optical networks." In: *Communications, 2002. ICC 2002. IEEE International Conference on*. Vol. 5. IEEE. 2002, pp. 2722–2726.

[42] Savera Tanwir, Lina Battestilli, Harry Perros, and Gigi Karmous-Edwards. "Dynamic scheduling of network resources with advance reservations in optical grids." In: *International Journal of Network Management* 18.2 (2008), pp. 79–105.

[43] Gerd E Keiser. "A review of WDM technology and applications." In: *Optical Fiber Technology* 5.1 (1999), pp. 3–39.

[44] Rajiv Ramaswami and Kumar N Sivarajan. "Routing and wavelength assignment in all-optical networks." In: *IEEE/ACM Transactions on Networking (TON)* 3.5 (1995), pp. 489–500.

[45] Neal Charbonneau and Vinod M Vokkarane. "A survey of advance reservation routing and wavelength assignment in wavelength-routed WDM networks." In: *IEEE Communications Surveys & Tutorials* 14.4 (2012), pp. 1037–1064.

[46] S. Sahni et al. "Bandwidth Scheduling and Path Computation Algorithms for Connection-Oriented Networks." In: *Networking, 2007. ICN '07. Sixth International Conference on*. 2007, pp. 47–47. DOI: 10.1109/ICN.2007.27.

[47] Antonio Sánchez-Monge and Krzysztof Grzegorz Szarkowicz. *MPLS in the SDN Era*. O'Reilly Media, Inc., 2015.

[48] Michal Pióro and Deepankar Medhi. *Routing, flow, and capacity design in communication and computer networks*. 1st ed. Elsevier, 2004. ISBN: 9780125571890.

[49] Martin Suchara et al. "Network Architecture for Joint Failure Recovery and Traffic Engineering." In: *Proceedings of the ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*. New York, NY, USA: ACM, 2011, pp. 97–108. DOI: 10.1145/1993744.1993756.

[50] Andreas Kassler et al. “Towards QoE-driven multimedia service negotiation and path optimization with Software Defined Nnetworking.” In: *Proceedings of the 20th International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*. IEEE. 2012, pp. 1–5.

[51] Sushant Jain et al. “B4: Experience with a Globally-deployed Software Defined WAN.” In: *Proceedings of the 2013 conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*. New York, NY, USA, 2013, pp. 3–14. DOI: 10.1145/2486001.2486019.

[52] Siddhartha Sen, David Shue, Sunghwan Ihm, and Michael J. Freedman. “Scalable, Optimal Flow Routing in Datacenters via Local Link Balancing.” In: *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies (CoNEXT)*. New York, NY, USA: ACM, 2013, pp. 151–162. DOI: 10.1145/2535372.2535397.

[53] Zhanfeng Jia and P. Varaiya. “Heuristic methods for delay constrained least cost routing using kappa;-shortest-paths.” In: *IEEE Transactions on Automatic Control* 51.4 (2006), pp. 707–712. ISSN: 0018-9286. DOI: 10.1109/TAC.2006.872827.

[54] S. Rai, B. Mukherjee, and O. Deshpande. “IP resilience within an autonomous system: current approaches, challenges, and future directions.” In: *IEEE Communications Magazine* 43.10 (2005), pp. 142–149. ISSN: 0163-6804. DOI: 10.1109/MCOM.2005.1522138.

[55] V. Foteinos et al. “Operator-Friendly Traffic Engineering in IP/MPLS Core Networks.” In: *IEEE Transactions on Network and Service Management* 11.3 (2014), pp. 333–349.

[56] Sergiu Nedevschi et al. “Reducing Network Energy Consumption via Sleeping and Rate-Adaptation.” In: *NSDI*. Vol. 8. 2008, pp. 323–336.

[57] ETSI. *Quality of Experience (QoE) requirements for real-time communication services*. Tech. rep. TR 102 643 V1. 2009.

[58] I. Foster et al. “A distributed resource management architecture that supports advance reservations and co-allocation.” In: *IWQoS ’99. Seventh International Workshop on Quality of Service*. 1999, pp. 27–36. DOI: 10.1109/IWQOS.1999.766475.

[59] Daniel O Awduche. “MPLS and traffic engineering in IP networks.” In: *IEEE Communications Magazine* 37.12 (1999), pp. 42–47.

[60] John McQuillan, Ira Richer, and Eric Rosen. “The new routing algorithm for the ARPANET.” In: *IEEE Transactions on Communications* 28.5 (1980), pp. 711–719.

[61] R. Cole, D. Shur, and C. Villamizar. *IP over ATM: A Framework Document*. RFC 1932. RFC Editor, 1996.

[62] I. Castineyra, N. Chiappa, and M. Steenstrup. *The Nimrod Routing Architecture*. RFC 1992. RFC Editor, 1996.

[63] Kathleen Nichols, Steven Blake, Fred Baker, and David L. Black. *Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers*. RFC 2474. RFC Editor, 1998.

[64] Christian E Hopps. *Analysis of an equal-cost multi-path algorithm*. Tech. rep. 2000.

[65] E. Rosen, A. Viswanathan, and R. Callon. *Multiprotocol Label Switching Architecture*. RFC 3031. RFC Editor, 2001. URL: http://www.rfc-editor.org/rfc/rfc3031.txt.

[66] A. Farrel, J.-P. Vasseur, and J. Ash. *A Path Computation Element (PCE)-Based Architecture*. RFC 4655. RFC Editor, 2006. URL: http://www.rfc-editor.org/rfc/rfc4655.txt.

[67] Takehiro Tsuritani, Masanori Miyazawa, Shuntaro Kashihara, and Tomohiro Otani. “Optical Path Computation Element interworking with Network Management System for Transparent Mesh Networks.” In: *Proceedings of the National Fiber Optic Engineers Conference (NFOEC)*. Optical Society of America. 2008, NWF5.

[68] Ka-Cheong Leung, Victor O. K. Li, and Daiqin Yang. “An Overview of Packet Reordering in Transmission Control Protocol (TCP): Problems, Solutions, and Challenges.” In: *IEEE Transac-*

tions on Parallel and Distributed Systems 18.4 (2007), pp. 522–535. DOI: 10.1109/TPDS.2007.1011.

[69] Mohammad Alizadeh et al. "CONGA: Distributed congestion-aware load balancing for datacenters." In: *ACM SIGCOMM Computer Communication Review*. Vol. 44. 4. ACM. 2014, pp. 503–514.

[70] Abhinav Pathak et al. "Latency inflation with MPLS-based traffic engineering." In: *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement (IMC)*. ACM. 2011, pp. 463–472.

[71] Nick Feamster, Jennifer Rexford, and Ellen Zegura. "The road to SDN: an intellectual history of programmable networks." In: *ACM SIGCOMM Computer Communication Review* 44.2 (2014), pp. 87–98.

[72] Andrew T Campbell, Irene Katzela, Kazuho Miki, and John Vicente. "Open signaling for ATM, internet and mobile networks (OPENSIG'98)." In: *ACM SIGCOMM Computer Communication Review* 29.1 (1999), pp. 97–108.

[73] David L Tennenhouse and David J Wetherall. "Towards an active network architecture." In: *Proceedings of the 2002 DARPA Active NEtworks Conference and Exposition (DANCE)*. IEEE. 2002, pp. 2–15.

[74] Jacobus E van der Merwe and Ian M Leslie. "Switchlets and dynamic virtual ATM networks." In: *Proceedings of the Fifth IFIP/IEEE International Symposium on Integrated Network Management*. Springer. 1997, pp. 355–368.

[75] Matthew Caesar et al. "Design and implementation of a routing control platform." In: *Proceedings of the 2nd Symposium on Networked Systems Design & Implementation (NSDI)*. USENIX Association. 2005, pp. 15–28.

[76] Jennifer Rexford et al. "Network-wide decision making: Toward a wafer-thin control plane." In: *Proceedings of the Third Workshop on Hot Topics in Networks (HotNets)*. 2004, pp. 59–64.

[77] *Open Networking Foundation*. Accessed: 2016-05-25. URL: https://www.opennetworking.org/.

[78] ONF. *Software-Defined Networking: The New Norm for Networks*. 2012.

[79] *SDNRG*. https://irtf.org/sdnrg. Accessed: 2014-06-24.

[80] E. Haleplidis et al. *Software-Defined Networking (SDN): Layers and Architecture Terminology*. RFC 7426. http://www.rfc-editor.org/rfc/rfc7426.txt. RFC Editor, 2015. URL: http://www.rfc-editor.org/rfc/rfc7426.txt.

[81] ONF Market Education Committee et al. "Software-Defined Networking: Arquitecture Overview." In: *ONF White Paper. Palo Alto, US: Open Networking Foundation* (2013).

[82] Xuezhi Jiang, Mingwei Xu, and Qi Li. "Compact Route Computation: Improving Parallel BGP Route Processing for Scalable Routers." In: *Proceedings of the 2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW)*. IEEE. 2011, pp. 1496–1501.

[83] Sugam Agarwal, Murali Kodialam, and TV Lakshman. "Traffic engineering in software defined networks." In: *Proceedings of the 32nd IEEE International Conference on Computer Communications (INFOCOM)*. IEEE. 2013, pp. 2211–2219.

[84] Pedro S Pisa et al. "Openflow and xen-based virtual network migration." In: *Communications: Wireless in Developing Countries and Networks of the Future*. Springer, 2010, pp. 170–181.

[85] P. Garg and Y. Wang. *Network Virtualization Using Generic Routing Encapsulation*. RFC 7637. RFC Editor, 2015.

[86] M. Mahalingam et al. *Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks*. RFC 7637. RFC Editor, 2014.

[87] A. Mendiola, J. Astorga, E. Jacob, and M. Higuero. "A Survey on the Contributions of Software-Defined Networking to Traffic Engineering." In: *IEEE Communications Surveys Tutorials* 19.2 (2017), pp. 918–953. ISSN: 1553-877X. DOI: 10.1109/COMST.2016.2633579.

[88] A. Farrel, J.-P. Vasseur, and J. Ash. *A Path Computation Element (PCE)-Based Architecture*. RFC 4655. RFC Editor, 2006. URL: http://www.rfc-editor.org/rfc/rfc4655.txt.

[89] *ForCES Working Group*. Accessed: 2016-05-25. URL: http://datatracker.ietf.org/wg/forces/charter/.

[90] Evangelos Haleplidis et al. "Software-Defined Networking: Experimenting with the control to forwarding plane interface." In: *Proceedings of the 2012 European Workshop on Software Defined Networking (EWSDN)*. IEEE. 2012, pp. 91–96.

[91] L. Yang, R. Dantu, T. Anderson, and R. Gopal. *Forwarding and Control Element Separation (ForCES) Framework*. RFC 3746. RFC Editor, 2004.

[92] A. Crouch et al. *Forwarding and Control Element Separation (ForCES) Applicability Statement*. RFC 6041. RFC Editor, 2010.

[93] E. Haleplidis, K. Ogawa, W. Wang, and J. Hadi Salim. *Implementation Report for Forwarding and Control Element Separation (ForCES)*. RFC 6053. RFC Editor, 2010.

[94] J. Halpern and J. Hadi Salim. *Forwarding and Control Element Separation (ForCES) Forwarding Element Model*. RFC 5812. RFC Editor, 2010.

[95] Evangelos Haleplidis et al. "Network programmability with ForCES." In: *IEEE Communications Surveys & Tutorials* 17.3 (2015), pp. 1423–1440.

[96] *NOX*. Accessed: 2016-05-25. URL: http://www.noxrepo.org/nox/about-nox/.

[97] *POX*. Accessed: 2016-05-25. URL: http://www.noxrepo.org/pox/about-pox/.

[98] *Trema*. Accessed: 2016-05-25. URL: http://trema.github.io/trema/.

[99] *Ryu*. Accessed: 2016-05-25. URL: http://osrg.github.io/ryu/.

[100] *FloodLight*. Accessed: 2016-05-25. URL: http://www.projectfloodlight.org/floodlight/.

[101] *Beacon*. Accessed: 2016-05-25. URL: https://openflow.stanford.edu/display/Beacon/Home.

[102] Zheng Cai, Alan L Cox, and TS Eugene Ng Maestro. *A system for scalable OpenFlow control*. Tech. rep. Technical Report TR10-08, Rice University, 2010.

[103] Andreas Voellmy, Bryan Ford, Paul Hudak, and Y Richard Yang. "Scaling Software-defined Network Controllers on Multicore Servers." In: *YaleCS TR1468* (2012).

[104] *Jaxon*. Accessed: 2016-05-25. URL: http://jaxon.onuos.org/.

[105] *SNAC*. Accessed: 2016-05-25. URL: http://www.openflowhub.org/display/Snac/SNAC+Home.

[106] Teemu Koponen et al. "Onix: A Distributed Control Platform for Large-scale Production Networks." In: *Proceedings of the 9th USENIX conference on Operating systems design and implementation (OSDI)*. Vol. 10. 2010, pp. 1–6.

[107] Amin Tootoonchian and Yashar Ganjali. "Hyperflow: a distributed control plane for openflow." In: *Proceedings of the 2010 Internet Network Management Workshop/Workshop on Research on Enterprise Networking (INM/WREN)*. USENIX Association. 2010, pp. 3–3.

[108] *Helios*. Accessed: 2016-05-25. URL: http://www.nec.com/.

[109] Rob Sherwood et al. "FlowVisor: A Network Virtualization Layer." In: (2009). accessed: 2016-05-25. URL: http://OpenFlowSwitch.org/downloads/technicalreports/openflow-tr-2009-1-flowvisor.pdf.

[110] *OpenVirteX*. accessed: URL: http://tools.onlab.us/ovx.html.

[111] Zdravko Bozakov and Panagiotis Papadimitriou. "Autoslice: automated and scalable slicing for software-defined networks." In: *Proceedings of the 2012 ACM conference on CoNEXT student workshop*. ACM. 2012, pp. 3–4.

[112] M. F. Bari et al. “Data Center Network Virtualization: A Survey.” In: *IEEE Communications Surveys & Tutorials* 15.2 (2013), pp. 909–928. ISSN: 1553-877X. DOI: 10.1109/SURV.2012.090512.00043.

[113] Piotr Rygielski and Samuel Kounev. “Network virtualization for QoS-aware resource management in cloud data centers: a survey.” In: *PIK-Praxis der Informationsverarbeitung und Kommunikation* 36.1 (2013), pp. 55–64.

[114] Andreas Blenk, Arsany Basta, Martin Reisslein, and Wolfgang Kellerer. “Survey on network virtualization hypervisors for software defined networking.” In: *IEEE Communications Surveys & Tutorials* 18.1 (2016), pp. 655–685.

[115] *Open Network Operating System*. accessed: 2016-05-25. URL: http://www.onosproject.org/.

[116] *Open Daylight Project*. accessed: 2016-05-25. URL: http://www.opendaylight.org/.

[117] *Cisco ONE*. accessed: 2016-05-25. URL: http://www.cisco.com/web/solutions/trends/open_network_environment/indepth.html.

[118] Alexander Shalimov et al. “Advanced study of SDN/OpenFlow controllers.” In: *Proceedings of the 9th Central & Eastern European Software Engineering Conference in Russia (CEE-SECR)*. ACM. 2013, p. 1.

[119] *IDR Working Group*. Accessed: 2016-05-25. URL: http://datatracker.ietf.org/wg/idr/charter/.

[120] H. Gredler et al. *North-Bound Distribution of Link-State and Traffic Engineering (TE) Information Using BGP*. RFC 7752. RFC Editor, 2016.

[121] Y. Rekhter, T. Li, and S. Hares. *A Border Gateway Protocol 4 (BGP-4)*. RFC 4271. RFC Editor, 2006.

[122] *Cisco ONE*. Accessed: 2016-05-25. URL: http://www.cisco.com/web/solutions/trends/open_network_environment/indepth.html.

[123] J. Ash and J.L. Le Roux. *Path Computation Element (PCE) Communication Protocol Generic Requirements*. RFC 4657. RFC Editor, 2006.

[124] D. Awduche et al. *RSVP-TE: Extensions to RSVP for LSP Tunnels*. RFC 3209. RFC Editor, 2001.

[125] Ramon Casellas et al. "IDEALIST control plane architecture for multi-domain flexi-grid optical networks." In: *Proceedings of the 2014 European Conference on Networks and Communications (EuCNC)*. IEEE. 2014, pp. 1–5.

[126] Dan-Dan Wang, Li-Gang Dong, Fu-Rong Zhou, and Wei-Ming Wang. "Study of OSPF Routing Optimization among Forwarding Elements in the ForCES Router." In: *Information Technology Journal* 12.2 (2013), pp. 351–356.

[127] Marc Koerner and Odej Kao. "Multiple service load-balancing with OpenFlow." In: *Proceedings of the 13th IEEE International Conference on High Performance Switching and Routing (HPSR)*. IEEE. 2012, pp. 210–214.

[128] Zhuge Bin et al. "Resource scheduling algorithm and ecnomic model in ForCES networks." In: *China Communications* 11.3 (2014), pp. 91–103.

[129] Chi-Yao Hong et al. "Achieving high utilization with software-driven WAN." In: *Proceedings of the 2013 conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*. Vol. 43. 4. ACM. 2013, pp. 15–26.

[130] C. P. Guok et al. "A User Driven Dynamic Circuit Network Implementation." In: *2008 IEEE Globecom Workshops*. 2008, pp. 1–5. DOI: 10.1109/GLOCOMW.2008.ECP.14.

[131] Mark Boddie et al. "On extending ESnet's OSCARS with a multi-domain anycast service." In: *Proceedings of the 16th International Conference on Optical Network Design and Modeling (ONDM)*. IEEE. 2012, pp. 1–6.

[132] Edsger W Dijkstra. "A note on two problems in connexion with graphs." In: *Numerische mathematik* 1.1 (1959), pp. 269–271.

[133] *OLiMPS's OSCARS with Floodlight*. URL: https://github.com/mbredel/floodlight-olimps.

[134] Saurav Das, Ali Sharafat, Guru Parulkar, and Nick McKeown. "MPLS with a simple OPEN control plane." In: *Proceedings of the 2011 Optical Fiber Communication Conference (OFC)*. Optical Society of America. 2011, OWP2.

[135] Saurav Das et al. "Application-aware aggregation and traffic engineering in a converged packet-circuit network." In: *Proceedings of the 2011 National Fiber Optic Engineers Conference (NFOEC)*. Optical Society of America. 2011, NThD3.

[136] Ariff Premji. "Using MPLS auto-bandwidth in MPLS networks." In: *Wireline and Wireless Carriers, Juniper Networks, Inc* (2005).

[137] M Gharbaoui et al. "On virtualization-aware traffic engineering in OpenFlow Data Centers networks." In: *Proceedings of the 2014 IEEE Network Operations and Management Symposium (NOMS)*. 2014, pp. 1–8.

[138] Davide Adami et al. "Effective resource control strategies using openflow in cloud data center." In: *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*. IEEE. 2013, pp. 568–574.

[139] Ori Gerstel, Masahiko Jinno, Andrew Lord, and SJ Ben Yoo. "Elastic optical networking: A new dawn for the optical layer?" In: *IEEE Communications Magazine* 50.2 (2012).

[140] D. King and A. Farrel. *A PCE-Based Architecture for Application-Based Network Operations*. RFC 7491. RFC Editor, 2015.

[141] Ioan Turus, Josva Kleist, and Anna Manolova Fagertun. "Evaluation of flex-grid architecture for nren optical networks." In: *Proceedings of Tnc2014* (2014).

[142] Ramon Casellas et al. "First Multi-partner Demonstration of BGP-LS enabled Inter-domain EON control with H-PCE." In: *Proceedings of the Optical Fiber Communication Conference (OFC)*. Optical Society of America. 2015, Th1A–4.

[143] Ramon Casellas et al. “SDN Orchestration of OpenFlow and GMPLS Flexi-Grid Networks With a Stateful Hierarchical PCE [Invited].” In: *Journal of Optical Communications and Networking* 7.1 (2015), A106–A117.

[144] Marta Cuaresma et al. “Experimental demonstration of H-PCE with BPG-LS in elastic optical networks.” In: *Proceedings of the 39th European Conference and Exhibition on Optical Communication (ECOC)*. 2013, pp. 1–3.

[145] Ricardo Martínez et al. “Experimental validation of active frontend—Backend stateful PCE operations in flexgrid optical network re-optimization.” In: *Proceedings of the 2014 European Conference on Optical Communication (ECOC)*. IEEE. 2014, pp. 1–3.

[146] B. Fortz and M. Thorup. “Internet traffic engineering by optimizing OSPF weights.” In: *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No.00CH37064)*. Vol. 2. 2000, 519–528 vol.2. DOI: 10.1109/INFCOM.2000.832225.

[147] A. Ghanwani et al. “Traffic engineering standards in IP-networks using MPLS.” In: *IEEE Communications Magazine* 37.12 (1999), pp. 49–53. ISSN: 0163-6804. DOI: 10.1109/35.809384.

[148] R. Zhang-Shen and N. McKeown. “Designing a Fault-Tolerant Network Using Valiant Load-Balancing.” In: *IEEE INFOCOM 2008 - The 27th Conference on Computer Communications*. 2008. DOI: 10.1109/INFOCOM.2008.305.

[149] E. Oki and A. Iwaki. “Load-Balanced IP Routing Scheme Based on Shortest Paths in Hose Model.” In: *IEEE Transactions on Communications* 58.7 (2010), pp. 2088–2096. ISSN: 0090-6778. DOI: 10.1109/TCOMM.2010.07.090219.

[150] Wolfgang Braun and Michael Menth. “Load-dependent flow splitting for traffic engineering in resilient OpenFlow networks.” In: *Proceedings of the 2015 International Conference and Workshops on Networked Systems (NetSys)*. 2015, pp. 1–5.

[151] S. Jouet, C. Perkins, and D. Pezaros. "OTCP: SDN-managed congestion control for data center networks." In: *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*. 2016, pp. 171–179. DOI: 10.1109/NOMS.2016.7502810.

[152] S. Kim, J. Son, A. Talukder, and C. S. Hong. "Congestion prevention mechanism based on Q-leaning for efficient routing in SDN." In: *2016 International Conference on Information Networking (ICOIN)*. 2016, pp. 124–128. DOI: 10.1109/ICOIN.2016.7427100.

[153] John Bresnahan et al. "Globus GridFTP: what's new in 2007." In: *Proceedings of the first international conference on Networks for grid applications*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering). 2007, p. 19.

[154] Che Huang, Chawanat Nakasan, Kohei Ichikawa, and Hajimu Iida. "A multipath controller for accelerating GridFTP transfer over SDN." In: *Proceedings of the 11th IEEE International Conference on e-Science (e-Science)*. 2015, pp. 439–447.

[155] Suoheng Li et al. "Flexible Traffic Engineering: When OpenFlow Meets Multi-Protocol IP-Forwarding." In: *IEEE Communications Letters* 18.10 (2014), pp. 1699–1702.

[156] Fung Po Tso and D Pezaros. "Baatdaat: Measurement-Based Flow Scheduling for Cloud Data Centers." In: *Proceedings of the 2013 IEEE Symposium on Computers and Communications (ISCC)*. 2013, pp. 765–770. DOI: 10.1109/ISCC.2013.6755041.

[157] Ramona Trestian, Gabriel-Miro Muntean, and Kostas Katrinis. "MiceTrap: Scalable traffic engineering of datacenter mice flows using OpenFlow." In: *Proceedings of the 2013 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. IEEE. 2013, pp. 904–907.

[158] Di Zhong, Weiming Wang, and Chuanhuang Li. "The implementation of multiple virtual FEs and their backup in the ForCES Router." In: *Proceedings of the 2nd International Workshop on Intelligent Systems and Applications (ISA)*. IEEE. 2010, pp. 1–4.

[159] Kien Nguyen, Quang Tran Minh, and Shigeki Yamada. "A Software-Defined Networking approach for Disaster-Resilient WANs." In: *Proceedings of the 22nd International Conference on Computer Communications and Networks (ICCCN)*. IEEE. 2013, pp. 1–5.

[160] S. Sharma et al. "Enabling fast failure recovery in OpenFlow networks." In: *2011 8th International Workshop on the Design of Reliable Communication Networks (DRCN)*. 2011, pp. 164–171. DOI: 10.1109/DRCN.2011.6076899.

[161] D. Staessens et al. "Software defined networking: Meeting carrier grade requirements." In: *2011 18th IEEE Workshop on Local Metropolitan Area Networks (LANMAN)*. 2011, pp. 1–6.

[162] S. S. W. Lee et al. "Software-based fast failure recovery for resilient OpenFlow networks." In: *2015 7th International Workshop on Reliable Networks Design and Modeling (RNDM)*. 2015, pp. 194–200. DOI: 10.1109/RNDM.2015.7325229.

[163] S. Sharma et al. "Fast failure recovery for in-band OpenFlow networks." In: *2013 9th International Conference on the Design of Reliable Communication Networks (DRCN)*. 2013, pp. 52–59.

[164] N. L. M. v. Adrichem, B. J. v. Asten, and F. A. Kuipers. "Fast Recovery in Software-Defined Networks." In: *2014 Third European Workshop on Software Defined Networks*. 2014, pp. 61–66. DOI: 10.1109/EWSDN.2014.13.

[165] K. Jeong, J. Kim, and Y. T. Kim. "QoS-aware Network Operating System for software defined networking with Generalized OpenFlows." In: *2012 IEEE Network Operations and Management Symposium*. 2012, pp. 1167–1174. DOI: 10.1109/NOMS.2012.6212044.

[166] Evangelos Haleplidis et al. "Network programmability with ForCES." In: *IEEE Communications Surveys & Tutorials* 17.3 (2015), pp. 1423–1440.

[167] Kwang-Tae Jeong, Chang-Hwan Lee, and Young-Tak Kim. "Performance analysis of multi-layered protection switching with control-forwarding separation." In: *Proceedings of the 14th*

Asia-Pacific Network Operations and Management Symposium (APNOMS). IEEE. 2012, pp. 1–4.

[168] M. Shand and S. Bryant. *IP Fast Reroute Framework*. RFC 5714. RFC Editor, 2010.

[169] Seung-Hun Yoon, Djakhongir Siradjev, and Young-Tak Kim. "Management of DiffServ-over-MPLS transit networks with BFD/OAM in ForCES architecture." In: *Large Scale Management of Distributed Systems*. Springer, 2006, pp. 136–148.

[170] Kevin Phemius and Mathieu Bouet. "Implementing OpenFlow-based resilient network services." In: *Proceedings of the 1st IEEE International Conference on Cloud Networking (CLOUDNET)*. IEEE. 2012, pp. 212–214.

[171] H. Nam, K. H. Kim, J. Y. Kim, and H. Schulzrinne. "Towards QoE-aware video streaming using SDN." In: *2014 IEEE Global Communications Conference*. 2014, pp. 1317–1322. DOI: 10.1109/GLOCOM.2014.7036990.

[172] E. Liotou et al. "An SDN QoE-service for dynamically enhancing the performance of OTT applications." In: *2015 Seventh International Workshop on Quality of Multimedia Experience (QoMEX)*. 2015, pp. 1–2. DOI: 10.1109/QoMEX.2015.7148106.

[173] A. Farshad et al. "Leveraging SDN to provide an in-network QoE measurement framework." In: *2015 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. 2015, pp. 239–244. DOI: 10.1109/INFCOMW.2015.7179391.

[174] Peter Dely, Andreas Kassler, and Nico Bayer. "Openflow for wireless mesh networks." In: *Computer Communications and Networks (ICCCN), 2011 Proceedings of 20th International Conference on*. IEEE. 2011, pp. 1–6.

[175] Damon Wischik and Albert Greenberg. "Admission control for booking ahead shared resources." In: *INFOCOM'98. Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*. Vol. 2. IEEE. 1998, pp. 873–882.

[176] J. Gehr and J. Schneider. “Measuring Fragmentation of Two-Dimensional Resources Applied to Advance Reservation Grid Scheduling.” In: *2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*. 2009, pp. 276–283. DOI: 10.1109/CCGRID.2009.81.

[177] E. Soltanaghaei and M. Veeraraghavan. “Multi-option, multi-class path scheduling methods for advance reservation systems.” In: *2015 IEEE 16th International Conference on High Performance Switching and Routing (HPSR)*. 2015, pp. 1–8. DOI: 10.1109/HPSR.2015.7483080.

[178] Y. Wang, C. Q. Wu, and A. Hou. “Periodic Scheduling of Deadline-Constrained Bandwidth Reservations for Scientific Collaboration.” In: *2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*. 2016, pp. 150–157. DOI: 10.1109/HPCC-SmartCity-DSS.2016.0032.

[179] P. Dharam, C. Q. Wu, and N. S. V. Rao. “Advance Bandwidth Scheduling in Software-Defined Networks.” In: *2015 IEEE Global Communications Conference (GLOBECOM)*. 2015, pp. 1–6. DOI: 10.1109/GLOCOM.2015.7416950.

[180] L-O Burchard. “Analysis of data structures for admission control of advance reservation requests.” In: *IEEE Transactions on Knowledge and Data Engineering* 17.3 (2005), pp. 413–424.

[181] Jun Zheng and H. T. Mouftah. “Routing and wavelength assignment for advance reservation in wavelength-routed WDM optical networks.” In: *2002 IEEE International Conference on Communications. Conference Proceedings. ICC 2002 (Cat. No.02CH37333)*. Vol. 5. 2002, 2722–2726 vol.5. DOI: 10.1109/ICC.2002.997338.

[182] Pragatheeswaran Angu and Byrav Ramamurthy. “Continuous and parallel optimization of dynamic bandwidth scheduling in WDM networks.” In: *Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE*. IEEE. 2010, pp. 1–6.

[183] Chongyang Xie and Nasir Ghani. "A prioritized routing algorithm for multi-domain optical networks supporting advance reservation." In: *High Capacity Optical Networks and Enabling Technologies, 2008. HONET 2008. International Symposium on*. IEEE. 2008, pp. 211–215.

[184] T. D. Wallace and A. Shami. "Connection management algorithm for advance lightpath reservation in WDM networks." In: *Broadband Communications, Networks and Systems, 2007. BROADNETS 2007. Fourth International Conference on*. 2007, pp. 837–844. DOI: 10.1109/BROADNETS.2007.4550520.

[185] Eun-Sung Jung, Yan Li, Sanjay Ranka, and Sartaj Sahni. "Performance evaluation of routing and wavelength assignment algorithms for optical networks." In: *2008 IEEE Symposium on Computers and Communications*. 2008, pp. 62–67. DOI: 10.1109/ISCC.2008.4625738.

[186] J. Kuri, N. Puech, M. Gagnaire, and E. Dotaro. "Routing foreseeable lightpath demands using a tabu search meta-heuristic." In: *Global Telecommunications Conference, 2002. GLOBECOM '02. IEEE*. Vol. 3. 2002, 2803–2807 vol.3. DOI: 10.1109/GLOCOM.2002.1189140.

[187] L. O. Burchard, H. U. Heiss, and C. A. F. De Rose. "Performance issues of bandwidth reservations for grid computing." In: *Proceedings. 15th Symposium on Computer Architecture and High Performance Computing*. 2003, pp. 82–90. DOI: 10.1109/CAHPC.2003.1250324.

[188] Ying Chen, Arunita Jaekel, and Ataul Bari. "Resource allocation strategies for a non-continuous sliding window traffic model in WDM networks." In: *Broadband Communications, Networks, and Systems, 2009. BROADNETS 2009. Sixth International Conference on*. IEEE. 2009, pp. 1–7.

[189] R Sedgewick and KD Wayne. *Algorithms. 4th*. 2011.

[190] Robert Endre Tarjan. *Data structures and network algorithms*. SIAM, 1983.

[191] Paul E Black. *Dictionary of algorithms and data structures.* National Institute of Standards and Technology Gaithersburg, 2004.

[192] Emmanouel A Varvarigos and Vishal Sharma. "An efficient reservation connection control protocol for gigabit networks." In: *Computer Networks and ISDN Systems* 30.12 (1998), pp. 1135–1156.

[193] Emmanouel Manos Varvarigos, Vasileios Sourlas, and Konstantinos Christodoulopoulos. "Routing and scheduling connections in networks that support advance reservations." In: *Computer Networks* 52.15 (2008), pp. 2988–3006.

[194] Olov Schelén, Andreas Nilsson, Joakim Norrgard, and Stephen Pink. "Performance of QoS agents for provisioning network resources." In: *Quality of Service, 1999. IWQoS'99. 1999 Seventh International Workshop on.* IEEE. 1999, pp. 17–26.

[195] Olov Schelén and Stephen Pink. "Resource sharing in advance reservation agents." In: *Journal of High Speed Networks* 7.3, 4 (1998), pp. 213–228.

[196] Roch A Guérin and Ariel Orda. "Networks with advance reservations: The routing perspective." In: *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE.* Vol. 1. IEEE. 2000, pp. 118–127.

[197] L-O Burchard, Barry Linnert, and Joerg Schneider. "Rerouting strategies for networks with advance reservations." In: *e-Science and Grid Computing, 2005. First International Conference on.* IEEE. 2005, 8–pp.

[198] L. O. Burchard, B. Linnert, and J. Schneider. "A distributed load-based failure recovery mechanism for advance reservation environments." In: *CCGrid 2005. IEEE International Symposium on Cluster Computing and the Grid, 2005.* Vol. 2. 2005, 1071–1078 Vol. 2. DOI: 10.1109/CCGRID.2005.1558679.

[199] Tao Wang and Jianer Chen. "Bandwidth tree-a data structure for routing in networks with advanced reservations." In: *Performance, Computing, and Communications Conference, 2002. 21st IEEE International*. IEEE. 2002, pp. 37–44.

[200] Joseph B Kruskal. "On the shortest spanning subtree of a graph and the traveling salesman problem." In: *Proceedings of the American Mathematical society* 7.1 (1956), pp. 48–50.

[201] Lars Wolf et al. "Issues of reserving resources in advance." In: *Network and Operating Systems Support for Digital Audio and Video*. Springer. 1995, pp. 28–38.

[202] Lars C Wolf and Ralf Steinmetz. "Concepts for resource reservation in advance." In: *Multimedia Tools and Applications* 4.3 (1997), pp. 255–278.

[203] Mugurel Ionut Andreica and Nicolae Tapus. "Efficient data structures for online QoS-constrained data transfer scheduling." In: *Parallel and Distributed Computing, 2008. ISPDC'08. International Symposium on*. IEEE. 2008, pp. 285–292.

[204] Mugurel Ionut Andreica. "Optimal Scheduling of File Transfers with Divisible Sizes on Multiple Disjoint Paths." In: *Proceedings of the IEEE Romania International Conference" Communications", 2008.(ISBN: 978-606-521-008-0)*. 2008, pp. 155–158.

[205] M. I. Andreica and N. Tapus. "Optimal Offline TCP Sender Buffer Management Strategy." In: *2008 International Conference on Communication Theory, Reliability, and Quality of Service*. 2008, pp. 41–46. DOI: 10.1109/CTRQ.2008.11.

[206] J. Schneider and B. Linnert. "Efficiently Managing Advance Reservations Using Lists of Free Blocks." In: *2011 23rd International Symposium on Computer Architecture and High Performance Computing*. 2011, pp. 183–190. DOI: 10.1109/SBAC-PAD.2011.25.

[207] M. Balman, E. Chaniotakisy, A. Shoshani, and A. Sim. "A Flexible Reservation Algorithm for Advance Network Provisioning." In: *2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. 2010, pp. 1–11. DOI: 10.1109/SC.2010.4.

[208] Maryam Barshan, Hendrik Moens, Jeroen Famaey, and Filip De Turck. "Deadline-aware advance reservation scheduling algorithms for media production networks." In: *Computer Communications* 77 (2016), pp. 26–40.

[209] Sahel Sahhaf et al. "Resilient algorithms for advance bandwidth reservation in media production networks." In: *Design of Reliable Communication Networks (DRCN), 2016 12th International Conference on the*. IEEE. 2016, pp. 130–137.

[210] Maryam Barshan, Hendrik Moens, and Bruno Volckaert. "Dynamic adaptive advance bandwidth reservation in media production networks." In: *NetSoft Conference and Workshops (NetSoft), 2016 IEEE*. IEEE. 2016, pp. 58–62.

[211] E. Mannie. *Generalized Multi-Protocol Label Switching (GMPLS) Architecture*. RFC 3945. RFC Editor, 2004.

[212] Lester R Ford and Delbert R Fulkerson. "Maximal flow through a network." In: *Canadian journal of Mathematics* 8.3 (1956), pp. 399–404.

[213] Michael L Fredman and Robert Endre Tarjan. "Fibonacci heaps and their uses in improved network optimization algorithms." In: *Journal of the ACM (JACM)* 34.3 (1987), pp. 596–615.

[214] F. Pakzad, M. Portmann, W. L. Tan, and J. Indulska. "Efficient topology discovery in software defined networks." In: *2014 8th International Conference on Signal Processing and Communication Systems (ICSPCS)*. 2014, pp. 1–8. DOI: 10.1109/ICSPCS.2014.7021050.

[215] D. Katz and D. Ward. *Bidirectional Forwarding Detection (BFD)*. RFC 5880. RFC Editor, 2010.

[216] A. Autenrieth and A. Kirstadter. "Engineering end-to-end IP resilience using resilience-differentiated QoS." In: *IEEE Communications Magazine* 40.1 (2002), pp. 50–57. ISSN: 0163-6804. DOI: 10.1109/35.978049.

[217] Christos Argyropoulos, Dimitrios Kalogeras, Georgios Androulidakis, and Vasilis Maglaris. "PaFloMon–A Slice Aware Passive Flow Monitoring Framework for OpenFlow Enabled Experimental Facilities." In: *Software Defined Networking (EWSDN), 2012 European Workshop on*. IEEE. 2012, pp. 97–102.

[218] Guy Roberts et al. "NSI Connection Service v2. 0." In: *GFD. 212* (2014), pp. 1–119.

[219] Chin Yang Lee. "An algorithm for path connections and its applications." In: *IRE transactions on electronic computers* 3 (1961), pp. 346–365.

[220] ESnet. *OSCARS Circuits*. accessed: 2017-03-15. 2017. URL: https://my.es.net/oscars/list.

[221] Alaitz Mendiola et al. "DynPaC: A Path Computation Framework for SDN." In: *Software Defined Networks (EWSDN), 2015 Fourth European Workshop on*. IEEE. 2015, pp. 119–120.

[222] A. Mendiola et al. *NSI based multi-domain connection provisioning across OpenFlow domains*. IEEE/ACM Conference on Supercoputing. 2014. URL: https://goo.gl/Y0mqxr.

[223] Eduardo Jacob et al. "Multi-domain Software Defined Networking: Exploring possibilities." In: *Selected papers of the Terena Networking Conference* (2014). ISSN: 978-90-77559-24-6.

[224] *Pionier Network*. Accessed: 2017-05-15. URL: http://www.pionier.net.pl/online/en/.

[225] *GÉANT Testbed Service*. Accessed: 2017-05-15. URL: https://www.geant.org/Services/Connectivity_and_network/GTS.

[226] Alaitz Mendiola et al. "Towards an SDN-based bandwidth on demand service for the European research community." In: *Networked Systems (NetSys), 2017 International Conference on*. IEEE. 2017, pp. 1–6.

[227] Jokin Garay et al. "Self-deploying Service Graphs over ELwUD (EHU-OEF Lightweight UNIFY Domain)." In: *Network Softwarization (NetSoft), 2015 1st IEEE Conference on*. IEEE. 2015, pp. 1–2.

[228] J. Astorga et al. “DynPaC: Dynamic and Adaptive Traffic Engineering for SDNs.” In: *Selected papers of the Terena Networking Conference* (2015). ISSN: 978-90-77559-25-3.

[229] Alaitz Mendiola et al. “Multi-domain bandwidth on demand service provisioning using SDN.” In: *2016 IEEE NetSoft Conference and Workshops (NetSoft)*. IEEE. 2016, pp. 353–354.

[230] Alaitz Mendiola et al. “DynPaC: a path computation framework for SDN.” In: *Proceedings of the Fourth European Workshop on Software Defined Networks (EWSDN)*. IEEE. 2015, pp. 119–120.

[231] A. Mendiola et al. *SDN-enabled AutoBAHN. Envisioning future BoD services*. accessed: 2017-05-23. 2016. URL: https://tnc16.geant.org/core/user/809.

[232] A. Mendiola. *Adapting the SDN-based BoD application to the particularities of the Corsa HW datapath*. GÉANT Symposium 2016. 2016.

[233] A. Mendiola, J. Ortiz, J. Aznar, and A. Juszczyk. *Demonstration of the multi-domain BoD application*. GÉANT Symposium 2016. 2016.

[234] A. Mendiola et al. *A solution for connection-oriented multi-domain SDN based on NSI*. accessed: 2017-05-23. 2015. URL: https://tnc15.terena.org/core/user/809.

[235] L. Prete et al. *Empowering GÉANT deployments with ONOS brigades (accepted)*. accessed: 2017-05-23. 2017. URL: https://tnc17.geant.org/core/user/1333.

[236] Pier Luigi Ventre et al. “GÉANT SDX - SDN based Open eXchange Point.” In: *2016 IEEE NetSoft Conference and Workshops (NetSoft)*. IEEE. 2016, pp. 345–346.

[237] Alaitz Mendiola et al. “An architecture for dynamic QoS management at Layer 2 for DOCSIS access networks using OpenFlow.” In: *Computer Networks* 94 (2016), pp. 112–128.

[238] Victor Fuentes et al. “Integrating complex legacy systems under OpenFlow control: The DOCSIS use case.” In: *2014 Third European Workshop on Software Defined Networks*. IEEE. 2014, pp. 37–42.

[239] Jon Matias et al. “FlowNAC: Flow-based network access control.” In: *2014 Third European Workshop on Software Defined Networks*. IEEE. 2014, pp. 79–84.

[240] Jon Matias et al. “The EHU-OEF: An OpenFlow-based Layer-2 experimental facility.” In: *Computer Networks* 63 (2014), pp. 101–127.

[241] Eduardo Jacob et al. “Deploying a virtual network function over a software defined network infrastructure: experiences deploying an access control VNF in the University of Basque Country’s OpenFlow enabled facility.” In: ().

[242] J. Matias, E. Jacob, C. Pinedo, and A. Mendiola. *Access Control NFV enforcement at the EHU-OpenFlow Enabled Facility*. IEEE Global Communications Conference. 2013. URL: https://goo.gl/lPelH5.

[243] J. Matias et al. *Access Control NFV enforcement at the EHU-OpenFlow Enabled Facility*. Jornadas Técnicas de RedIRIS. 2012. URL: https://goo.gl/jBVaYL.

[244] J. Matias et al. “Implementing Layer 2 Network Virtualization Using OpenFlow: Challenges and Solutions.” In: *Proceedings of the 2012 European Workshop on Software Defined Networking (EWSDN)*. 2012, pp. 30–35. DOI: 10.1109/EWSDN.2012.18.