Control plane in dynamic software networks

Adrien Wion

INSTITUT POLYTECHNIQUE DE PARIS

NNT : 2021IPPAT007

Thèse de doctorat

Control Plane in Dynamic Software Networks

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à Télécom Paris

École doctorale n°626 Ecole doctorale de l'Institut Polytechnique de Paris (ED IP Paris)
Spécialité de doctorat : Informatique

ADRIEN WION

Composition du Jury :

Hind Castel Professeure, Télécom Sud Paris	Présidente
Stefano Secci Professeur, CNAM	Rapporteur
Thierry Turletti Directeur de Recherche, INRIA	Rapporteur
Andrea Araldo Maitre de Conférence, Télécom Sud Paris	Examinateur
Gerard Memmi Professeur, Télécom Paris	Directeur de thèse
Mathieu Bouet Directeur de Recherche, Thales	Co-directeur de thèse
Vania Conan Directeur de Recherche, Thales	Invité
Luigi Iannone Maitre de Conférence, Télécom Paris	Invité

Je dédie ce modeste travail à mes grands parents, partis trop tôt.

Remerciements

Au moment de poser les derniers mots sur ce qui fut l'ouvrage de trois années, un étrange sentiment vous habite. A regarder en arrière, on se rend compte du chemin parcouru, surpris d'avoir pu marcher jusque-là. A relire le manuscrit, on réalise qu'une page se tourne, alors qu'on peine à croire être l'auteur de ces lignes. C'est en se remémorant ce long voyage, lorsque trois années de souvenirs refont surface, que l'essentiel vous revient, mettant au jour une erreur fondamentale d'attribution. Ce périple n'a été possible que grâce à une multitude de mains tendues, qui inlassablement, m'ont soutenu, donné, transmis. C'est à toutes ces personnes que je tiens à montrer par ces quelques mots ma plus profonde gratitude. Sans elles, ce travail n'aurait pu aboutir.

Tout d'abord, je tiens à remercier ceux qui m'ont non seulement donné la chance de réaliser cette thèse, mais m'ont aussi suivi et épaulé tout au long de cette aventure: mes encadrants. Vous m'avez transmis le sens de la rigueur scientifique, du verbe juste, tout en me guidant avec humanité. Je n'aurais pas pu rêver de meilleurs mentors que vous. Je remercie Mathieu Bouet, qui en plus de ses conseils scientifiques, a toujours su me redonner un fil d'Arianne lorsque je me sentais perdu. Je te remercie également de m'avoir fait comprendre l'importance de savoir transmettre ses idées et de convaincre de leur pertinence. Grâce à toi, j'ai réalisé à quel point avoir raison seul c'était avoir tort. Je remercie Luigi Iannone, qui en plus d'être un modèle d'intégrité, a toujours su alimenter mes recherches avec de nouvelles idées intéressantes, novatrices et appliquées. Je te remercie de m'avoir transmis un sens de la rigueur et du mot juste lors des phases de rédactions, me rappelant que ce qui se conçoit bien s'énonce clairement. Je remercie Vania Conan non seulement de m'avoir accueilli chaleureusement au sein de TAI, mais également d'avoir suivi avec intérêt mes travaux tout au long de ces trois années. Tu as, à de nombreuses reprises, su me faire prendre de la hauteur sur les problèmes, tout en transformant les difficultés que je rencontrais en opportunités. Je remercie également Gérard Memmi qui a rendu cette thèse possible. Je vous remercie également pour tous les moments de joie passés avec vous que ce soit en réunion, en conférence ou même au détour d'un café qui ont

égayé ces trois années de thèse.

Je tiens également à remercier les membres du jury qui ont donné leur temps et leur énergie pour évaluer ce travail. En particulier je remercie Stefano Secci et Thierry Turletti pour avoir relu avec attention ce manuscrit et pour leurs précieux commentaires qui m'ont permis d'améliorer le contenu de cette thèse. Je remercie aussi Andrea Araldo et Hind Castel qui ont accepté de faire partie du jury. Je suis sûr que leur expertise améliorera mes réflexions sur le domaine.

Je remercie également l'ensemble de mes collègues du laboratoire TAI qui par leur bonne humeur ont constitué un havre de paix que j'étais heureux de retrouver tous les matins (en commençant bien sûr par une sacro-sainte pause café !). Merci pour les nombreux pots, les moments de rire, vos conseils tout au long de ces trois années que je n'oublierai pas. Merci à Allan, Raphaël, Filippo, Erwan, Damien, Dallal, Hicham, Ehsan, Geoffroy, Farid et Bruno. Je tiens également à remercier Kevin, qui en plus d'être le plus grand organisateur de pot de TAI, a sur un angle plus professionnel implémenté une version SRv6 du NFV-Router, donnant un impact plus important à cette contribution au sein de Thales. Je suis aussi reconnaissant envers Azharia qui a toujours su m'aider avec gentillesse dans la gestion des différentes procédures au sein de Thales tout au long de ma thèse.

J'ai également une pensée particulière pour mes compagnons de galère Clément, Agathe et Sadia qui m'ont respectivement précédé, accompagné et suivi dans ce doctorat. Merci pour votre soutien et ces longues discussions qui m'ont tant apporté au cours de ces trois ans.

Sur un plan plus personnel, je tiens à remercier mes amis qui parfois malgré la distance continuent, tel le pilier Djed, à être une source de stabilité et de joie dans ma vie. Merci d'être là, vous êtes comme une deuxième famille pour moi. Merci à Léo, Augustin, Magda, Célia, Maxime, Basile, Théo, Florine, Mathilde, Catherine, Paul P., Camille, Victoria, Paul J., Louise et Justin. Je vous dois énormément. Je suis fier et reconnaissant de vous avoir comme amis.

Finalement, je tiens à remercier toute ma famille qui est une source de soutien indéfectible. Plus particulièrement, je tiens à remercier mes parents Eric et Annick. Merci d'avoir toujours été là pour moi, merci de croire en moi plus que je ne sais le faire, merci pour toutes les valeurs que vous m'avez transmises, rien de tout cela n'aurait été possible sans vous. Je tiens également à remercier particulièrement mon frère et ma soeur: Maxime et Eloïse. Depuis la plus tendre enfance vous avez toujours été des compagnons de

route et surtout une source de bonheur absolue pour moi. Merci pour tout.

Abstract

English Version

During the last years, network infrastructure has moved from dedicated-hardware solutions implementing fixed functions to more flexible software based ones. On one hand, SDN (Software Defined Network) can flexibly control forwarding operations, while on the other, NFV (Network Function Virtualization) creates elastic functions that can scale with the user demands. So far, these solutions have been used to simplify network management and operations, but they let envision a network that can automatically react to network events. In this thesis, we explore to what extent these new software networks can be used to react and adapt finely to the network dynamics. Our first contribution focuses on service chaining: the ability to steer flows through a set of waypoints hosting functions before they reach their destinations. We show that a distributed control plane that relies on existing routing protocols and is constituted by autonomous nodes can dynamically steer traffic through chains of services. Our solution finely adapts its decision to the network traffic and automatically balances the induced load on the functions present in the network. Moreover, our proposal, contrary to existing solutions, can be incrementally deployed in today's network. In our second contribution, we compare two types of chaining decisions: a centralized one with an end-to-end view of the chain and a distributed approach that solely routes flow from a function to another. We show that the two decisions are close in realistic topologies. Thus, hop-by-hop chaining could be used without affecting chaining performance. Finally, we explore how software networks can react to network dynamics in datacenters. So far, load balancers use static policies to spread incoming traffic on servers, which leads to imbalance and overprovisioning. We propose to close the loop and dynamically adapt the policy to the server load variation. Our MPC (Model Predictive Control) approach proved to be efficient to reduce load imbalance at a slow pace, thus improving the number of requests a cluster can process.

French Version

Au cours de ces dernières années, les réseaux se sont transformés passant d'une infrastructure à base de matériel dédié implémentant des fonctions statiques à des solutions logicielles plus flexibles. D'un côté, le SDN (Software Defined Networks) permet de contrôler les opérations de transmission, alors que de l'autre le NFV (Network Function Virtualization) crée des fonctions élastiques qui peuvent s'adapter à la demande. Jusqu'à présent, ces solutions ont été utilisées pour simplifier la gestion et l'exploitation des réseaux mais elles laissent également envisager un réseau qui peut automatiquement réagir à des événements réseaux. Dans cette thèse, nous explorons dans quelle mesure ces nouveaux réseaux logiciels peuvent être utilisés pour s'adapter à la dynamique inhérentes aux réseaux. Notre première contribution s'intéresse au chaînage de service, c'est à dire la capacité de diriger des flux de données à travers un ensemble de points intermédiaires, qui hébergent des fonctions, avant d'atteindre leur destination. Nous montrons qu'un plan de control distribué, qui s'appuie sur les protocoles de routage existants et qui est constitué par des noeuds autonomes, peut dynamiquement diriger le trafic à travers des chaines de services. Notre solution adapte sa décision au trafic sur le réseau et équilibre automatiquement la charge induite sur les fonctions présentes sur le réseau. De plus, notre proposition, au contraire des solutions existantes, peut être déployée progressivement dans les réseaux actuels. Dans notre seconde contribution, nous comparons deux types de décision de chaînage : une approche centralisée avec une vue de bout en bout de la chaîne et une approche distribuée qui dirige uniquement les flux d'une fonction à l'autre. Nous montrons que ces deux décisions sont proches dans des topologies réalistes. Ainsi, un chaînage saut par saut pourrait être utilisé sans affecter les performances du réseau. Finalement, nous explorons comment les réseaux logiciels peuvent réagir à la dynamique des réseaux dans les centres de données. Jusqu'à présent, des équilibreurs de charges utilisaient des politiques statiques afin de répartir le trafic sur les serveurs, ce qui amenait du déséquilibre et gâchait des ressources. Nous proposons d'asservir le système et d'adapter dynamiquement la politique à la variation de charge des serveurs. Notre approche MPC (Model Predictive Control) est efficace afin de réduire le déséquilibre de charge à une basse fréquence de contrôle améliorant ainsi le nombre de requêtes qu'un ensemble de serveur peut traiter.

Résumé

Nés dans les centres de données il y a un peu plus d'une dizaine d'années les technologies de virtualisation et d'automatisation ont ouvert la voie à des services réseaux conçus, implémentés déployés et orchestrés comme un ensemble de logiciels. Ce changement de paradigme proposait de passer d'un réseau constitué d'éléments matériel déployés statiquement et pouvant encaisser les pics de demande à un modèle où les différents éléments constitutifs d'un service pouvaient être déployés et adaptés dynamiquement à l'ampleur de la demande des utilisateurs. Originaire des centres de données, ce mouvement s'est propagé et a rapidement conquis les réseaux mobiles [96, 101] et d'infrastructure [38, 55] et révolutionne actuellement les architectures des réseaux ainsi que leur exploitation. Ce mouvement appelé *network softwarization* a permis de i) avoir un contrôle plus fin sur les chemins empruntés par les flux de données et ii) d'adapter dynamiquement le dimensionnement de l'infrastructure. Ces deux innovations donnent une interface de contrôle pour à la fois allouer au mieux la demande des utilisateurs tout en adaptant le dimensionnement de l'offre de services. Ces réseaux programmables permettent d'imaginer de nouvelles boucles de contrôles pour s'adapter dynamiquement à ces changements. Par exemple, des flus de donées utilisateurs pourraient être assignés dynamquement à des ressources réseaux afin de minimiser leur utilisation tout en délivrant le service requis [2, 45, 59].

La conception de tels systèmes de contrôles qui pourraient optimiser le réseau sans réduire sa robustesse et sa flexibilité est complexe[66, 89]. En effet, afin de pouvoir être déployé de manière incrémentale, ces systèmes additionnels doivent être compatible avec les réseaux déployés actuellement. De plus les degrés de liberté supplémentaires introduits par les réseaux logiciels augmentent significativement la complexité des problèmes de contrôle et nécessitent de i) superviser l'état du réseau et d'extraire des informations supplémentaires de ce dernier et ii) de prendre rapidement des décisions afin de s'adapter au mieux à la dynamique de ces changements d'états. Finalement, ces nouveaux systèmes de contrôle devraient être conçus afin de minimiser la complexité additionnelle qu'ils introduisent tout en permettant une augmentation des performances du réseau ou l'introduction de nouveaux

services.

Dans cette thèse, nous explorons les possibilités données par les réseaux logiciels de contrôler dynamiquement le plan de commutation. A la place d'algorithme statique prenant des décisions à partir d'une description statique du résau (topologie, bande passante d'un lien ...) nous proposons de nouveaux réseaux où la charge courante du réseau est utilisée par une boucle de rétroaction afin de modifier et d'optimiser les décisions de routage. Ici, nous utilisons le terme routage dans un sens large qui n'est pas limité aux décisions prises par des routeurs afin de commuter des paquets, mais peuvent également comprendre des équilibreurs de charge qui routent des requêtes sur des serveurs applicatifs.

Cette proposition introduit un processus dynamique qui est asservi à la performance du réseau. Notre travail se distingue des approches classiques d'ingénierie de trafic qui sont utilisées sur des échelles de temps longues (de l'ordre de l'heure) et s'appuyant sur des tendances de long terme afin d'estimer des matrices de trafic. Dans notre travail, nous gérons la volatilité du trafic utilisateur à une granularité plus fine (e.g. des flux TCP). Cette décision est motivée par deux choses. Premièrement, l'introduction de fonctions additionnelles dans le réseau introduit de l'état dans le réseau (souvent associés à un flux identifiés par les *5-tuples*). Ainsi rerouter de tels flux (et a fortiori changer l'instance de fonction qui la traite) est difficile et devrait être évité. Il est alors crucial de choisir un chemin adéquat (i.e., non surchargé) lors du routage du premier paquet d'un flux. Deuxièmement, gérer le trafic à une granularité plus fine augmente le gain potentiel de performance, car nous plaçons des unités atomiques plus petite. Cependant, prendre des décisions de routage dynamique à une telle fréquence impose alors à la boucle de contrôle de réagir rapidement à la dynamique du trafic. Ces difficultés ajoutent des contraintes significatives pour toute architecture ou algorithme de contrôle envisagé.

Nous proposons d'appliquer cette idée dans deux domaines : le chaînage de service et l'équilibrage de charge dynamique. Le **Chapitre 2** présente le contexte et les travaux existants liés aux différentes contributions présentées dans les chapitres ultérieurs.

Dans le Chapitre 3 et le Chapitre 4, nous présentons les contributions relatives au chaînage de service. Le chaînage de service correspond à la capacité de diriger le trafic réseau à travers une séquence ordonnée de fonction réseaux (appelée chaîne) avant de livrer les paquets à leur destination.

Dans le **Chapitre 3**, nous proposons une nouvelle solution pour créer des

chaînes de services. La plupart des propositions existantes s'appuient sur une approche centralisée pour calculer et installer les règles de commutations sur chaque équipement réseau. Cette flexibilité a un prix : une solution plus fragile comparée aux protocoles de routages actuels qui requiert le déploiement d'une toute nouvelle infrastructure réseau. A la place, nous proposons d'étendre la couche de routage à la notion de service. Notre solution s'appuie sur les protocoles de routages distribués existants, ainsi des noeuds autonomes sont capables d'annoncer des informations relatives aux services virtualisés qu'ils peuvent fournir. Cela crée une vue étendue du réseau qui permet de créer des chaînes de services où la décision de routage est prise à partir des performances actuelles du réseau. Notre architetcure est modulaire, peut être déployée de manière incrémentale et a été implémentée dans des routeurs modifiés appelés *NFV-Router*. Nous montrons avec des expériences à large échelle que ces NFV-Routers peuvent diriger efficacement le trafic à travers des chaînes de service, qu'ils répartissent finement la charge sur différentes instances de fonctions, et, que le trafic de contrôle additionnel induit par notre solution est négligeable.

Dans le **Chapitre 4**, nous comparons deux approches afin de construire des chaînes de services : un routage à la source et un routage au saut par saut. Notre précédente contribution permet de construire une vue distribuée de la topologie réseau ainsi que des services réseaux instanciés sur ce dernier. A partir de cette vue, les flux entrants sur le réseau doivent être routés à travers un ensemble de points intermédiaire qui peuvent soit être choisi au point d'entrée du réseau ou au fil l'eau: segment par segment (i.e., chaque router où est instancié un service choisi le chemin jusque au prochain service et ceux jusqu'à atteindre la fin de la chaîne de service nécessaire). Dans la première approche, quand la décision de routage est prise, la chaîne dans son entier est considérée,alors que dans la seconde, la décision est prise à chaque routeur delivrant un service. Le routage au saut par saut a plusieurs avantages: i) il augmente la résilience de l'architecture de contrôle, et ii) il permet de séparer le problème de contrôle en un ensemble de sous problèmes parallélisable sur les NFV-Routers. Cependant, la décision au saut par saut, à cause de cette séparation en sous-problème considérant uniquement le prochain service (à la manière d'un algorithme glouton) peut amener à une décision sous optimale qui peut avoir un impact sur les performances du réseau. Dans ce chapitre, nous montrons les compromis entre ces deux types de décisions. Nous proposons deux modèle ILP afin de comparer de manière analytique

ces décisions. Nos évaluations montrent que i) la décision de chaînage distribuée est proche de la solution centralisée et ii) l'approche distribuée grâce à sa boucle de contrôle locale a une plus grande réactivité pour s'adapter aux événements réseaux comparativement à la solution centralisée.

Tout comme le routage réseau, l'équilibrage de charge applicatif est principalement statique et ne s'adapte pas à la dynamique du trafic ce qui amène à un déséquilibre de charge significatif dans les centres de données. L'une des principales raisons est que les équilibreurs de charges sont agnostiques à la charge des serveurs et ne peuvent donc pas réagir à cette dernière. S'inspirant de notre travail fait au Chapitre 3, nous proposons d'asservir la politique d'équilibrage de charge à différentes métriques renvoyées régulièrement par les serveurs, nous permettant ainsi de créer une politique d'équilibrage dynamique.

Dans le **Chapitre 5**, nous proposons une solution basée sur la théorie du contrôle afin de créer un équilibrage de charge dynamique dans des centres de données. L'introduction de plan de commutation programmable a permis d'implémenter des politiques d'équilibrage sophistiqués même si le nombre de serveurs dans une grappe change au cours du temps. Cette avancée permet de fermer la boucle et de construire un équilibreur de charge asservie à la performance des serveurs. Cependant, les politiques dynamiques existantes requièrent une fréquence de mise à jour de l'état des serveurs importante afin de réduire le déséquilibre de charge. Ainsi, ces dernières peuvent générer un important trafic de contrôle additionnel, ce qui rend impossible leurs utilisations à large échelle. Dans ce chapitre, nous proposons un plan de contrôle pour l'équilibrage de charge avec une fréquence lente de mise à jour de l'état des serveurs. Nous proposons de modéliser une grappe de serveurs comme un système dynamique asservi par la politique d'équilibrage de charge et d'utiliser une approche de commande prédictive afin de minimiser le déséquilibre de charge. Notre approche a permis de réduire le déséquilibre de charge sur une grappe de serveur. Nous avons comparé avec des simulations notre approche aux politiques d'équilibrage existantes utilisés par d'importants fournisseurs de services dématérialisés et avons montré qu'elle améliorait l'équilibrage de 10% avec une fréquence de contrôle lente. Notre solution permet donc de réduire significativement le nombre de serveurs nécessaires pour encaisser une charge donnée de trafic utilisateur.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

> Ne pas railler, ne pas déplorer, ne pas maudire, mais comprendre.
>
> Baruch Spinoza

Born in datacenters more than a decade ago, virtualization and automation technologies paved the way to network services designed, implemented, provisioned, and orchestrated as pieces of software. This paradigm shift envisioned to move from a static over-provisioned model to one where services can be dynamically deployed and scaled to fit users' demands. From datacenters, this trend has quickly spread to mobile [96, 101] and infrastructure networks [38, 55] and is now revolutionizing network architecture and operation. This movement called *network softwarization* has enabled to i) have a finer control on the flow paths and ii) rapidly tune the current infrastructure provisioning. The first gives a control interface to allocate the demand while the second allows to finely adapt the supply. These programmable networks let envision new control loops to accommodate dynamic changes. For instance, user flows could be dynamically assigned to network resources so to minimize their usage while delivering the required service [2, 45, 59].

Designing new control systems that could optimize the network without weakening its robustness and scalability is challenging [66, 89]. Indeed, to be incrementally deployed, these additional systems should be compatible with the current networking devices. Moreover, the degrees of freedom have significantly increased the complexity of the control problems while requiring to i) monitor or extract additional information in the network and ii) take fast decision so to adapt to the network dynamics. Finally, these new control systems should be carefully designed to minimize their complexity and overhead while increasing the network performance or delivering new services.

In this introduction, we first review the flexibility introduced by softwarized networks. We then expose the challenges in controlling such systems. Finally we introduce our different contributions to the field.

1.1 Controlling Softwarized Networks

Network Softwarization has introduced two main changes in networks. First, it enables to program the behavior of networking functions. Second, it has modified the network-provisioning model, switching from specialized hardware devices to elastic virtual appliances running on a shared infrastructure.

1.1.1 From Hardware to Software Networking Operation

Once statically defined by slowly standardized protocols, now networks can more and more be finely controlled by the operators who actually run the network.

Networking devices are traditionally divided into two main components: the control plane and the data plane. The data plane achieves the primary functions of switches or routers: it processes and forward packets at high speed. In most devices these functions are achieved by fixed hardware primitives controlled by the information stored in the Forwarding Information Base (FIB). The control plane consists of the algorithms and protocols that manage the data plane by populating the FIB. This software part can be for instance distributed routing protocols such as OSPF or BGP. This component is mostly implemented in software and is often bundled with the device operating system.

In most of networking devices, these two planes were tightly integrated by the manufacturer who only exposes to the final user an interface to configure the required features making them hard to manage. FoRCES [42] and Openflow [65] started to open these black boxes by introducing a standardized API between the control and the dataplane. These new APIs gave the possibility to rethink the control plane architecture and the way it was implemented. For instance, Openflow proposed to move the control plane logic from forwarding devices to a central control point connected by an out-of-band network.

This new control interface on networking behavior quickly gained traction to simplify the Traffic Engineering, which was limited by the logic of standardized routing protocols (e.g., packets only follow the shortest path).

Google [55] and Microsoft [38] successfully leveraged on this technology to improve the bandwidth usage. In their WAN, flows can be dynamically allocated to specific paths so to reduce link usage and meet the required quality of services. These widescale deployments showed the potential gain for the industry.

From this technological shift, two main approaches have emerged. The first proposes to consolidate the control plane state and processes in a logically centralized component, reducing networking devices to forwarding elements remotely controlled [65]. Even if this solution simplifies network management, it raises scalability and resiliency issues since it introduces a single point of failure, hence fragile. Moreover, this solution introduces new devices that are not interoperable with already deployed switches and routers. The second advocates that local distributed control plane is primordial for the network robustness. They proposed a centralized solution that can achieve the same level of programmability on top of the existing and already deployed routing protocols [6, 43, 102, 103, 110, 111]. In this vision, a central control point can optimize the network while distributed protocols conserve the original robustness of the IP stack as a default behavior.

Even with these new APIs, networking devices behavior was still limited by the fixed logic available on the switching chips often implemented as Application Specific Integrated Circuits (ASIC) [11]. It was conventional wisdom in the networking community that programmable chips were 10 to 100 times slower and consume much more power compared to fixed function chips [11]. This statement started to collapse in 2013 when a first chip design was introduced which supported reconfigurable match table [11]. Quickly, a first language was proposed to program the forwarding plane [10], while in 2016 the first programmable networking chip as fast as ASIC hits the market.

This breakthrough allows a programmer to specify fully how a packet is processed in the dataplane while benefiting of the performance of the underlying hardware target. This flexibility enables to redefine the fundamental primitives done by the data plane which can for instance more finely monitor packets [18, 40, 51] or accelerate some of the control plane functions [48, 49].

All these advances have made possible to network operators to finely control packet processing and forwarding in their network giving them i) a sharper view on the network events and ii) more flexibility in the possible reactions to these events. This wider span of control opens up perspectives to rethink network architecture. Yet the proposed improvements still need to

deal with the same problem inherent to distributed system (e.g., state management) [31, 80] and cope with the underlying hardware limitations (e.g., number of forwarding rules limitations) [46].

1.1.2 Software Networks Provisioning

In 2012, major Internet Service Providers proposed the concept of Network Functions Virtualization (NFV [45]). This proposal aimed at bringing the benefits of virtualization technologies into the network. Virtualization enables to run and isolate different software functions on the same underlying hardware. This technology would allow different tenants to provide isolated services on the same device and open the way to more elastic networks where functions can be instantiated, scaled or removed on the fly to meet the demand.

A first target of NFV was middleboxes. Middleboxes are specific hardware appliances that were first introduced by operators to quickly deploy new features in their networks (e.g., Firewall, NAT...). This poor solution, already widespread in 2002 [14], represents a significant part of networking devices in enterprises and datacenters networks while being the source of the majority of the infrastructure cost and network failures [82, 93]. Middleboxes virtualization offers to simplify their management and reduce their cost by moving their function into virtualized software appliances running on commodity hardware.

The virtualization of these service functions significantly modifies the way network are managed. First, it requires adding the notion of service in the forwarding decision. Originally, middleboxes were physically connected and acted as a bump in the wire that processed packets while being invisible to the network topology. Now, the packets have to be i) classified and map to the adequate set of functions and ii) routed through a set of waypoints to deliver a service before reaching their destination [41]. Second, it requires finely managing the resource usage in the network [45]. A slice of processing resources is allocated to each functions (e.g., CPU core) and if exceeded would lead to performance degradation. These lead to two complementary approaches to optimize the resource use: i) dynamically allocate incoming flow based on the resource usage and ii) scale the function's resource slot based on the current workload.

Many deployment of virtualized network functions have been made on

x86 servers due to their cheap cost. Yet, this architecture, despite being optimized by kernel bypass techniques [21, 29], performs poorly compared to specialized hardware. Some envisions using hardware accelerator to optimize the function's packet processing [26], while interestingly, others try to add virtualization into programmable networking chips [115, 117]. Although these different hardware targets promise to improve the performance, it significantly increases the resource management problem complexity [26].

NFV allows operators to finely compose the available resources and functions in the network. Even if it promises to optimize the resource use, it considerably grows the network environment complexity, which have to dynamically control additional components.

1.2 Challenges

With network softwarization, operators have now the tools to i) finely allocate the flows that consume the network resources (bandwidth, CPU...) and ii) modify the current resource provisioning to adapt to the load. Even if these tools provide the building blocks to optimally allocate resource consumers and providers, performing such a task online in real network is extremely challenging.

It can be tempting to try to solve these two problems jointly. Several works propose linear programming models and heuristic algorithms to compute the optimal placement of both the flow and the resource provisioning (e.g., network functions location and resource allocation [45]). Although these solutions can work for network planning, they can hardly be ported when dynamics are considered. Indeed, they assume that a control system can i) gather the necessary information for their algorithm and ii) apply the computed decision on the network. Yet, performing both of these tasks online in real networks is arduous. In practice when the first packet of a TCP flow is routed, little is known about its future resource consumption (link and CPU usage). Moreover, most of the considered network functions keep state on the flow they process, meaning that they can only be rerouted at high cost. Finally, the forwarding and allocation decisions are taken at different timescale. A packet should be forwarded in few nanoseconds in high-speed networks, while modification in the network provisioning takes several orders of magnitude longer. These facts have several consequences: i) the forwarding decision has to be taken solely on the past or current network state (reactive control process), ii) a change in the forwarding decision should only

affect new incoming flows and iii) flow routing and resource provisioning should be split into orthogonal sub-problems.

Even when reduced to resource aware routing (i.e., forward flows based on the current resource usage), the problem is hard to implement in real networks. So far, a routing process considers static and stable network related metrics to build a routing table that is not modified unless sporadic event occurs (link failure, policy change...). Finely routing flows based on the current performance requires regularly broadcasting additional metrics not only related to the network state (e.g., link usage) but also information about the current network function performance (e.g., CPU consumption). These additional metrics are crucial to know the current state of the network and accurately route new incoming flows. Yet high frequency broadcast of this information can lead to a significant control overhead that could reduce network performance. Based on this accurate network description, new algorithms should be designed to build resource aware routing table. One of the challenges raised by these algorithms will be to react quickly to the dynamics of network and computing metrics without creating network instability.

One of the most daunting task is to propose solutions that not only succeed to cope with the above-mentioned challenges but also are interoperable with current network, meaning that any introduced solution could be incrementally deployed in the network. Moreover, when an additional component is added into the network, it should try to minimize the added operational complexity and scale with the network growth. For instance, architectural principles such as fate sharing, or loose coupling, should remain guidelines when proposing new ideas that modify the network architecture [66].

1.3 Contributions and Thesis Outline

In this thesis, we explore the possibility given by softwarized network to dynamically control its forwarding behavior. Instead of a control algorithm taking decision only on static network description (topology, link bandwidth), we envision new networks where the current resource usage is feedback to modify and optimize the routing decision. Here we use the term routing decision in a broad sense which is not limited to the decision taken by routers to forward packet from a source to a destination and could also encompass load balancers which route requests to backend servers.

This introduces a dynamic control process that is driven by the current network performance. Our work sets apart from traditional traffic engineering that occurs at coarse timescales (hours) based on long-term estimates of traffic matrices. In our work, we aim at handling traffic volatility at a fine-grained level (e.g., TCP flows). This decision is motivated by two facts. First, the introduction of in-network function adds state in the network (often related to 5-tuples). Thus rerouting a flow from a path (and a fortiori changing of function instance) is hard and should be avoided, making crucial to choose an adequate path (i.e., not overloaded) in the first place. Second, handling traffic at a finer granularity increases the potential performance gain, since we handle smaller unsplittable objects. Nevertheless, handling flow at this small timescale puts a particular stress on the control loop, which should then quickly reacts to the dynamics, adding significant constraints on the considered control algorithm and architecture design.

We propose to apply this idea in two domains: service function chaining and dynamic load balancing. **Chapter 2** presents the background and related work of the different contributions necessary to a full understanding of our different pieces of work.

In Chapter 3 and Chapter 4, we present our contributions related to service function chaining. Service function chaining refers to the ability to steer network traffic through a sequence of in-network function called service chain before delivering packets to their destination.

In **Chapter 3**, we propose a new solution to perform service function chaining. Most of the existing proposals rely on a logically centralized approach to compute and install the correct forwarding behavior on every network devices. This flexibility comes at the cost of increased fragility compared to current Interior Gateway Protocols (IGP) and of a whole new infrastructure. Instead, we propose to augment the routing layer with the notion of services. Our solution leverages on existing distributed routing protocols where, in addition, autonomous nodes announce information about the virtual services they provide. This creates an *augmented network vision* that can drive performance-aware routing through chains of services. Our design is modular, incrementally deployable and has been implemented in what we call an *NFV Router*. We show in our large-scale testbed deployment that NFV Routers efficiently steer traffic through chains, finely distribute the load among different function instances and only add a small overhead to control traffic.

In **Chapter 4**, we compare two approaches to build service function chains:

source routing and hop-by-hop routing. Our previous work enables to build a distributed view of the network topology and related services function. Based on this view, incoming flows have to be routed through a set of waypoints that can either be chosen at the ingress point or segment by segment (i.e., each service point chooses the path to the next one until the chain ends). In the first approach, when the routing decision is taken, the whole chain is considered, while in the second, the decision is taken at each service point. Hop-by-hop routing has several advantages: i) it increases the control architecture resiliency and ii) it allows to split the chaining problem into smaller pieces that can be parallelized on the NFV-Routers. Nonetheless, hop-by-hop decision, due to its myopic view, can lead to a sub-optimal chaining decision that could impact network performance. In this chapter, we show the tradeoff between these two types of decisions. We propose two ILP models to compare analytically the decisions. Our evaluations show that i) distributed chaining decisions are close to optimal centrally-computed paths and ii) the distributed approach, because of its local control loop, has a faster reaction to network events than centralized solutions.

Similarly to network routing, load balancing policies are mostly static and do not adapt to traffic dynamics, which lead to a significant imbalance on datacenters' server. One of the main reasons is that the load balancers are agnostic to the servers' load dynamic and cannot react to them. Inspired by our work done in Chapter 3, we propose to drive the load balancer's traffic splitting by additional metrics regularly broadcasted by the servers, thus creating a dynamic load balancing policy.

In **Chapter 5**, we propose a control theory based solution to drive dynamic load balancing in datacenters. Programmable data planes have allowed implementing sophisticated dynamic load balancing policies even if the number of active servers and load balancers changes. This breakthrough enables to close the loop and build a performance aware load balancer's control plane. However, existing dynamic policies require a fast control pace to reduce load imbalance. Thus, they can generate a huge control traffic overhead, which makes it unfeasible in large-scale deployment. In this chapter, we propose a dynamic load balancing control plane with a slow control pace. We propose to model a server pool as a dynamic system controlled by its load balancing policy and use a model predictive approach to minimize the system load imbalance. Our approach proved to be successful to reduce load imbalance on the servers. We compares in simulations our proposal to current load balancing policies used by large cloud providers and showed that

it improves load imbalance up to 10% with a low-frequency update. Our solution can significantly reduce overprovisioning in large-scale datacenters while still fulfilling service level agreements.

Chapter 2

Related Work

> La différence entre la théorie et la pratique, c'est qu'en théorie il n'y a pas de différence entre la théorie et la pratique, mais qu'en pratique, il y en a une.
>
> Albert Einstein

This chapter introduces the concepts, background and related work that are fundamentally related to the work presented in Chapter 3, 4 and 5. This related work can be split into two main parts: first, works related to Service Function Chaining, and, second, ones linked to load balancing.

2.1 Service Chaining

In this section, we present the problems, concepts and related work to service function chaining which are tackled in Chapter 3 and 4. Service Function Chaining refers to a technique used to steer traffic (often atomically reduced to UDP/TCP flows) through a set of functions before delivering the packets to the destination. We call these functions virtual Service Functions (vSF). This technique allows extracting vSFs from the physical topology (middleboxes used to intercept traffic on the wire) at the cost of supplementary constraints in the routing process. In addition to forward packets from a source to a destination, packets should be steered through a set of waypoints where service functions are instantiated. While some work focus on chain of functions when the instances are located on the same machine [62], we instead are interested on network functions distributed among a set of hosts in the network.

With this new paradigm, technical challenges have emerged:

Routing Decision

A routing process should gather and exchange the location and related metrics of vSFs to compute path through service function chains. This decision can be static or taken online. The control architecture can either rely on a SDN controller or directly leverage on existing distributed routing protocol.

Traffic Steering

IP packets are forwarded from a source to a destination. Thus to modify the packets path and deliver them to service functions additional information should be added. This additional state either can be added to the packets or populated on the network devices to properly forward packets through chains of service.

State Management

Most of the vSF are stateful (e.g., NAT, IDS...), which adds a significant constraint in the routing process. Along its lifetime, a flow should be steered through the same set of vSF instances. Otherwise, solutions have to be proposed to share state between multiple instances, either to migrate a flow from a path to another, or to react to a vSF failure.

Resource Allocation

The delivery of flows through service function chains should use the minimal set of resources and avoid overloading a link or a function. This task is particularly challenging since a flow should be allocated when the first packet of a flow arrives and little is known on its future resource usage.

In the following subsections, we summarize the existing work to highlight the main techniques used to solve these problems.

2.1.1 Routing Decision

Most existing solutions manage service chains by relying on a central control point, following the Software Defined Network architecture (SDN). Based on a holistic network view, they run an offline resource allocation algorithm to place vSFs and assign static flow paths [37, 45]. Even if these solutions provide theoretically optimized placement for a batch of requests, they can hardly be ported in real networks. Indeed, they assume that (i) requests are known in advance, and (ii) vSFs and flows can be placed at the same time.

Some works integrate in their orchestration scheme the technical limitations induced by their chain placements, such as vSF flow affinity [79]. Nonetheless, they mainly rely on real-time responses from a remote central controller to monitor vSFs, network state and enforce per-flow static path. This approach tightly integrates vSF placement and flow paths, which thus increases their system fragility (single point of failure, control loop delay). In such an approach, vSF placement and service chaining decisions are jointly taken. However, they deal with different timescales. Instead, we argue that these decisions should be decoupled: service chains are steered in a fast loop so to enforce traffic-engineering policies on existing vSFs, while a slower control loop adapts vSF provisioning.

To cope with some of these limitations, recent works [17] propose to distribute the orchestrator. Yet, distributed SDN controllers such as Opendaylight [77] or ONOS [75] have their own limitation. Indeed, they rely on a consistent distributed database meaning that when network partition occurs, some service function chains may not be enforced for new incoming flows [80]. SDN controllers have preferred consistent network update compared to the eventual consistency approach (and thus availability) of classical routing protocols. This considerably hardens network update and can even lead to potential deadlock if not handled correctly [13, 31, 56, 72].

To manage the limitations of a centralized architecture, while keeping its benefits, proposals have been made to design a hybrid SDN architecture [102]. For instance, they propose to rely on a central control node to configure network devices and use existing distributed network protocol as a primitive to enforce complex behavior [43, 103]. Our approach, presented in Chapter 3, follows this line of thought to manage service chains paths, but differs from it since autonomous nodes instead of a central controller take the decisions.

2.1.2 Traffic Steering

To enforce service function paths, several traffic steering techniques have been proposed. Most of the existing works leverage on a central controller to populate fine-grained forwarding rules on every network appliances along a flow path [30, 35, 36, 79, 116]. Several limitations of this approach have been identified. First, these rules grow with the number of flows, policies and chains' size, while forwarding state on network appliance is limited by costly memory [43]. Second, they grow in complexity when a vSF make a

hard to handle change in network headers (e.g., Network Address Translation service) [30, 35, 83].

Recent works, instead, propose to (i) encode service chains as a set of waypoints in the packet header, and (ii) rely on the network routing layer for waypoint connectivity [1, 43, 114]. This approach is not only interoperable with regular IP networks but also reduces forwarding state. Indeed, a flow path is either fully described in packet headers [1, 43] or stored on waypoints at flow initialization [114]. Several techniques have been proposed to encode the path of service function chains [1, 84, 114]. In Segment Routing v6 [1] and Dysco [114], an ingress node is in charge of setting a list of locations to reach before being delivered using IPv6 and TCP extensions respectively. Recent work at the Internet Engineering Task Force (IETF) proposes Network Service Header (NSH) as a dedicated encapsulation header for service chaining [84]. We follow this line of thoughts to build service functions chains and, in addition and differently from previous work, we propose a generic method to build a distributed control plane based on existing routing protocols for service chaining in Chapter 3.

2.1.3 State Management

The majority of vSFs, stores session state about the flow they process, which hardens service elasticity. Indeed, vSF instances can be created, scaled or destroyed due to fluctuation in flow volumes, migrated for resource optimization, or just recovered due to failure. When these events happen, flow's paths can be modified and flow's session state can be migrated from a vSF to another. Some solutions have been proposed to coordinate forwarding and session state. OpenNF [36] and Split/Merge [86] propose to add an open interface on vSFs so to allow a central control point to coordinate flow re-routing and session state migration with a make-before-break approach. Dysco [114] argue instead that forwarding and session state should be consolidated in vSFs and rely on a distributed session protocol to reconfigure a service chain. Kablan et al. [57] avoid this state coordination problem by making vSFs stateless. They consolidate session state in a consistent high-speed back-end data store, which limits this solution to vSFs sharing the same location. Khalid et al. [60], state that existing solutions lack consistent shared state between vSF instances. They propose a framework fulfilling a set of requirements to support correct and efficient service chains composed of stateful functions. We argue that even if state coordination and consolidation are suitable for

local changes (happening on the same Point of Presence); they do not scale to a multi-site environment, which would be better served by a distributed protocol.

2.1.4 Resource Allocation

Resource allocation problem have been widely covered in the literature [45]. Most of the solutions make the following hypothesis: i) similarly to jobs in a datacenter, SFC requests are sent to a controller which should place them ii) these requests are constituted by the set of needed vSFs and the capacity (CPU, bandwidth) they will use. With this set of data, a model of the network is proposed and an objective function designed to take into account one or several metrics (resource use [26], energy consumption [53], cost [73], flow latency [44], bandwidth requirement [52]...). In these proposals, placement of the vSFs and the requests is done conjointly. Most of the papers show that their problem is NP-Hard for arbitrary network topologies and propose heuristic algorithms to solve the problem in a reasonable amount of time. Most of the proposals study the offline problem (the set of request is a batch) but more and more online solutions are proposed [70] (the requests arrive successfully and should be placed). In this approach, service chains are built as circuits with a coarse-grained static allocation of resource that cannot cope with network dynamics (e.g., traffic variation).

The major limitation of the above-mentioned solutions is that they do not handle traffic dynamics leading to overprovisioning. When a Service Chain is requested, the amount of required resource to process the expected traffic has to be estimated. Thus, this solution leads to the same overprovisioning and rule of thumb used by operator before the virtualization of function. In our work, we try to handle service function chaining resource allocation at a finer granularity. In this vision, the routing decision is made at the granularity of a TCP or UDP flow and takes into account both networking and computing metrics. The routing decision is taken based on the current performance of the network and the instantiated vSFs. Our approach gives the possibility to place more finely resource consumers and reduces overprovisioning since more traffic can be processed on the same infrastructure. In that sense, our work is close to QoS routing where additional metrics are considered to take a routing decision [39, 97] but also multipath routing where multiple paths for a same destination are used to maximize the network throughput [34, 48, 49]. Recent proposals at the IETF follow this line of thought to

handle both the computing and network related metrics in the routing decision [113].

2.2 Load Balancing

In this section, we present the concept and related work to load balancing which is studied in Chapter 5. Load balancing emerges when a single resource (server, path...) is not enough to provide a service to all the end users. To scale, additional resources are provisioned, the goal of load balancing mechanism is to uniformly spread the incoming flow, connections toward the backend resources. In the network, this function is often done by router and rely on ECMP [47] while for application services (e.g., a web service), a middlebox map incoming connections toward a pool of server.

Some challenges are similar to the resource allocation problem explained above: incoming connection resource use is not known when its first packet enters the network and that the load balancing decision should be taken. These similarities are in fact inherent to IP networks since the induced load in the network (bandwidth use, CPU consumption ...) are driven by the end nodes. Therefore, network elements can only react to these changes and cannot proactively allocate resource consumers to services in an optimal manner (unless the network and end-users are controlled by the same entity [55]). Yet, efficient load balancing solutions significantly interest service providers since the only operational solution to load imbalance is overprovisioning the backend resources, which represents resource waste and additional cost.

First, we describe the different load balancing systems proposed for datacenters and the problems they tackle. Second, we explain the different load balancing policies used to spread connections on a set of servers.

2.2.1 Traffic Splitting

Several techniques can be used to dispatch incoming connections toward a pool of servers delivering the same service. One of the easiest to implement is DNS round robin. In theory, each client when connecting to a service need to translate a domain name into an IP address. With this solution, the DNS aims at a different IP for the same domain name in a round robin fashion each time a request is received. Yet this solution is limited by the wide use of cache in the DNS hierarchy meaning that the domain DNS will not be requested each

time a new connection to a service is happening. This solution can help to balance load on different site but can be inefficient at finer granularity [94].

One of the most widely used solution to dispatch incoming connections is thus L4 load balancer. This middlebox functionality masquerades the IP of a given service and dispatch new requests toward a pool of server. A server is chosen when the first packet of a connection arrives, using a hash of the 5-tuples to identify the connection and map the decision. There is two type of load balancer: stateful and stateless. Stateful load balancers keep a state related to the ongoing connection that map the 5-tuple to the backend server. Stateless load balancers on the other side do not track the connection but instead solely rely on a hashing function to dispatch incoming connections to the server which can lead to broken connections if the size of the pool change due to failure or scaling.

With stateful load balancers, this 5-tuple hash is directly mapped to the chosen backend server. In this scheme, the load balancing policy is decoupled from its enforcement at the cost of a high number of states (which rises with the number of connections). However, these solutions can mitigate per connection consistency violation [27, 67, 81]. Recently, Barbette et al. [8] have proposed a load balancing architecture that can enforce any load balancing policy while ensuring per connection consistency even when scaling events occur.

With stateless load balancers, the connection identifiers space is partitioned among the set of servers. Thus, any 5-tuple hash serves as a key to the chosen backend server. In this scheme, the load-balancing decision is coupled to and limited by the underlying stateless hash data structure. Usually, the server identifiers are uniformly distributed in the hash table [5, 47, 74] and aim at evenly spread the connections on the servers. Yet, when the servers' capacity is different, a weight can be associated with each server [118] meaning that a server with two times more capacity will appear two times more frequently in the hash table than a standard server. In their recent work, Hsu et al. [48] propose to use hash space boundary instead of server occurrence to choose the destination. They show that their data structure can be modified at line-rate, making dynamic load balancing possible at short timescales.

These techniques are used to enforce a load balancing policy which is often static and does not change with time (i.e., the load balancing distribution is agnostic to the load experienced by the server. Nonetheless, these static techniques fail to spread uniformly the load among servers [27]. New tools

relying on programmable dataplane allow modifying at runtime the distribution, thus paving the way to adaptive load balancing. These techniques can i) enforce a complex policy on high-speed load balancer (distribution based on server weights) [8] and ii) modify the load balancing policy in short timescales [48, 49]. This shift allows to rethink load balancing policy once static and to close the loop: a controller can adapt the load balancing distribution based on the current load experienced by the servers to reduce load imbalance.

In the work presented in chapter 5 we explicitly limit our scope to L4 load balancers which spread incoming connections towards service frontend (sometimes themselves L7 load balancers). Nonetheless notice that the same idea could be applied in lower level (L3 LB) or upper (L7 LB) in the IP stack.

2.2.2 Load Balancing Policies

Most of the existing load balancing policies are static, meaning that they aim at a mean steady-state load-balancing objective over time. Uniform load balancing can be enforced in a stateless manner with hashing space [47], at the cost of a suboptimal load balancing, which rises with the number of servers [61]. This distribution can also be enforced with a round-robin policy that sends new connections to servers in a cyclic way. Each of these policies has its alternative to deal with capacity asymmetry: weighted round-robin [104] and WCMP [118]. These solutions cannot adapt to the traffic dynamics, thus leading to imbalance.

Dynamic load balancing policies such as Power of two [69] or the least connection policy have shown to reduce the load imbalance [8]. Indeed, when a single load balancer spread the load on a service, it can use its own connection table to know the backend servers' current state instantaneously. These solutions are not scalable since it makes the ingress load balancer a bottleneck for the service. When multiple load balancers are present, these solutions need to synchronize to know the current server state. Thus, a tradeoff has to be made between the resource saved on the server and the additional control overhead that would be added on the network.

The flexibility introduced by programmable dataplane for the policy enforcement open the way to the design of more complex policy, which could forecast the load evolution and adapt the current traffic splitting to reduce load imbalance. These techniques could be based on control theory or machine learning techniques. In Chapter 5, we make a first proposition that

leverages on Model Predictive Control to reduce load imbalance. We believe that closing the loop of load balancing policy is an interesting way to explore so to reduce load imbalance in datacenter networks.

Chapter 3

Let there be Chaining: How to Augment your IGP to Chain your Services

> Que la force me soit donnée de supporter ce qui ne peut être changé et le courage de changer ce qui peut l'être mais aussi la sagesse de distinguer l'un de l'autre.
>
> Marc Aurèle

ISPs have started to replace their dedicated appliances with virtual Service Functions (vSF) through which traffic has to be steered. Most of existing work has proposed to rely on the Software Defined Networking paradigm to route traffic into the appropriate set of functions. These solutions tend to be poorly scalable and need to deploy a new architecture to benefit of Network Function Virtualization. In this chapter, we show that the same goal can be achieved by relying on IGP and present the first major contribution of this thesis: the *NFV-Router*. Our proposal tackles the following challenges to deliver service function chaining: i) incremental deployment ii) scalability and robustness and iii) performance aware routing.

This work have been incrementally designed and evaluated and have been presented and published in the following conferences and journal: the *ACM Symposium on SDN Research* (2019) [108], the *IFIP Networking Conference* (2019) [109] and the *IEEE Transactions on Network and Service Management Special Issue on Softwarized Networks* (2019) [107]. This contribution has been presented in the IRTF Computing in Network research group during the IETF 105 meeting. The technical design and concept presented below have also been patented. This proposal have received the *IEEE INFOCOM 2019 Best*

Poster Paper Award Runner Up.

3.1 Background and Motivation

Network services used to be built as an ordered set of physically wired hardware appliances that processed traffic for security or optimization purpose [85]. With Network Functions Virtualization (NFV), middleboxes are more and more software-based running on top of virtualization-enabled commodity equipment, thus allowing cost reduction and network flexibility [45]. Nevertheless, with this new paradigm, new challenges have arisen. Indeed, the set of service functions, often chained to offer complex services, are completely separated from the physical topology and virtual Service Functions (vSF) are more ephemeral and dynamic in nature. Steering traffic through these sparsely located virtual entities, without compromising end-users' sessions and Quality of Service (QoS), is, therefore, a complex challenge.

Even though Internet Service Providers (ISPs) critically rely on middleboxes for security and policy compliance [93], most existing NFV management solutions rely on an omnipotent logically centralized entity, generally named orchestrator. Such a centralized approach succeeds in controlling the forwarding behavior to perform service chaining. Nonetheless, such a result comes at a cost. Indeed, building a logically centralized solution with the same reactivity and resiliency provided by current distributed routing protocols is challenging. These remote control points must maintain a holistic view of the network topology while being able to react quickly to failures and topology changes. On the contrary, distributed routing protocols naturally parallelize the computation of forwarding behavior and make local decisions. Moreover, this centralized solution is poorly interoperable with legacy appliances and is hard to deploy incrementally. Thus, it requires that the operators completely change their operational model and drastically modify their network (legacy appliances, management tools...).

In this chapter, we show that service chaining can be performed without the above-mentioned constraints. We propose an architecture compliant with current networks, which can also be deployed incrementally. Our distributed solution takes local decisions providing better reactivity, scalability, and resiliency. Following the current philosophy in existing networks, we centralize sporadic long-term decisions that can be applied by human operators or an automated policy server and rely on distributed Interior Gateway Protocol (IGP) to compute the forwarding behavior (see Fig. 3.1). To that extent, we

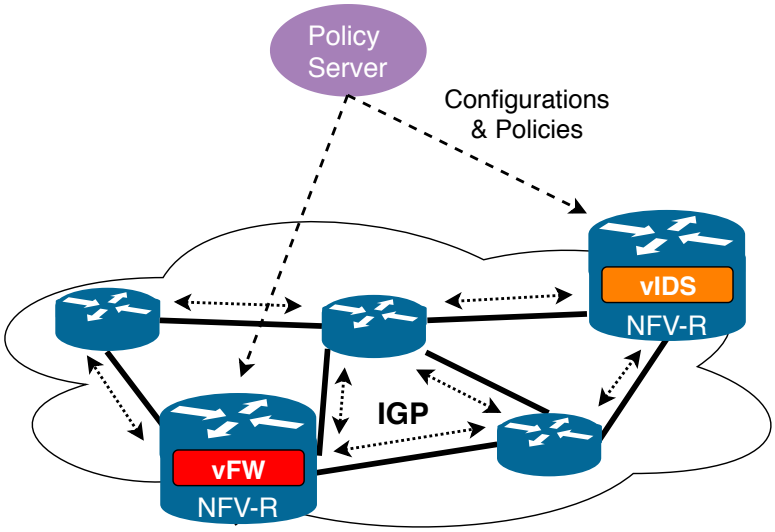
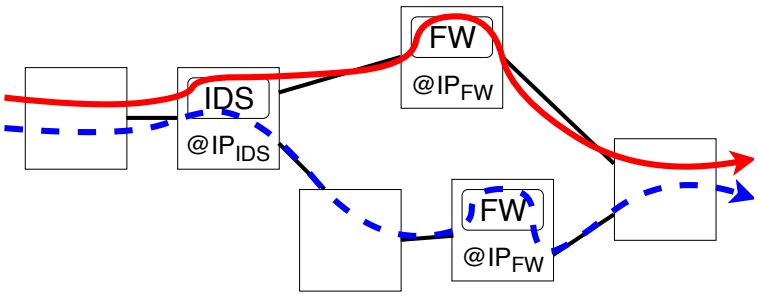


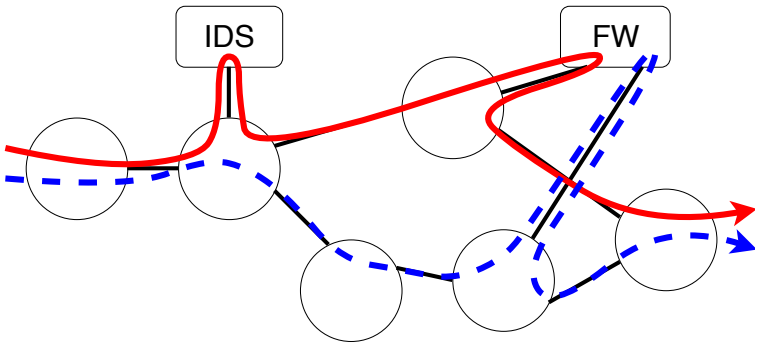
FIGURE 3.1: Network combining classical IP routers and NFV Routers. Both router types are managed and configured by a remote central policy server. IP routers and NFV Routers announce with the IGP the subnetworks that are directly connected to them. In addition, NFV Routers advertise with the IGP the vSF instances they are hosting. Based on this augmented network view, the NFV Routers take distributed chaining decisions.

propose to augment the network routing layer to make it service-aware. *We propose to leverage on any Interior Gateway Protocol (IGP), anycast addressing, and any service chaining encapsulation to construct a distributed service-aware control plane.* We design a modular architecture that we name *NFV-Router (NFV-R for short)*. We show that it does not require complex elements and remains interoperable with legacy appliances. We also implemented an NFV Router and emulated real-world ISP topologies. We show how our system successfully steers traffic through the intended service chains. We also evaluated the introduced overhead and the network dynamics in different configurations, as well as service chaining impact on link load.

The rest of the chapter is organized as follows. First, we introduce in Sec. 3.2 our modular approach to augment IGP so to allow distributed service chaining. We detail in Sec. 3.3 the architecture of our augmented node: the NFV Router. In Sec. 3.5, we describe the methodology we first present the functional evaluation of our prototype on a small scale testbed. In Sec. 3.6, we show the results obtained through large-scale emulation. In Sec. 3.7, we discuss tradeoffs and limitations of our solution. Finally, Sec. 3.8 concludes the chapter.



(A) Network topology.



(B) IGP logical view.

FIGURE 3.2: Network topology composed of 6 NFV-Rs, with 3 of them hosting a vSF instance (Fig. 3.2a). The IGP views the two FW instances as a single entity, since they announce the same anycast IP address (Fig. 3.2b). A first flow (plain red line) is routed through the IDS and the top FW instance. A second flow (dashed blue line) is then routed through the IDS and the bottom FW instance as the top FW instance is already loaded with the first flow.

3.2 Distributed Chaining with IGP Service Augmentation

In this section, we present how the network routing layer can be augmented to enable distributed service function chaining. For shortness and clarity, we explicitly limit our scope to Interior Gateway Protocol (IGP) and leave the case of external gateway protocol for future work. Indeed, *any network IGP can be directly leveraged* to convey the location, the type, and the necessary information associated with a virtual appliance and build an augmented network view. Based on this enhanced topology, any routing scheme can be used to steer traffic through service functions, that is to chain services. Such an approach enables to fully reap the benefits of the IGP field-proven scalability and robustness. Moreover, we leverage on the existing in-network intelligence, instead of adding another layer of complexity, to distribute the above-mentioned information.

An IGP enables gateways (in general routers) to exchange routing information. This routing information is then used by each gateway to construct an IGP network view and route network-layer protocols. We propose to augment such a view with the concept of service. We call it the *service plane topology*. It is composed of two types of nodes: *NFV Routers (NFV-R)*, which are equivalent to IGP gateway nodes, and *virtual Service Functions* (vSF), which is a new type we introduce. NFV-Rs are physical appliances that not only run the IGP but also host vSFs. NFV-Rs can be classic IP routers with VNF hosting capabilities, Points of Presence or even datacenters. vSFs correspond to virtual service functions instances, also named virtual network functions. They can provide different services depending on their type: Intrusion Detection System (IDS), Firewalling, NAT, stream encoding, etc. These instances are hosted by NFV-Rs, which allow them to directly announce on the IGP the functions they can provide.

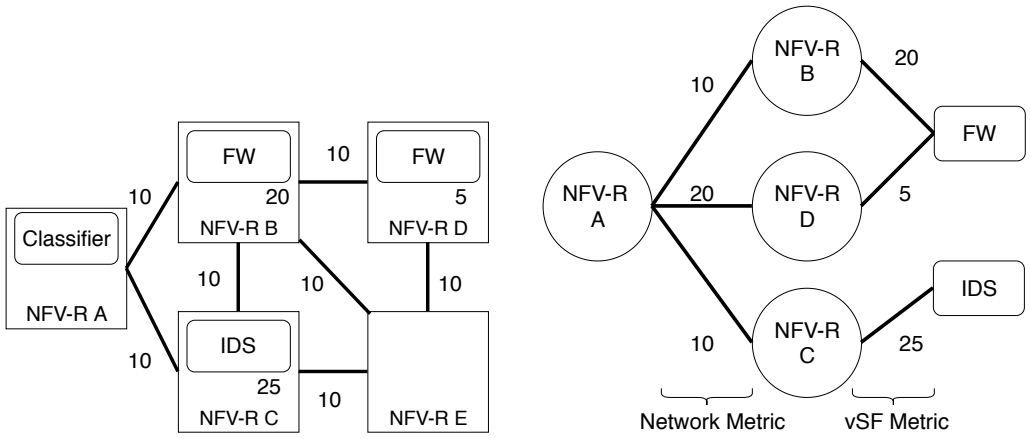
We also propose to leverage on *anycast addressing* to include vSFs in the service plane topology. All the vSF instances providing the same service are announced, by the NFV-Rs that host them, with the same IP address but with their own vSF cost. Thus in the service plane topology, a type of service is represented by a vSF node while an instance is represented by a link (see Fig. 3.2). This approach has multiple advantages. First, it reduces the number of vSF routes announced on the IGP. Second, a service chain can be explicitly and unambiguously described as an ordered sequence of waypoints to reach (the anycast addresses) and rely on the network routing layer to choose the vSF instances and the path to use. Finally, NFV-Rs only have to announce a vSF to make it immediately available for new incoming flows without any further configuration. Nonetheless, anycast routing is known to have shortcomings: packet belonging to the same flow can be routed to different vSF instances if the best path changes, which would break vSF flow affinity. To prevent this behavior, we enhance vSF announces with a forwarding address: a dedicated unicast address on the NFV-R acting as a *vSF proxy* (e.g., a router loopback interface). We also leverage on flow routing to consistently steer all the packets belonging to the same flow in the same sequence of vSF instances and thus provide statefulness at the flow level. To do so, NFV-Rs cache the initial routing decision they make when the first packet of a flow is sent to the chosen vSF forwarding address.

Figure 3.2 illustrates with a toy example the approach we propose. Figure 3.2a represents the network topology constituted of NFV-Rs. Each vSF

instance of a given type is announced on the network with the same anycast address. In particular, the two Firewall (FW) instances announce the same address: $@IP_{fw}$. Flows have to be processed here by a unique chain: $IDS + FW$. The first flow is thus routed through the IDS instance and then through the top FW instance. Indeed, in this example, this vSF instance is at one hop from the NFV-R that hosts the IDS instance. The NFV-Rs that host the used vSFs advertise their neighbors with the new experienced load or any other relevant information. When the second flow arrives, the Firewall instance at the bottom is preferred, resulting in load balancing among the FW instances (Figure 3.2b). Since the route of the first flow has been cached, it continues to be driven to the top FW instance even if the best path has changed. Note that in Figure 3.2b, since the same address is announced but no adjacency is made between the vSFs (the two Firewall instances in our example), the flows that use a link to reach a service function (drawn as boxes) have to use the same link to go out of it. However, note as well that this link is only *virtual*, since it is the representation of the vSF instance in the IGP, but in reality, is running directly on an NFV-R.

Augmenting the IGP modularly allows to fully benefit from what is already done at the network layer routing. Anycast addressing leverages IGP information sharing to build the augmented topology. Based on this topology a routing decision maps vSF type to the appropriate next NFV-R(s) based on network and instances metric. Finally, the IGP gives us robust IP connectivity between NFV-Rs to steer flows through the correct set of instances. Note that the IGP prevents flow remapping in case of link failure. Indeed, once the IGP has converged, connectivity to NFV-Rs is restored without any change in cached routing decisions. Moreover, NFV-Rs can be incrementally deployed in domains where they coexist with classical routers. Classical routers will only see NFV-Rs as IP routers announcing prefixes. Indeed, since services are announced as IP addresses, classical routers will advertise them to their neighbor. Based on this raw IP topology, an NFV-R is able to reconstitute the service plane topology.

In our approach, a lightweight central management node is responsible to configure *high-level policies* on the NFV-R. As for any IGP, these policies are common to all the nodes. They allow controlling the decision-making at each NFV-R. Such policies include flow classification rules, to map traffic to the needed service chain. They also concern routing decisions since all NFV-Rs must share the same routing objectives. Based on the service plane topology, the NFV-Rs can use any path computation algorithm (e.g., shortest



(A) Network topology.

(B) Service plane topology seen at the NFV-R A.

FIGURE 3.3: Each NFV-R builds its service plane topology (example at Node A on Fig. (b)) with the Network costs and vSF costs so as to choose the next hop(s).

path first), to choose which instance of the next vSF of the chain the flow will go through. Additionally, high-level policies can define how to compute vSFs' IGP costs, stating which data to use and the function to translate such data in a cost.

Our approach can rely on network encapsulation to convey the necessary information so to drive flows through the associated service chain. This information can be used to make routing decision at the source or at every NFV-R processing the flow (hop-by-hop). This header should include *(i)* part or all of the service chain identified at the classification step at the ingress of the network, *(ii)* the next service step in this chain, and *(iii)* a consistent flow identifier to cache the routing decision. This identifier is a hash computed on the 5-tuple, when a packet enters the network. For instance, in the example in Figure 3.2, the NFV-R that hosts the IDS instance must have a mean to know that a packet belonging to a specific flow has been assigned to the service chain $IDS + FW$, that the next service to apply is FW, and which of the FW instances it actually has to go through.

In the rest of the chapter, we focus on hop-by-hop routing, although source routing is applicable too, and take OSPF as an example for the IGP. Fig. 3.3 illustrates how the service plane topology is constructed from the network topology. In this approach, every NFV-R computes the best path(s) to every vSF type in terms of network and vSF cost and map each vSF type to the associated NFV-R forwarding address. The criteria to select the paths as well as the algorithms to choose them is out of scope of this contribution. Our

solution can convey any useful metric related to the network or the vSF instances through the IGP. Thus any suitable algorithm could be used to choose the next vSF instance(s). We give an example with the Shortest Path First algorithm to illustrate how a routing decision could be taken. With this routing algorithm, every NFV-R computes the shortest path to every vSF type in terms of network and vSF cost and map each vSF type to the associated NFV-R forwarding address. This mapping can be easily computed by running the Dijkstra algorithm on the topology presented in Fig. 3.3b and by getting the last hop before the destination in the shortest path to a vSF type. Using OSPF, NFV-Rs, as classic routers, already compute routing table to get aggregated network costs to NFV-Rs. Thus, in this case, the algorithm to find the route to the next vSF instance has to run only on a graph of depth 2 and add very low computing overhead to the routing system.

3.3 NFV-Router Architecture

In this section, we describe the architecture of an NFV-R and the design of its main modules. An NFV-R, as illustrated in Figure 3.4, is composed of a normal IP *router* providing network connectivity, a *connector*, which attaches the router to the different vSF instances, the *vSFs* themselves, providing the services, and a *Distributed MANagement and Orchestration (D-MANO)* component, which allows local autonomous management of the node.

Router

The router connects our system to the network and participates in this network's IGP. It is directly connected to the connector external interface and announces this interface's IP address on the IGP, making the connector forwarding address reachable. The router only conveys packets based on their destination IP and is unaware of the service chaining encapsulation. It exposes a control interface used by the D-MANO to inject or remove vSF anycast addresses, announcing the services available on the node and the associated costs. This control interface is also used to get the IGP topology to build the service plane topology.

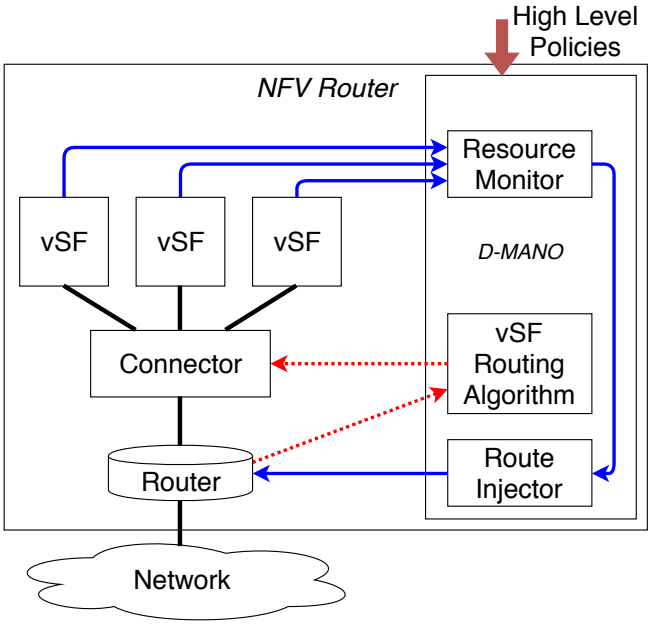


FIGURE 3.4: NFV-R architecture. Doted arrows illustrate vSF routing control flow. Solid arrows show how vSFs state is monitored, transformed in a cost, and then injected in the IGP.

Connector

The connector acts as a proxy for the vSFs and allows dispatching traffic to them. It exposes an external interface whose IP address is used as a forwarding address in anycast announce. It enforces chaining decisions as follows. It forwards incoming packets to the intended vSF instance, based on the encapsulation header. Once the packets have been processed, the vSF forwards them back to the connector, which enforces a forwarding decision toward the next vSF instance location (i.e., its connector) according to the service topology. These forwarding decisions are cached in the connector, indexed by a hash computed using flow-related information.

The connector also exposes a control interface, used by the D-MANO, to populate the service-aware routing table and the mapping between service function and vSF instance unicast address. This information is used by the connector to enforce chaining decisions for outgoing traffic, and locally balance the load among the vSF instances that provide the same service (same prefix).

Virtual Service Function

vSF instances process service flow packets according to the service they provide. Once a packet has been processed, the vSF instance updates the chaining encapsulation header to point to the next service. Each instance is monitored and managed by the D-MANO.

Distributed MANO

The D-MANO controls and manages the other NFV-R's components. It is configured with high-level policies, which guide its autonomous orchestration decisions. It has three essential control functions (illustrated in Figure 3.4). The first one consists in monitoring vSF instances, deriving from them vSF costs and the second one in injecting such costs in the IGP, via the router. The third function consists in getting IGP information from the router to build the service plane topology, computing the service-aware routing table and then pushing it in the connector. In our works, we tested two class of routing decision both distributed and taken hop-by-hop: single path routing algorithm which maps each vSF type to a unique destination (e.g., Shortest Path First) and multipath routing algorithm which associates multiple destination for a single type of vSF (e.g., ECMP, WCMP). With the second class, each destination is mapped with a metric and the routing decision is made in the dataplane (i.e., the connector) often using weighted hashing mechanisms. While the first class would be prefered to minimize a given metric (e.g., latency), the second exhibits better load balancing properties among vSF instances.

3.4 Implementation

We have implemented our proposed solution, which we describe in the present section. In this section, we include the technical choices we made for each component of the architecture described in the previous section.

3.4.1 System-Level Choices

Encapsulation header

Our implementation uses the *Network Service Header (NSH)* to steering the traffic through the different services [84]. Figure 3.5 presents the NSH header encapsulation format. Our choice is motivated by the fact that NSH is an

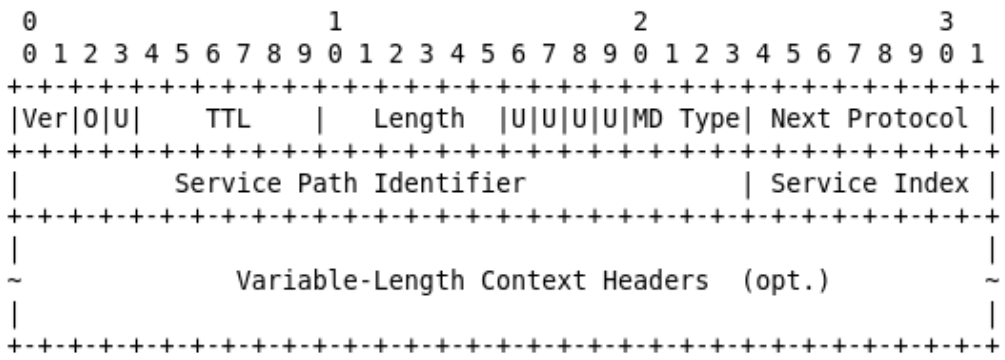


FIGURE 3.5: NSH encapsulation header.

IETF standard explicitly designed for service chaining and is widely used in many open-source frameworks (e.g., [29, 75–77]). In NSH, the Service Path Identifier (SPI) field uniquely identifies a set of abstract service functions (i.e., the Service Function Chain), while the Service Index (SI) points to the next function the packet has to be delivered to in the SPI set. NSH also provides extensible metadata fields that we leverage to convey the hash value used to consistently identify a flow along its chain. Such hash value is computed at the classification step with the 5-tuple of the original packet. On the contrary to the original packet 5-tuples which could be modified by a vSF (e.g. by a NAT) and would fail to identify a flow along the chain, this field is not modified since it is not processed by the vSFs.

High level policies define the mapping between the service function chain, and the SPI as well as the vSF with the SPI, SI tuple. The D-MANO populates this mapping on the connector to properly identify the next vSF type for any packets to be forwarded. Figure 3.6 represents this mapping. Notice that the same vSF type can be identified by multiple SPI,SI tuples. Indeed a type of function could be mandatory in multiple service chains. For instance, a firewall could be present in several security related chains. Once the next function is properly identified, it can be mapped to the next destination corresponding to the location of the next vSF. We detail below the structure of this service-aware routing table and its computation process.

Interior Gateway Protocol

As IGP protocol, we use *Open Shortest Path First (OSPF)*, since it is widely used and easily extensible, thanks to opaque Link State Advertisements (LSA). Opaque LSAs are leveraged to share information about vSF instances and links. Even if flooding opaque LSAs increases control traffic overhead, it

NSH to vSF Mapping		
SPI	SI	vSF
10	3	Firewall
10	2	IDS
5	2	Firewall

FIGURE 3.6: Mapping between the NSH fields and the associated vSF type. This mapping is implemented in the connector and populated by the D-MANO based on high level policies.

vSF to next hop	
vSF	Next Hop
Firewall	192.168.0.1
IDS	192.168.0.2

(A) Single path routing table.

vSF to next hops		
vSF	Next Hop(s)	Metric
Firewall	192.168.0.1	50
	10.0.4.6	50
IDS	192.168.0.1	10
	192.168.0.2	40
	10.0.3.1	50

(B) Multipath routing table.

FIGURE 3.7: Service aware routing table. Maps vSF type to the next hop(s).

does not affect OSPF stability, since they do not trigger shortest path re-computation. We use *vSF opaque LSAs* to convey 3 pieces of data: *i)* the anycast address of a vSF instance, *ii)* the associated vSF cost, and *iii)* the NSH endpoint IP address (i.e., the IP address of the next Connector). In our initial implementation, we choose to use a simple vSF metric: the remaining processing capacity of the vSF instance. The NFV-Rs use the provided information to build a graph linking vSFs and NFV-Rs, each link weighted with the associated cost (Fig. 3.3). In a second implementation, we have announced with the IGP a metric indexed with the number of packets processed by the vSF. Thus, each NFV-R is able to build the service plane topology based on the information shared via OSPF.

Service-Aware Path Computation Algorithm

Based on the information shared through the IGP we compute the *service-aware routing table* (see Figure 3.7). This table maps vSF types (that can be

identified by NSH SPI and SI fields) with the location of instances of these vSFs. This location corresponds to the next hop in the chain. A vSF type can be associated with one or multiple next hops depending if we use single path or multipath routing. In single path routing, every flows forwarded by a connector to a vSF type are sent to the same next hop (Figure 3.7a). In multipath routing, multiple instances can be populated in the connector as next hops (Figure 3.7b). When multiple paths are considered, incoming flows are forwarded consistently to different instances based on the routing policy (often based on a metric). We have implemented two routing scheme in our NFV-R. The first WCMP is a multipath routing decision while the second is the widely known Shortest Path First, which enforces single path per vSF type.

In our first implementation, we choose to use *Weighted Cost MultiPath (WCMP)* [118] to compute nodes' service-aware routing table. It is particularly suited for our anycast-based approach as it allows balancing the traffic based on the vSF cost. As illustrated on Figure 3.3, we use network link costs and vSF costs to weight the paths to a vSF anycast address. In this example, we show the service topology as seen by node A. It is easy to see that the cost to reach the *FW* instance on node B is 30 and to reach the one on node D is 25. WCMP combines network and vSF cost in order to assign weight to the different vSF instances. This weight corresponds to the probability for a new flow to be sent to a given vSF instance. Since the vSF cost is regularly updated, WCMP adapts to the load by distributing the traffic on the lightly loaded instances (i.e., lower cost, hence, higher WCMP weight).

In a second implementation, we choose to use a shortest path algorithm to take into account the network and vSF metric conjointly. Based on a service view illustrated on Figure 3.3, a shortest path algorithm is used to build the service-aware routing table. In Sec. 3.7 we discuss more about the metrics.

3.4.2 Node-Level Choices

We build our NFV-R using Linux and use network namespaces to isolate the components. We first propose an implementation using Mininet. Mininet was used to orchestrate the different namespace and build the network topology. While this solution was sufficient to test the functionality of our solution on small topologies (e.g., 4 nodes), it was poorly scalable. Indeed, with Mininet, the NFVRs were sharing the same CPU and memory resources on a single machine creating a bottleneck for our emulation experiments. To face

these limitations, we choose to package the NFVRs in LXC container to deploy them on separate machines and build the topology with virtual links between the physical machines (see 3.6). In the containers, logical components were still isolated by network namespaces as described below.

Router

In our implementation, we use *FRRouting* [99], an open source IP routing protocol suite, to implement our OSPF router. In particular, we use the OSPF API offered by FRRouting to mirror the Link-State Database (LSDB) in the D-MANO and to inject vSF opaque LSAs.

Connector

We implemented the connector logic in *P4*, a language for programming the dataplane [10]. P4 has been chosen for several reasons: it can support any header (e.g., NSH) and it is stateful (registers), allowing us to cache routing decision so to handle flow affinity. Our P4 code is run on the simple_switch target [78]. Its runtime CLI is exposed to the D-MANO to configure the switch and populate the service-aware routing table at runtime.

Virtual Service Function

vSFs are implemented as simple processes (using *scapy* [90]) parsing incoming packets, decrement their NSH SI field, and forward them back to the connector. The focus of the initial implementation being on the different components of the proposed approach, we purposely choose simplistic vSFs for the time being. The Python *psutil* library enables us to monitor the resources used by the vSF processes.

D-MANO

The D-MANO has been implemented in Python. Its main loop runs as follows. First, it polls the resource use of the local vSF instances to build the related costs. The costs are then announced on the network with vSF opaque LSAs. Second, the D-MANO gets the vSF announces from its mirrored LSDB. With these data, it builds a service view (see Fig. 3.3b). Based on this topology, it computes the service-aware routing table.

3.5 Functionnal Evaluation

In this section, we first demonstrate that our solution succeed in steering traffic through service function chain. We show on a small topology that NFV-R can efficiently balance traffic among multiple vSF and that it easily supports the addition of new vSF instance on the network. When a vSF is added in the network, part of the traffic is forwarded towards it without any modification of the nodes made by an operator or a centralized orchestrator.

3.5.1 Evaluation Methodology

In this section, we evaluate a simple scenario to show how we can achieve load balancing on different vSF instances of the same type by using the proposed solution. We selected the parameters in order to assess that our system successfully steers traffic through the target service chain.

We consider a network topology that looks like Figure 3.3b, except that we use one single generic service. The link cost between NFV-R A and NFV-R B and the link cost between NFV-R A and NFV-R C are set to the same value. Link cost between NFVR-R A and NFV-R C is set to a different value so to be less preferred than the previous ones. All the vSF instances have the same initial capacity (i.e., vSF cost). It is the maximum number of packets per second a vSF instance can process, normalized so to have the same range of values as the link costs. We use Mininet to emulate this topology [63].

Traffic has to go through one of the vSF instances and then toward the egress. We generate constant bit-rate flows on a source connected to Node A. Each flow lasts 50 seconds and consumes 2 of processing units at the vSFs. The arrival rate is of two flows per second. Our scenario evolves in 2 phases. *Phase 1:* only the vSFs on NFV-R B and NFV-R C are running. *Phase 2:* After 150 seconds, a third vSF is instantiated on NFV-R D, leading to traffic redistribution.

3.5.2 Evaluation

Figure 3.8 presents the traffic distribution over time on the vSF instances. Since each flow lasts 50s, during the first 50s of the experiment, the system load rises until it reaches its steady state. Note that, in this example, a measure of each vSF load is measured and advertised every 2s. We can see that, during the first phase, each vSF instance receives in average the same amount of traffic. Indeed, they do have the same network cost from the ingress point

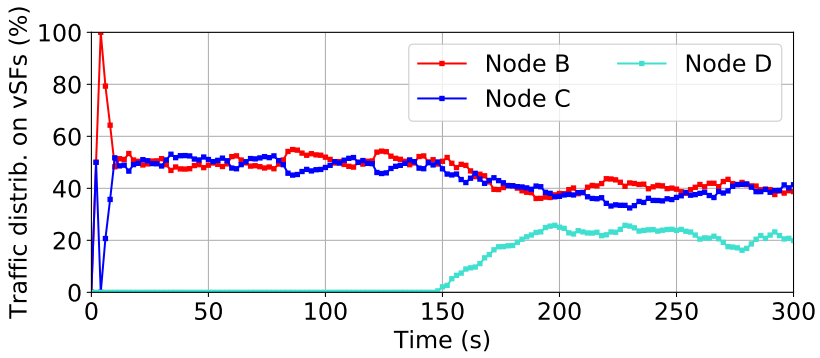
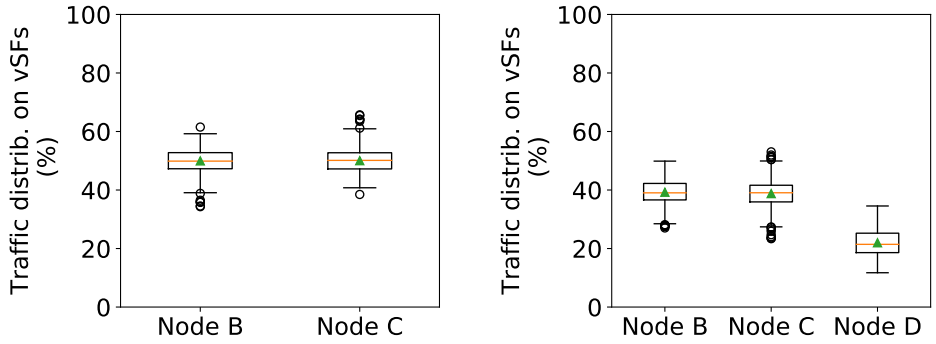


FIGURE 3.8: Traffic distribution over time on the vSF instances. During the first 150s only two vSFs are running. At $t = 150s$, a third vSF is instantiated.



(A) Phase 1 (50-150s). (B) Phase 2 (200-300s).

FIGURE 3.9: Mean traffic distribution on the vSF instances during the two phases.

of view and the same initial vSF cost. Once Phase 2 starts, after the 50 seconds of transition, which lasts between $t = 150s$ and $t = 200s$, a new steady state is reached. Now the vSFs on Node B and C, each process 40% of the traffic, while the vSF on Node D roughly processes 20%. This distribution of traffic corresponds to the WCMP weights that consider links' cost and vSFs' cost.

Figure 3.9 presents the mean traffic distribution on the instances for the steady state of the two phases of the scenario. They result from 20 runs of the experiment. We can observe that our solution is able to balance the load among the available vSFs. The mean and median loads are centered on the values we can compute from WCMP: 50/50% in Phase 1 and 40/40/20% in Phase 2. In addition, 50% of the loads are less than 3 points from the median value, while the max and min values are at most 10 points from it. Such limited variation shows that the system remains quite stable.

3.6 Large Scale Evaluation

In a second time, we tried to evaluate our NFV-R on a more realistic setting. To make our prototype deployment scalable and build large scale emulation, we leveraged on the Grid'5000 platform presented below. We consider a scenario where service function chains are deployed on the WAN of an Internet Service Provider. When a flow is routed from a source to a destination, it has to be analyzed by a chain of security functions (e.g., a firewall, an IDS...). NFV-Rs are deployed on this topology and hosts vSFs. Flows are then routed through the adequate chain of service as explained in the previous sections. Our experiments, in addition to show the scalability of our proposal aims at evaluating the introduced overhead in the IGP signaling as well as the induced traffic distribution on realistic topologies.

3.6.1 Evaluation Methodology

Dataset

We use three ISP topologies that were previously used in [43] and made publicly available [22]. They are summarized in Table 3.1. The first one was synthetically generated, while the two others were inferred in the Rocketfuel project,which has provably build realistic and accurate map of ISP topologies [95]. They include path delays. We use the weights provided with the dataset to configure the IGP link costs. We consider all the nodes as NFV-Rs able to host vSF instances. The dataset also contains demand matrices that we use for service demands.

We randomly select 5% of the overall demands to build our service requests (ingress, egress, bitrate), as in [43], which represents our baseline traffic intensity. We run two types of scenarios. In *scenario 1*, all the requests have to be steered through one vSF. Only one vSF type is present on the network. There are 10 instances of it. In *scenario 2*, all the requests have to be steered through a vSF of type 1 and then a vSF of type 2. There are 5 instances of each type. In all the scenarios, the vSF instances are placed on the nodes that have the highest betweenness centrality, i.e., the nodes traversed by the highest number of IGP shortest paths. Such selection criteria has been shown to be efficient for vSF chaining in centralized approaches [98].

TABLE 3.1: Evaluation dataset.

Names	Nodes	Edges	Demands	Type
RF1755	87	322	7527	Rocketfuel inferred
RF3967	79	294	6160	Rocketfuel inferred
SYNTH50	50	276	2449	Synthetic

Grid'5000 environment

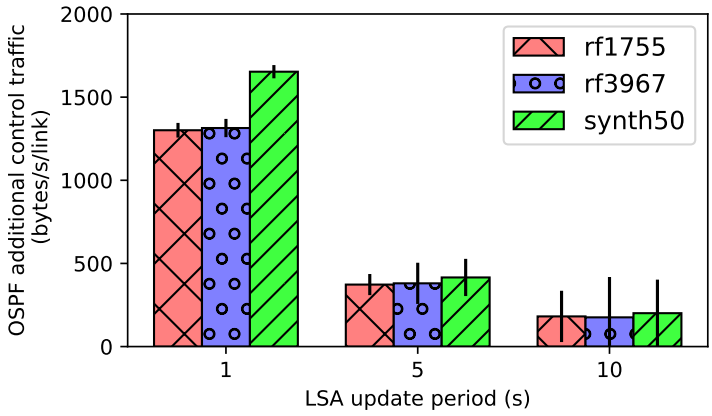
We deployed emulated topologies of NFV-Rs on the Grid'5000 testbed [7]. Grid'5000 is a large scale and versatile testbed, which provides access to a large number of resources (12000 CPU cores) distributed on different sites and interconnected by a 10Gb/s WAN. This testbed is highly reconfigurable, which makes it a great tool for experiment-driven research. In our experiments, we use Distem [24], a network emulation tool, to deploy NFV-R LXC on bare-metal servers. For each NFV-R, Distem uses Linux cgroups to allocate 4 vCPUs (i.e., 4 CPU cores) to each NFV-R. Distem connects NFV-R with VXLAN tunnels to emulate the topology links. We run our experiments on a cluster of 48-nodes with the following host main characteristics: Intel Xeon E5-2630L v4 (Broadwell, 1.80GHz, 2 CPUs/node, 10 cores/CPU), 10Gb Ethernet interface. We deployed the topologies and scenarios with these tools.

Traffic is generated at the granularity of a UDP flow. We fix flows arrival rate, duration, and packet size. The packet rate of each flow is then accommodated to correspond to the demand's bandwidth in bits per second. Once a first flow duration period has elapsed a steady-state is reached. At this point, each demand in the dataset is constituted of k UDP flows. This value corresponds to the flow duration divided by the arrival rate, which we set to 50 seconds and 2 new flows per second respectively. Thus in our experiments, each request is constituted of 100 different UDP flows. Even if this uniform traffic distribution is simplistic, it gives a first assessment on our system behavior.

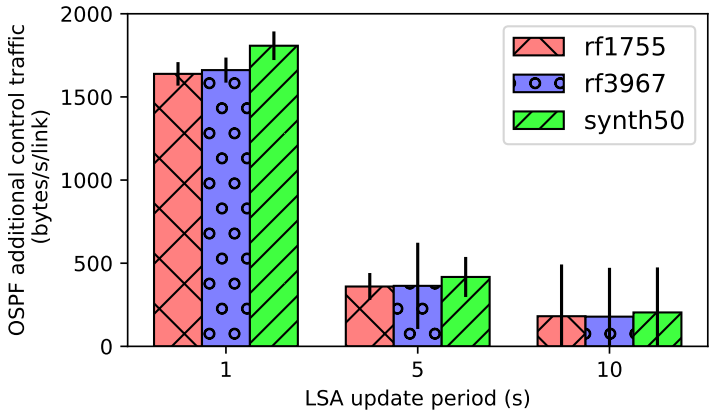
3.6.2 Evaluation

We now present the results of our large-scale experiments. We compare different update periods for vSF LSAs and discuss the tradeoff between the network traffic control overhead induced by these LSAs and the network dynamics.

Fig. 3.10a and Fig. 3.10b show the additional overhead generated by NFV-Rs with one and two types of vSFs respectively. In both cases, there are at



(A) 1 vSF chains.



(B) 2 vSF chains.

FIGURE 3.10: OSPF overhead induced by NFV-Rs with various LSA update periods.

all 10 instances in the network. Logically, when the frequency is low, the overhead is low. We can also observe that it is slightly higher with two vSF types than with one. Indeed, the routers have to propagate distinctive LSAs.

The control overhead has to be discussed with the dynamics of traffic steering. Indeed, a low LSA period results in a less accurate view of the network at routers. Fig. 3.11 shows the traffic distribution when there is one vSF type on the largest topology. We can see that the load is well distributed on the instances, considering also OSPF weights. When the LSA period increases, the traffic distribution tends to be less stable but remains quite fair. Fig. 3.12 presents the results when there are two vSF types. The load doubles on the instances as there are five of each type. We can also observe that the spread slightly increases with the increase of the LSA period. Finally, we summarize in Table 3.2 and Table 3.3 the experiment results for the two other topologies with one and two vSF types respectively. They conform to the

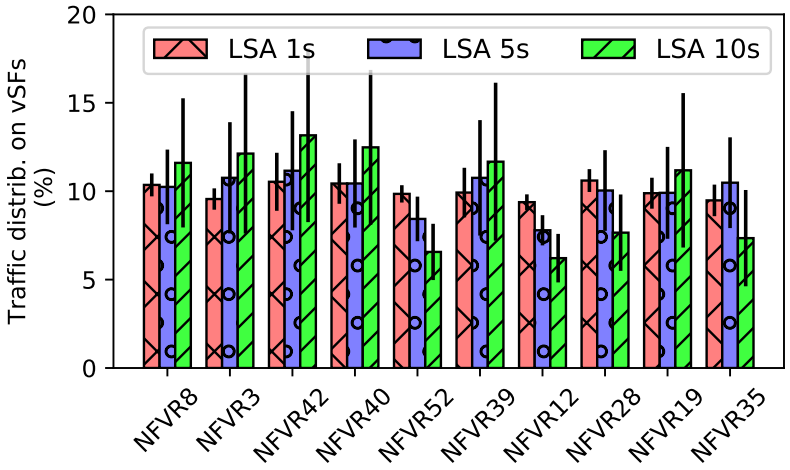


FIGURE 3.11: Traffic distribution over vSF instances with various LSA update periods for 1 vSF chains on the RF1755 topology.

above observations.

We also study the impact of the LSA period on link use. Indeed, compared to the analytical model, the Grid5000 emulation introduces: (i) the update time (measurement and propagation) of vSF metrics and (ii) the traffic variation over time. Fig. 3.13 shows the max link load on the three emulated topologies. We can see that the deviation is extremely small on rocketfuel topologies especially with chains of 1 vSF. It increases lightly with the LSA update period and the number of vSF in the chain, meaning that the paths to NFV-R tend to go through the same core links. On the contrary, the link load values tend to be less stable on SYNTH50 topology. Indeed, this topology is closer to a mesh one, meaning that shortest paths are more likely to go through different links. In the scenario with chains composed of 2 vSF, the links experience congestion. In this paper we study the normal behavior of our system, as a consequence, no bandwidth limit has been set on the interface (explaining the load superior to 100%). We leave for future work the study of the congestion impact on our system and how to tackle this challenge (which could impact the LSA flooding process). We can also observe that the link use values are higher compared to the analytical ones. This is due both to the network dynamics and to the fact that OSPF does not take into account the link load on its link metric.

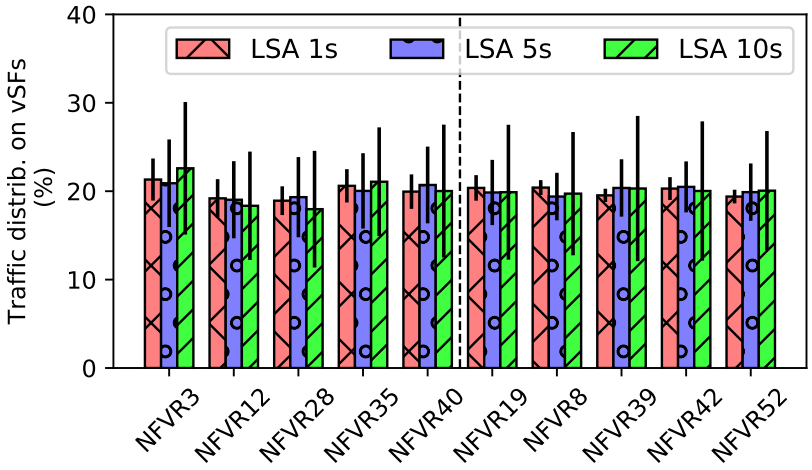


FIGURE 3.12: Traffic distribution over vSF instances with various LSA update periods for 2 vSF chains on the RF1755 topology, type 1 vSFs on the left, type 2 vSFs on the right.

TABLE 3.2: Traffic distribution (%) over vSF instances with various LSA update periods for 1 vSF chains.

Topology		**RF3967**			**SYNTH50**		
LSA Period		**1s**	**5s**	**10s**	**1s**	**5s**	**10s**
Worst NFV-R	**Mean**	11.80	12.19	11.92	10.68	11.08	14.52
	Std	0.13	2.38	3.49	2.12	3.62	7.13
Best NFV-R	**Mean**	7.34	7.78	7.79	9.54	9.05	4.25
	Std	0.24	1.23	2.61	1.80	1.81	2.78

3.7 Discussion and Perspectives

Our results illustrate that service chaining can be achieved in a distributed manner by augmenting the network layer routing and configuring autonomous nodes with high-level policies. However, while opening interesting perspectives, it opens as well a number of questions. How our solution could handle failures compared to centralize one? What metric should be used to take chaining decision and how to design it? What would be the memory overhead of such a distributed solution? We discuss these questions in this section.

Augmented Network Layer Routing

We have shown that we can steer traffic through a service function chain by augmenting the network routing layer. Nonetheless, finding feasible path with or without constraint is a hard problem [4]. Precomputed hop-by-hop

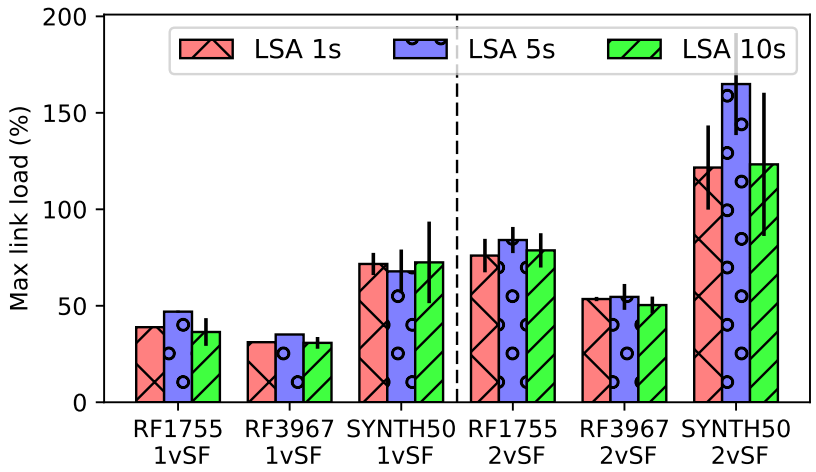


FIGURE 3.13: Max link loads on topologies for chains of 1 or 2 vSFs with different LSA update paces.

TABLE 3.3: Traffic distribution (%) over vSF instances with various LSA update periods for 2 vSF chains.

Topology		**RF3967**			**SYNTH50**		
LSA Period		**1s**	**5s**	**10s**	**1s**	**5s**	**10s**
Worst NFV-R (Type 1 vSF)	**Mean**	22.00	21.55	21.49	20.55	26.70	22.93
	Std	5.50	6.04	5.71	3.93	6.18	6.65
Worst NFV-R (Type 2 vSF)	**Mean**	24.08	24.15	23.27	20.22	20.88	21.11
	Std	0.28	1.48	0.23	2.20	5.42	8.97
Best NFV-R (Type 1 vSF)	**Mean**	18.31	18.16	18.70	19.57	17.91	18.18
	Std	1.50	4.36	4.42	1.50	3.52	5.52
Best NFV-R (Type 2 vSF)	**Mean**	18.31	18.16	18.71	19.57	17.91	18.18
	Std	0.90	2.06	1.14	1.91	4.75	6.08

routing decisions could follow simple and fast heuristic (shortest path to the next vSF) to steer best effort traffic. Yet, we believe that flows, which require QoS guarantees (e.g., VoIP), would be best served using the source routing paradigm (e.g., with segment routing) so to enforce on-demand optimized path. Even if our service plane topology provides support for both approaches, such hybrid scenarios and related tradeoff need further investigation.

Multi-domain SFC

In this paper, we define an augmented IGP routing logic to provide distributed SFC decisions. This is a first step towards the design of multi-domain services. Indeed, our proposal can be extended to inter-domain routing with BGP [88]. With the use of communities [64], operators could

choose the information to share to build multi-domain SFC thus opening new business opportunities. Work at the IETF also propose to leverage on BGP Address Family Identifier and Subsequent Address Family Identifier to announce service function reachability [28].

Reaction to Failure

Our proposal can leverage on existing work to support vSF maintenance, failure or even chain modification. Indeed, with our approach, maintenance can be easily handled through any existing loop-free graceful shutdown mechanism [32]. In case of vSF failure, the NFV-Router locally detects it and does not announce anymore the associated anycast prefix. Since our system relies on IGP, it can converge in a few hundred milliseconds [33], making the failed vSF instance unavailable for new flows. In comparison, a centralized solution would add a non negligible delay because of the need to send a failure notification to the central controller and receive the recovery action to be performed. Furthermore, some vSF state migration use-cases can be locally dealt with on NFV-Rs ([57, 86, 112]). However, service management operation involving several NFV-Rs is more challenging, since the state (vSF session state, routing cache entries...) has to be coordinated. Such operations are needed when a service is modified (e.g., suspicious flows redirected to an IDS), or when a vSF is migrated to a distinct NFV-R. Existing work [114], which can be used in our case, identified challenges and possible solutions to keep end-user sessions alive during reconfiguration.

Virtual Service Function Cost/Metric

To take service-aware routing decisions, two different types of entities are involved: network links and vSF instances. While assigning a cost to a link is straightforward (based on bandwidth, latency, etc.), the *cost* of a vSF instance is an open research area. This cost may be based on a plethora of vSF state parameters [16, 71], but should also be in the same order of magnitude of the links' metric. More importantly, it has also to be additive, so to guarantee loop-free even when considering multiple constraints [54, 106]. Such a cost could either be statically defined at network configuration, or dynamically updated based on vSF state (e.g., CPU use, incoming packets per second ...). The first, even if fine-tuned with traffic engineering algorithms, leads to static chain instances, prone to congestion in case of bursty traffic. The second builds service paths based on regularly updated vSF metric, and can better

adapt to quick traffic variations, at the risk of route flapping. Nonetheless, this instability can be mitigated by making routing decisions at the flow level and fine-tuning metric update rate [92]. The evaluation of our flow routing solution confirms these results by comparing the vSF load distribution with different LSA update frequencies.

Service Path State

In our approach, we do not keep flow state on every switch [30, 35, 36, 79, 116], neither encode the whole flow path in the header [1, 114]. Instead, we only define in the header the set of service functions in the chain, then each NFV-R along the path will choose and store the next service hop for a flow it processes. This solution offers a trade-off between the flow state stored in the network infrastructure and the packet header overhead introduced by segment routing. Even if this memory overhead is limited to NFV-R, this may raise scalability issues in case of a large number of flows. Our route caching system can be assimilated as a flow table: a set of matched packet field is mapped to a destination. Research has shown how to cope with a large amount of flow state [35, 50, 83]. It is possible to apply such techniques to our solution, so to minimize the memory usage.

Distributed Orchestration Decisions

NFV-R decouples traffic steering from orchestration. However, it hardens vSF resource allocation problem, which is already difficult when decisions are taken by a centralized orchestrator [45]. Even if distributed approaches like ours improve the architecture resiliency and scalability, how autonomous nodes could take orchestration decisions (e.g., instantiating a new vSF instance to balance the global load) using on our augmented topology is another problem to be tackled.

3.8 Conclusion

In this chapter, we have made the case for orchestrating service chaining in a distributed manner. We proposed to augment the network layer routing by using anycast addressing for vSF so to build what we call the service topology, allowing embedding service chaining into routing. By doing so, our solution can rely on the robustness and scalability of IGPs to steer traffic into service chains. We designed our NFV Router whose architecture is based

on this concept and we implemented a first prototype. A first evaluation performed with our implementation shows that flows can be successfully driven through the chain of services according to available resources. A second implementation on Grid'5000 testbed emulates large scale ISP topologies, thus proving the scalability of our solution. The evaluation shows that our solution introduce a small control traffic overhead and efficiently distributes traffic on multiples service function. We have also shown that service function chaining tends to have a significant impact on link use compared to classical destination based shortest path. Our approach sets itself apart from previous work, and as such, it still needs to be thoroughly investigated. To this end, we provide a research agenda highlighting the different aspects that need to be tackled. However, what comes out as well is quite promising and opens interesting perspectives.

One interesting problem to tackle is to compare the quality of the chaining decision. Indeed, the chains can either be built by autonomous nodes taking hop-by-hop decision to the next vSF in the chain or by a central controller which could optimize the path for the whole chain. Even if the first solution is more resilient, since it does not add a new single point of failure, it may introduce worst performance. In this chapter, we have focused on a chaining decision taken hop-by hop. Autonomous nodes, to build their own service-aware routing table, used our augmented service view. Yet this view could also be leveraged by a central controller to compute end to end service function chains. This controller could enforce these paths either by populating the NFV-R routing table or by leveraging on the source routing paradigm to steer the flows through the adequate sequence of vSF. In the next chapter, we will analytically compare these two chaining decisions.

Chapter 4

The cost of distributed decision in Service Function Chaining

> Combien de fois
> abandonnons-nous notre chemin,
> attirés par l'éclat trompeur du
> chemin d'à côté ?
>
> Paulo Coelho

Service Function Chaining decision used to be centrally taken by a controller who would compute the path for the whole chain. The NFV-Router presented in the previous chapter open the way to a different path computation approach: the path can be computed segment by segment in a hop-by-hop manner. In this chapter, we present the second contribution of this thesis, which directly arise from the introduction of the *NFV-Router*. In the following sections, we evaluate the fundamental performance differences between a centrally taken chaining decision and a hop-by-hop one. We propose an analytical formulation of this problem to explore the different tradeoff between these two types of decisions.

This work have been incrementally presented and published in the following conference and journal: the *IFIP Networking Conference* (2019) [109] and the *IEEE Transactions on Network and Service Management Special Issue on Softwarized Networks* [107] (2019).

4.1 Background and Motivation

In Chapter 3, we have presented a solution, which can rely on a distributed decision taken hop-by-hop to build service function chains. Even if this architecture is more resilient and robust, it may introduce worse performance

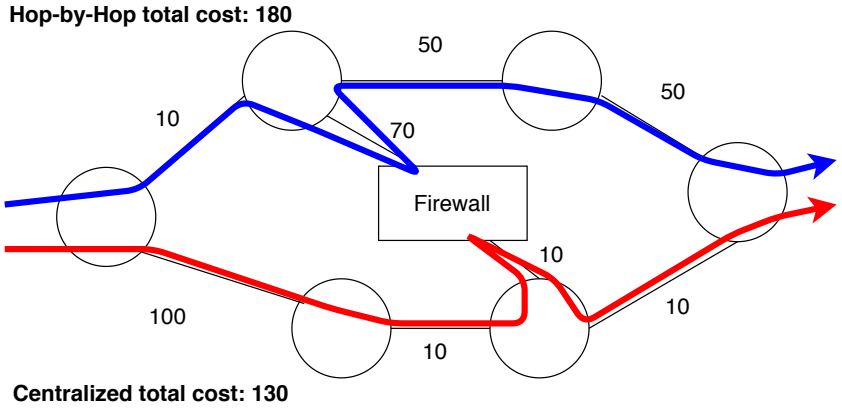


FIGURE 4.1: Comparison between a hop-by-hop and a centralized chaining decision. The hop-by-hop decision minimizes the cost from the source to the firewall and then from the firewall to the destination. The centralized decision minimize the cost for the whole chain.

compared to centrally taken decision. Indeed, for any metric considered, a hop-by-hop decision only routes the packet segment by segment. Thus, it only considers the metrics to reach the next vSF in the chain and does not take into account the next hops and the destination to reach. Therefore, hop-by-hop routing can suffer detour.

An example of this behavior is presented in Figure 4.1, which represents the service aware topology. Two incoming flows are routed from a source (on the left) to a destination (on the right) and have to be processed by a Firewall. Two instances of the Firewall are present, one on the upper path with a vSF metric of 70 and one on the bottom path with a cost of 10. Two policies are compared: a shortest path taken hop by hop (from the source to the firewall and then from the firewall to the destination) and an end-to-end shortest path, which can be centrally computed. In this situation, the hop by hop decision route the incoming flow through the upper path since the first segment is cheaper. Indeed the cost from the source to the Firewall is 80 for the upper one compared to 120 in the second. The centralized decision takes the bottom path since the overall cost is cheaper (130 vs 180). This simple example shows that a hop-by-hop decision due to its myopic view can route traffic through non optimal path. This decision could then introduce higher latency or even congestion.

Then should the centralized decision be preferred? While convincing, this example only shows that hop-by-hop decision *may* take different paths compared to a centrally computed one. Yet, this difference seems to be highly topology dependent, and may only be a corner case unusual enough to be neglected. In this chapter, we compare the inherent differences of service

function chaining when tackled centrally or in a distributed manner. We first propose two Integer Linear Problems to model central and hop-by-hop decisions. We then leverage on realistic topologies to compare these decisions in term of cost, path length and link usage. We also quickly compare the reactivity to an event (such as a link or vSF failure) for a centralized or distributed decision. We show that (i) distributed chaining decisions are close to optimal centrally-computed paths, and (ii) a distributed system, because of its local control loop, has a faster reaction to network events than centralized solutions.

The rest of the chapter is organized as follows. First we describe in Sec. 4.2 the modelization we use. We detail in Sec. 4.3 the centralized optimization problem we considered and its distributed counterpart. We then describe the evaluation methodology we used in Sec. 4.4 and expose our results in Sec. 4.5.

4.2 Network Modelization

In this section, we detail how we model the network and the SFC routing problem. We consider a network where NFV-Routers are present and host vSFs. These vSFs are already placed. Each of these vSF is associated with a metrics that is function of its current load. The link are also associated to a metric that represent the network part. As explained in the previous chapter, we consider that these metrics are additive and load sensitive. You can refer to Section 3.7 for more discussion on the metrics that could be considered. We choose to limit our model to simple linearly dependant metrics to more accurately represent the scenarios presented in Chapter 3 and compare centralized and distributed decisions.

SFC Routing Model Parameters

We formulate as an Integer Linear Program (ILP) model the SFC routing problem. The notations for the variables and parameters are summarized in Table 4.1. The network is represented by a directed graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$, where $\mathcal{N}$ is the set of nodes (classic routers and NFV-Rs) and $\mathcal{E}$ is the set of edges. $\mathcal{P}$ represents the subset of nodes that are actually NFV-Rs. $\mathcal{R}$ represents the set of service requests. The set of different vSF types is depicted by $\mathcal{V}$. On our topology, vSF instances placement is represented by the input $l_{v,p}$. We describe a SFC request with the following parameters: i_r the ingress

TABLE 4.1: Notations.

Parameters	
G	Graph representing the network
$\mathcal{N}$	Set of routers and NFV-Rs
$\mathcal{P}$	Subset of $\mathcal{N}$ representing the NFV-Rs
$\mathcal{E}$	Set of links
R	Set of service requests to serve
$\mathcal{V}$	Set of available vSF types
$l_{v,p}$	Boolean. An instance of vSF v is located on NFV-R p
c^{n_1,n_2}	IGP link cost between node n_1 and node n_2
q^{n_1,n_2}	Link capacity between node n_1 and node n_2
u_r	Number of vSFs in the service chain of the request r
i_r	Ingress node for the request r
e_r	Egress node for the request r
b_r	Bandwidth used by the request r
v_r^i	i^{th} vSF asked by the request r
Decision variables	
$x_r^{i,v}$	Boolean representing where v_r^i is placed
y_r^{l,n_1,n_2}	Float representing flow from vSF v_r^{l-1} to vSF v_r^l between node n_1 and n_2

node, e_r the egress node, b_r the bitrate, $V_r = (v_r^1, v_r^2, ..., v_r^{l_r})$ the set of requested vSF types.

Cost Function

We consider two types of costs in our system: the vSF cost and the network link cost. We model these costs as follows.

vSF Cost C_p: The vSF cost represents the cost to use vSF instances. It is proportional to the requests' bandwidth, which may be expressed in packet or Bytes per second.

$$C_p = \sum_{p \in \mathcal{P}} \sum_{r \in \mathcal{R}} \sum_{i=1}^{u_r} b_r x_r^{i,p} \tag{4.1}$$

This cost could be extended by taking into account other vSF state parameters.

Link Cost C_l: The link cost corresponds to the network cost defined on the IGP. c^{n_1,n_2} is the IGP static link cost of the link n_1, n_2.

$$C_l = \sum_{(n_1,n_2) \in \mathcal{E}} \sum_{r \in \mathcal{R}} \sum_{l=1}^{u_r+1} y_r^{l,n_1,n_2} c^{n_1,n_2} \tag{4.2}$$

Note that in our model the link cost is proportional to the used bandwidth to take into account shortest paths.

4.3 Problem Formulation

In this subsection, we describe the models for centralized and distributed chaining schemes.

The goal of a centralized orchestrator is to find a path for each request, which minimizes both network and processing costs while steering traffic through the correct sequence of vSFs. The problem is formulated as follows:

Objective:

$$min\ C_p + C_l \tag{4.3}$$

Subject to:

$$\forall p \in \mathcal{P}, \quad \forall r \in \mathcal{R}, \quad \forall i \in [1 : u_r], \quad x_r^{i,p} \leq l_{v_r^i,p} \tag{4.4}$$

$$\forall r \in \mathcal{R}, \quad \forall i \in [1 : u_r], \quad \sum_{p \in \mathcal{P}} x_r^{i,p} = 1 \tag{4.5}$$

$$\forall r \in \mathcal{R}, \quad \forall (n_1, n_2) \in \mathcal{E}, \quad \forall i \in [1 : u_r + 1], \quad y_r^{i,n_1,n_2} >= 0 \tag{4.6}$$

$$\forall (n_1, n_2) \in \mathcal{E}, \quad \sum_{r \in \mathcal{R}} \sum_{i=1}^{u_r+1} y_r^{i,n_1,n_2} <= q^{n_1,n_2} \tag{4.7}$$

$$\forall r \in \mathcal{R}, \quad \forall n_1 \in \mathcal{N}, \quad \sum_{n_2/(n_1,n_2) \in \mathcal{E}} y_r^{i,n_1,n_2} - \sum_{n_2/(n_1,n_2) \in \mathcal{E}} y_r^{i,n_2,n_1} = \begin{cases} (x_r^{i-1,n_1} - x_r^{i,n_1}) \cdot b_r, & 2 \leq i \leq u_r \\ (1 - x_r^{i,n_1}) \cdot b_r, & for \quad n_1 = i_r, i = 1 \\ (x_r^{i,n_1}) \cdot b_r, & for \quad n_1 \neq i_r, i = 1 \\ (x_r^{i-1,n_1} - 1) \cdot b_r, & for \quad n_1 = e_r, i = u_r + 1 \\ (x_r^{i-1,n_1}) \cdot b_r & for \quad n_1 \neq e_r, i = u_r + 1 \end{cases} \tag{4.8}$$

The objective function aims at minimizing both the vSF cost and the link cost. Equation 4.4 ensures that the vSF instances used by the service requests are actually instantiated on the specific NFV-Rs. Equation 4.5 ensures that each request uses only 1 vSF instance to process their flow at each step of the chain. Equation 4.6 ensures that the amount of resource unit on a link is not negative. Equation 4.7 ensures that the requests routed through the link between nodes n_1 and n_2 do not exceed its capacity q^{n_1,n_2}. Equation 4.8 ensures network flow conservation.

This formulation models the decision of a central orchestrator to find the overall best paths for each request. We adapt it to also model *the distributed hop-by-hop decision* taken by NFV-Rs. Thus we split the centralized problem in different subproblems that we call segments. Each subproblem aims at finding the next destination for each request (each vSF in the chain and finally the egress).

This model introduces some light modifications in the centralized formulation presented above. Instead of solving the entire problem with the presented input so to get the final optimization for the whole chain, we split it into smaller steps, one for each vSF in the chain ($\forall i \in [1 : u_r]$). By solving each of these steps we obtain the placement of a segment of the chain (from the ingress to the first vSF, from the first vSF to the second, etc.). At each step the output of the previous step is used as an input (i.e., the chosen vSF becomes the new ingress point), considering that the step in the chain i is fixed in the constraints. Finally, at each step, we minimize the cost to reach the next vSF, defined by:

$$C = \sum_{p \in \mathcal{P}} \sum_{r \in \mathcal{R}} b_r x_r^{i,p} + \sum_{(n_1,n_2) \in \mathcal{E}} \sum_{r \in \mathcal{R}} y_r^{i,n_1,n_2} c^{n_1,n_2} \tag{4.9}$$

In the first step, we initialize the ingress point with the same input used in the central problem. At each following optimization step, we use the vSF instance placement (where $x_r^{i-1,p} = 1$) to initialize the ingress point. At the last step of the chain ($u_r + 1$), there is no more vSF to reach, thus we only minimize the cost between the last vSF and the egress point (e_r).

We implemented these ILP formulations with CPLEX Optimization Studio. In Section 4.5, we compare these two routing schemes.

4.4 Evaluation Methodology

We have run experiment that are closed to the setup described in Section 3.6 but with different goals. In our evaluation we aim to compare the chaining decisions (centralized and distributed) in terms of i) cost (as define in Section 4.3) ii) path length and iii) link usage.

For each scenario, we select a topology with 10 NFV-R hosting vSF and generate a batch of requests which represents the incoming flow on the network. Each request should be forwarded through a set of service functions

before reaching its destination. The two ILPs solutions give us a mean behavior of the placement that would occur on NFV-R. We use the following parameters to define the ILP problems and build our scenarios.

Topology

We use the two Rocketfuel topologies and a synthetic one that are publicly available [22]. The dataset provide weights that we use to configure the link cost and traffic matrix that will be suse to generate the requests.

Request Generation

We randomly select 5% of the overall demands to build our service requests (ingress, egress, bitrate), which represents our baseline traffic intensity and is used to populate the bandwidth requirements parameters. We run experiments with higher traffic intensity. In this case, we multiply the percentage of overall demands randomly selected.

Service Chain

We consider three scenario, one for each chain length. In the first, there is only one type of vSF present and the chain is of length 1. In the second, there is two type of vSFs and a chain is composed of the succession of these two types. In the third, there is five types of vSF and the chain is of length five.

Virtual Service Function Placement

vSF are placed on the node with the higher betweeness centrality. There are ten instances of vSFs. Depending on the scenario, there exist multiple type of vSF present.

These different scenarios are used to fill the ILP parameters. Notice that we run each scenario 10 times and that the same input are given to both the centralized and distributed problem. Based on the results of these optimization problems, we then compare the two decisions request per request.

In addition to these ILP comparisons, we use the path delay included with the topology to compare the reaction time to a failure of a centralized and a distributed decision.

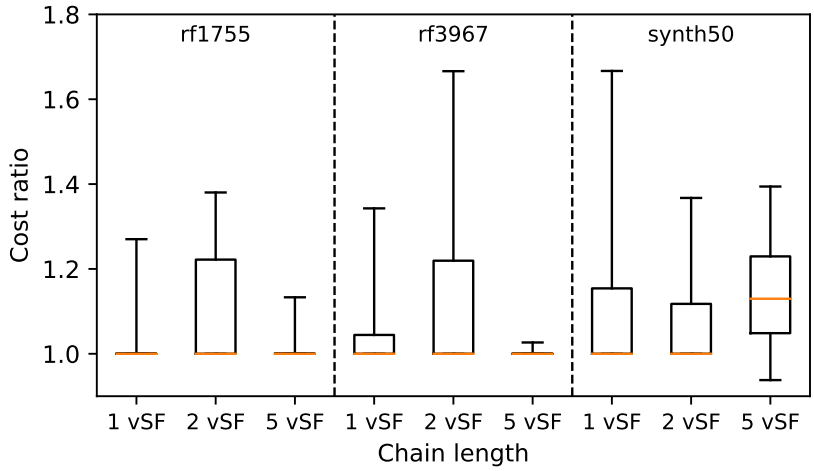


FIGURE 4.2: Cost ratio between centralized and distributed chaining decisions for chains with 1, 2 and 5 vSFs.

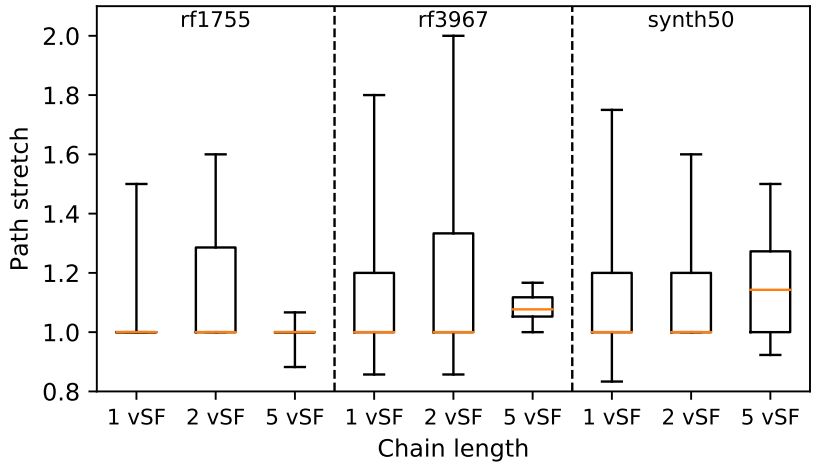


FIGURE 4.3: Path stretch between centralized and distributed chaining decisions for chains with 1, 2 and 5 vSFs.

4.5 Evaluation

In this section, we present the results of the experiments described in the previous section. We show that the distributed decision is close to the centralized one while the centralized architecture induce a stretch in the control loop reaction.

4.5.1 Network Cost and Path Stretch

First, we compare the overall cost addition and path stretch induced by our distributed hop-by-hop chaining decision scheme. For each combination of

topology and scenario, we run 100 experiments with different traffic matrices (computed as explained in Section 4.4). In each experiment, we compute with our ILP the cost of each demand and the path length in both the centralized and distributed cases. We then make the ratio of the distributed cost and centralized cost for each demand. We apply the same methodology to compute the path stretch distribution.

Fig. 4.2 presents the cost ratios. We can observe on the two largest topologies (RF1755 and RF3967) that the median costs are the same for any chain length. On these topologies, the cost ratio variability tends to grow, reaching a maximum for 2 vSFs (75% of the requests have a difference below 20%) and decreasing for 5 vSFs. This is due to two main reasons. First, the longer the chains, the more likely the distributed decisions will deviate from the centrally computed solutions (growth from 1 to 2 vSFs). Second, the lower the number of possible vSF instances per type, the lower the number of possible chain paths (decrease from 2 to 5 vSFs). We run this experiment with 20 vSF instances and chains up to 10 vSF types and observed the same behavior. Overall, the distributed decision scheme still well performs on the two largest topologies. SYNTH50 performs slightly differently. The cost ratio difference and the variability increase with the number of vSFs in the chains but remains overall below 20% for 75% of the requests. Indeed, this topology is very dense, thus there is still the deviation due to the myopic view of our distributed decisions. Nonetheless, there exist more different paths from one node to another thus reducing the path constraints for longer chains.

Fig. 4.3 presents the path stretch. We can observe similar results in the path stretch compared to the cost ratio. With chains of 1 vSF, the median path stretch is equal to 1 in every topology meaning that at least 50% of the request paths have the same length in the distributed and centralized models. Notice that the path stretch can even be less than 1 (e.g., for the RF3967 topology) since the centralized model might prefer a less costly but longer path. The stretch follows the same bell shape for the two largest topologies and the same increase tendency for SYNTH50.

4.5.2 Control Reactivity

This limited cost increase of the distributed chaining scheme is counter-balanced by the control reactivity. Indeed, while NFV-Rs take routing and chaining decision almost instantaneously since they use the augmented IGP, that is the

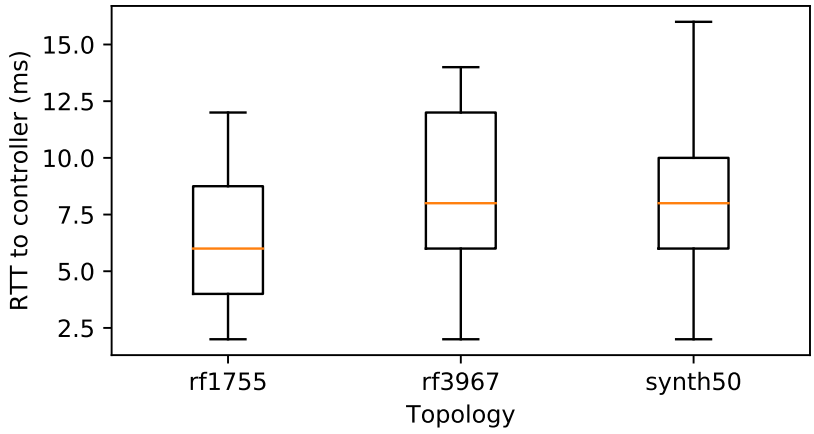


FIGURE 4.4: Centralized controller reactivity.

service plane topology, a centralized system will rely on a control node to monitor and take decisions.

On each topology, we place the central network orchestrator on the node with the highest betweenness centrality and compute the Round Time Trip (RTT) to every node on the network. This RTT corresponds to the control loop of a central orchestrator node, it first gets the data from the network nodes and then enforces a decision on it. Fig 4.4 shows the control loop latency induced by a centralized orchestration. This overhead can reach 14ms in the worst-case scenario for inferred topologies and 16ms for the synthetic one. Even if these values can seem small, they are significant for an orchestrator, which applies chaining decisions on network appliances forwarding traffic at line rate. In addition, they have to be compared to the reactivity of the NFV-Rs that is almost instantaneous.

4.5.3 Link Load

Chaining decisions also impact network resources like bandwidth use. Indeed, service chaining imposes more constraints on flows' path, since they have to reach a sequence of waypoints before being delivered to their final destination. This paradigm shift modifies network traffic engineering in two ways. First, it reduces the number of authorized paths reducing the load balancing possibilities. Second, there may not exist a loop-free path in such a situation [4]. Thus, this additional path constraints may stress the network and lead to congestion. We now analytically compare centralized and distributed decision impact on network load. We use our LP models to assess the impact of service chaining on a network resource in both centralized and

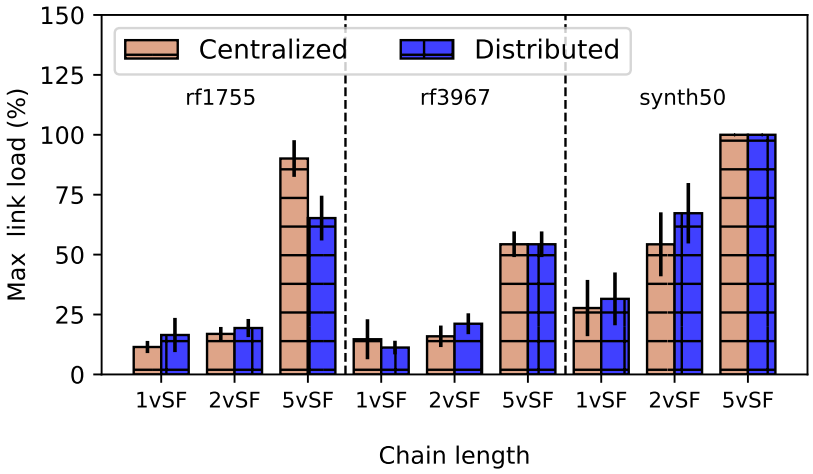


FIGURE 4.5: Max link loads on topologies for chains of 1, 2 or 5 vSFs.

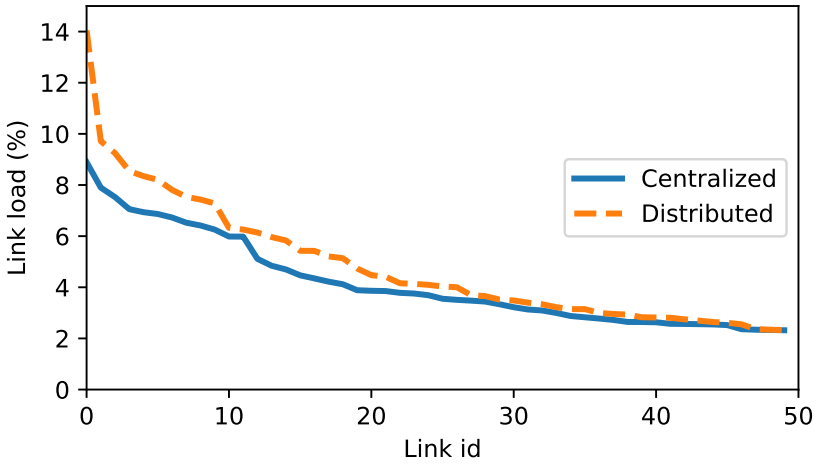
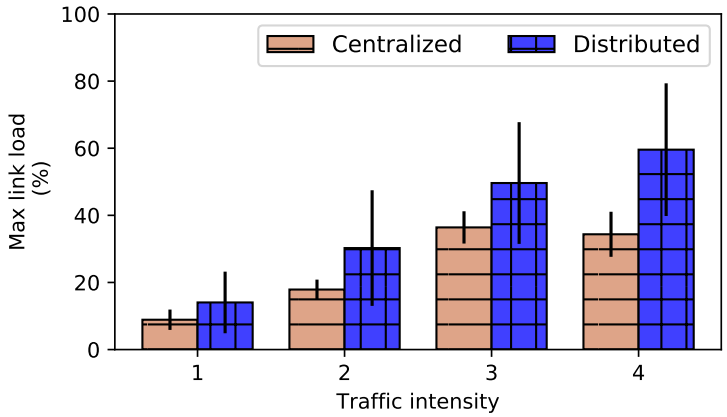


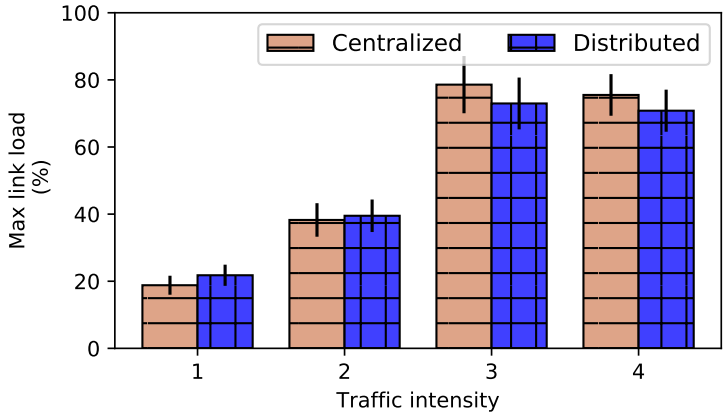
FIGURE 4.6: Top 50 more loaded links with 1 vSF chains on the RF1755 topology.

distributed schemes. Notice that link costs are inversely proportional to link capacities, thus the objective function will prefer the path with higher capacity.

First, we compare centralized and hop-by-hop service chaining models for each combination of topology and scenario. We run 20 experiments, as described in Sec. 4.4, and measure in each of these experiments the link load. We then take the maximum of these values, since traffic engineering algorithms aim at minimizing this value. We represent in Fig. 4.5 and Fig. 4.7 the mean of these maxima, as well as the standard deviation. A high standard deviation means that the maximum value varies depending on the traffic matrix while a low one means that this value is rather independent of the traffic matrix.



(A) 1 vSF chains.



(B) 2 vSF chains.

FIGURE 4.7: Max link loads with increasing traffic on the RF1755 topology.

We can observe in Fig. 4.5 that for chains composed of 1 vSF, the maximum link load is between 10% and 20% for the rocketfuel topologies and around 30% for the SYNTH50 topology. In the 2 vSFs scenario, the max link use is around 20% for the inferred topologies and around 50% for the synthetic one. With 5 vSFs, the max link load comes close to 100% utilization and reaches it in the SYNTH50 topology. Still for the RF3967 topology the max link load stays close to 60% in both centralized and distributed models. We can notice that the centralized and distributed schemes perform similarly on almost all scenarios. The only exception is SYNTH50 with chains of 2 vSFs and RF1755 with chains of 5 vSFs. In the first scenario, the max link load is significantly higher with the hop-by-hop decisions (difference of 20%). In the second one the distributed scheme performs better. Indeed, the two schemes can lead to different link utilization if the paths are different. Both models tend to choose links with the highest capacity. However, the distributed

model prefers vSF instances closer to the source (either to the initial source or the previous vSF in the chain). This difference in vSF choice lead to a different link usage. The max link load in our model is lower for the centralized model for all the scenarii except for RF1755 with 5 vSFs and RF3967 with 1 vSF. For the first, the two nodes providing the first vSF instance have similar centrality values overall, leading the distributed model to fairly balance the demands between these two instances. Nonetheless, among these two nodes, one is closer to the following vSF instances in the chain, leading the centralized model to choose paths involving this node. For the second, the mean difference is very small and is caused by close centrality values for the 10 vSF instances. Thus the load is well balanced between the vSFs, but leads to suboptimal path for the distributed model. The centralized one instead prefers to slightly increase the load on one link to find overall shortest path to the destination. Overall, the network load seems to increase with the chain length. Indeed, with these longer chains, the number of possible paths is reduced. More particularly the segment between the 2 types of vSF instances might be a bottleneck since the whole traffic has to go through a relatively small subset of paths. For instance, in the 2 vSF scenario, there exist only 25 possible shortest paths (which may not be disjoint) from one vSF type to the other, depending on the choice of the instance in a pool of 5 instances of each vSF type. Moreover, this additional waypoint increases the path length for each request in both models. Thus each path involves an higher number of links leading to an heavier traffic load on the network, stressing especially links connected to vSF nodes. This trend is confirmed with chains of 5 vSFs, where path constraints are even stronger and where the max link load increases significantly. Indeed, on the SYNTH50 topology, there is even a link which has reached its maximum capacity in both the centralized and distributed scenarios. We can also observe that the max link load values are more stable on the rocketfuel topologies compared to the synthetic one. This means that request paths tend to use different links in the latter, depending on the set of source-destination pairs in each experiment. Thus, some randomly chosen set might significantly increases the load on a particular link, while another may fairly balances the flows among the links. Indeed, this topology is more densely connected compared to the rocketfuel ones, thus requests are more likely to use different links depending on the selected requests. Notice that overall SYNTH50 has a higher max link load compared to the rocketfuel topologies since all the links have the same capacity in this dataset, which is not the case in the rocketfuel topologies where core links

have more capacity. Thus in this topology, links tend to be more loaded since a significant part of the traffic goes through the links directly connected to the vSF instances. Moreover, the increase in chain length leads to a diminution of vSF instances per vSF type, which thus receive a much more significant part of the traffic; leading to a higher link usage.

We show in Fig. 4.6 the 50 more loaded link on RF1755 for chains of 1 vSF. We can see that the centralized and distributed uses are close for most of the links. As seen in Fig. 4.5, the max link load is higher in the distributed model (14%) than in the centralized one (8%). Nonetheless, the distributed link load quickly drops to become close to the centralized (less than 1% of difference). The results on other topologies are similar: the gap between centralized and distributed model quickly reduces after the first links with the highest load.

Fig. 4.7 shows the max link load of RF1755 topology under different traffic intensity (heavier traffic matrix generation process is explained in Sec. 4.4). The scenario with 5 vSFs had no solutions for higher traffic matrix because of the link capacity constraint. We can observe on Fig 4.7a that the centralized model still outperforms the hop-by-hop one. The max link load remains under 40% even when the network load is multiplied by 4 in this scenario, while the distributed model reaches 60% of link load. On the contrary, the two schemes perform similarly with chains of 2 vSFs even when the total traffic increases.

4.6 Conclusion

In this chapter, we tried to answer the following question: what is the cost to compute the service function chain path hop-by-hop in real network ? We proposed an analytical formulation of this problem and compared a centrally taken decision to a distributed one. What we have shown is that ISP topologies constrain the centralized decision and thus tend to induce the same paths for the centralized or distributed decision. On the contrary, more exotic topologies tend to exhibit more differences in the routing decision but in general remains close to each other. The two problems show similar path cost and length as well as close max link load. Moreover, we show that a centralized solution would tend to react slower than a distributed routing process to a link or vSF failure. This work completes the one presented in Chapter 3 and makes a strong case to distribute the chaining decision since the gain of a centralized decision appears to be small.

Chapter 5

Using Model Predictive Control to Balance Service Load in Data Centers

> Qui dit équilibre dit menace qu'il se rompe. Aucune stabilité n'est jamais qu'équilibre.
>
> Maurice Druon

In datacenters, some services can receive millions of queries per second. Thus, to cope with demands, service providers require techniques to spread the incoming connections on large clusters of servers at high speed. So far, L4 load balancers have fulfilled this role. They masquerade the service IP address and spread incoming connections to the backend. Even if this solution succeeds to scale with the growth of user traffic, it fails to do it efficiently. Due to their static uniform policy (e.g. Round Robin), the servers' load is imbalanced and the infrastructure requires overprovisioning to avoid congestion.

In this chapter, we propose to leverage network softwarization to close the loop and adapt to the traffic dynamics. Inspired by our work in Chapter 3, we envision that the servers' load is feedback to a controller that will modify the load balancing decision for the next incoming connections. We focus our contribution to the control part of this system and present a Model Predictive Control approach to reduce load imbalance in datacenters.

This work has been submitted to the *IEEE Transaction on Network and Service Management* journal and is currently under revision.

5.1 Background and Motivation

Layer-4 load balancers are used to scale-out services hosted in cloud datacenters. They are capital to meet the high demand for services hosted on a pool of servers distributed among multiple clusters. A previous study has shown that an average of 44% of cloud traffic needs load balancing function, making it critical to provide clients with a good quality of experience with the least amount of resources [81]. An L4 load balancer maps connections (TCP or UDP flows) destined to a service with a virtual IP address (VIP) to a pool of servers with multiple direct IP addresses (DIPs or DIP pool).

Google [27] and Facebook [67] have recently focused on the efficiency of L4 load balancers, proposing architectures delivering services to millions of clients. While these systems succeed in keeping up with demands, they still fail to distribute the connections efficiently among the servers. Even Google's Maglev [27] (load balancing used by Google in their datacenters) was held responsible for up to 30% load imbalance on their servers. Unfortunately, this poor performance forces operators to overprovision their infrastructure leading to resource waste.

So far, these load balancers implement static and uniform load balancing policies such as Round Robin or Equal Cost Multipath (ECMP). We make the point that a significant part of servers' load imbalance is not due to these load balancers' accuracy but instead to an elephant in the room: requests' characteristics are generally not normally distributed nor follow a symmetric distribution. Instead, a small group of elephant flows/heavy hitters contributes to a significant part of the resource use [9, 20, 91]. Thus, uniformly distributing the incoming connections does not lead to uniform load distribution.

With the rise of programmable dataplanes, more sophisticated load balancing solutions have been proposed to adapt to traffic conditions [8, 48, 49]. In these solutions, the load balancing policies become dynamic: the resource state is fed back to the load balancer, which modifies its traffic splitting accordingly. Yet, these solutions rely on simple decision heuristics such as the *least loaded policy*, which requires a frequent state update to reduce the load imbalance. This additional control overhead cannot be neglected as it rises with the number of servers in a pool and with the control frequency, making it unfeasible in deployment with hundreds of load balancers and thousands of servers.

In this chapter, we propose a control-theoretic method to design dynamic

load balancing policies. We begin by formulating the load imbalance minimization problem as an *optimal control problem*. We formally define the key dynamics variables involved, as well as a concrete objective to minimize imbalance. We identify a crucial shortcoming in existing approaches: they rely solely on the servers' last state. They do not consider the current network dynamics nor their previous load balancing decisions. Yet, this knowledge can significantly reduce the load balancers' control update rate. Thus, we propose a Linear Time-Invariant (LTI) model, which links the next server load to the previous state and load balancing decision. We use this model to solve the optimal control problem and find the optimal load balancing inputs to minimize load imbalance on a finite horizon. This model can be easily estimated online with only a set of 3 counters on the servers.

Building on insights from our control-theoretic formulation, we argue that *Model Predictive Control* (MPC) [15] is a suitable class of algorithms to adapt the load balancing policy to the load dynamics. At a high level, with the servers' current state, MPC aims to predict their future state as a function of its load-balancing decision. It solves an exact optimization problem on a finite time horizon based on this prediction. It then applies the next control decision to the system. MPC is widely used in real-world problems [15]. This modular approach can support complex objectives as well as control or state constraints. Moreover, it is not bound to a model, and the same process could be used with another estimator, which may give a better prediction. Finally, it has been experimentally shown that the repeated feedback inside MPC can correct for many modeling errors [87].

We evaluate our solution by simulating a load balancer dispatching requests towards a server pool. We generate a realistic workload and compare our MPC solution with the round-robin policy under different traffic dynamics and control periods. We show that our solution outperforms the round-robin solution if the control period is smaller than the mean connection duration and can provide a systemic load imbalance reduction of 10%. Moreover, we reduce the overall imbalance with MPC, meaning that the load is more stable on the servers. Finally, we show that our solution seems to work well on short horizons, meaning that a simple heuristic could be found to solve the optimal load balancing problem based on our LTI model's parameters and the current server state. Finally, we show that our solution can perform better than the static policy even with a large control period (e.g., 1s). In contrast, comparatively, the least loaded policy exhibits disastrous performance at this control pace. Our work shows that knowing the connection duration

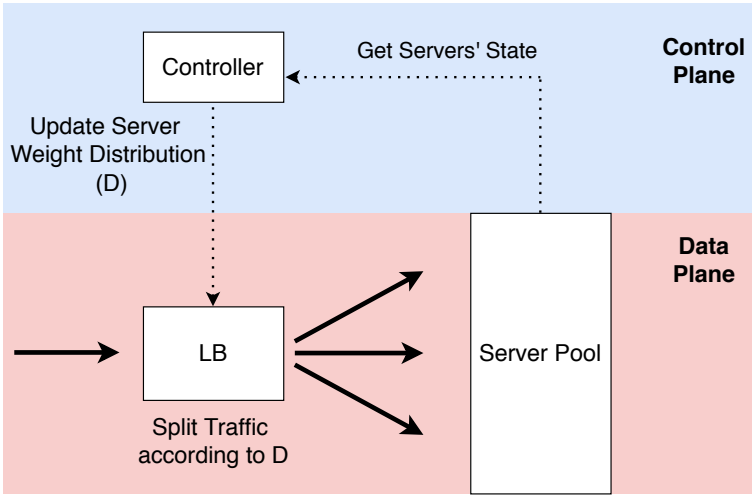


FIGURE 5.1: Dynamic Load Balancing.

distribution, one could adapt its control period to minimize the control traffic overhead.

The rest of the chapter is organized as follows. First, we begin by discussing the tradeoff introduce by dynamic load balancing and properly motivate our approach in Section 5.2. Then we formally describe a realistic load balancing architecture to build a generic optimal control problem to minimize load imbalance in Section 5.3. We leverage on this analysis to propose a Model Predictive Load Balancer, which is a control plane interoperable with real load balancers, in Section 5.4. We detail in Section 5.5 the methodology used to simulate our solution that we evaluate in Section 5.6. Finally, we conclude the paper by discussing possible extensions and limitations of our work in Section 5.7.

5.2 Control Overhead Tradeoff

Large-scale services are deployed on multiple clusters of servers located within one or more datacenters. They rely on a load balancing mechanism to (i) dispatch new incoming requests toward a service instance; (ii) guarantee that instance selection remains consistent for all packets; (iii) uniformly distribute the load on the backend servers.

In a datacenter, each service is associated with a Virtual IP (VIP), which identifies a set of instances of the service running on different servers. The server of each instance of the service is associated with a unique identifier: the Direct IP (DIP). A load balancer receives the connections towards a VIP and selects an instance of the service running on a particular server. It then

forwards every packet of this connection, in general identified by its 5-tuples, to the selected server's DIP. A large scale datacenter can be constituted of tens of thousands of servers and hundreds of load balancers. Incoming connections are demultiplexed at different stages. First, L3 load balancers (e.g., BGP routers using ECMP) dispatch incoming connections toward the same VIP onto a first set of L4 load balancers. These load balancers then select the destination server.

In such architecture, services are designed to scale with demand. Indeed each service instance has a load threshold that, when exceeded, triggers the instantiation of a new instance. Therefore, if the load is not uniformly distributed among instances, a threshold can be exceeded even if there are still enough resources on the server pool to process the workload, leading to resource waste. Similarly to other works [8, 27], we define the *imbalance of a server* as the ratio between the number of active connections on that server and the mean number of connections on the server pool. We call *systemic imbalance* the maximum value of load imbalance in a server pool. The higher this metric, the higher the resource waste.

Most of L3 and L4 load balancers use a static set of rules to spread the load on a server pool. They associate each server in a pool with a weight, the weight distribution D representing the expected distribution of incoming requests on the servers. This distribution can be uniform (ECMP [47], round-robin) or non-uniform (WCMP [118], Weighted Round Robin [104]) depending on whether the targeted resources have the same capacity or not. This static strategy uses the following heuristic: if the incoming connections follow the D distribution onto the servers, their load will also follow the D distribution. Thus, this policy assumes that the requests' characteristics (connection time, connection traffic, etc.) are normally distributed or at least close to their average value. Nonetheless, requests' characteristics have been shown to exhibit a fat tail behavior, meaning that a minority, yet a significant number of requests can account for the majority of network traffic [9, 91] or last several orders of magnitude longer than the majority of the requests [20, 91]. A static distribution of these requests can only lead to load imbalance.

Several works have recently proposed leveraging on a dynamic load balancing policy to cope with transient load burst, thus reducing imbalance and resource waste. In such architecture, the load balancer gets feedback from the targeted resources so to adapt the server weight distribution (see Figure 5.1). They leverage on the softwarization of the network control plane, as

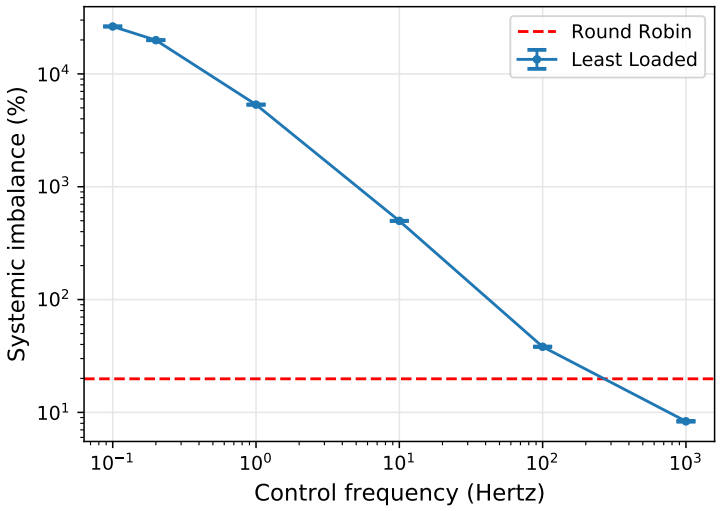


FIGURE 5.2: Static vs Dynamic Load Balancing.

well as programmable data plane to dynamically adapt load balancing policies to network dynamics [3, 48, 49, 58, 68]. While promising, these dynamic policies can only exhibit better performance if adapted to the load dynamics. Indeed, the performance of these policies is a function of (i) the server weight computation, (ii) the load dynamics, and (iii) the control frequency (i.e., the control period between two policy updates). For instance, the most straightforward implementation of this mechanism is the *least loaded policy* (used by HAProxy, Kubernetes...). This policy gets direct feedback from the server load, selects the least loaded server in the pool, and forwards the new connections toward this server solely during the next control period. In theory, this policy could optimally balance the load if it had instantaneous feedback on the server state. Instantaneous feedback can be obtained when there is a unique load balancer, but in large scale deployments, a unique load balancer would become a bottleneck for the service. Thus, multiple load balancers dispatch traffic for the same VIP and need to synchronize to enforce this policy. Furthermore, this policy can provide disastrous performance if the control loop frequency is not adapted to traffic dynamics.

We show with a toy example in Figure 5.2 that a simple uniform static load balancing policy outperforms the least loaded policy if the control frequency is too low. We consider an L4 load balancer distributing connections on a cluster of 468 servers. The load balancer receives 6,000 new requests per second, each of which has a mean duration of 10s. In Figure 5.2, we can see that even if the least loaded policy is dynamic, it is outperformed by

the simple round-robin policy in terms of systemic imbalance when the control period is higher than a few milliseconds. Such a result can be explained by two main reasons, both linked to traffic dynamics. First, since the least loaded policy directs the traffic to a unique server during a period, the period needs to be close to the inter-arrival period to route every connection optimally. Second, it requires that the load dynamics are slow, meaning that the request duration is long compared to its update period, so to deal with a stable load. Nonetheless, even if these conditions were fulfilled, this performance improvement has a cost: the communication between the servers and a load balancer controller at a high frequency leads to a significant control traffic overhead. In our setup, for a control frequency of 1,000 Hz (1ms period), the servers would send 468,000 new messages per second to the controller! Similarly, in [48], the probe period lasts a few milliseconds leading to a huge control traffic overhead.

From the limitations of both static and dynamic load balancing policies, we conclude this section by asking the following question: *"Is it possible to design a dynamic load balancing policy that outperforms static policies with a low control traffic overhead?"*

5.3 Dynamic Load Balancing

In our approach, we propose to consider the load dynamics, instead of getting high-frequency feedback from the servers, to reduce the load imbalance in a dynamic load balancing approach. In this section, we first describe the load balancing architecture that we consider. Then, we formally analyze such a system to finally propose a load imbalance minimization problem.

5.3.1 Load Balancing Architecture

We consider a web service deployed in a cloud environment. A VIP identifies this service in the datacenter. Incoming HTTP sessions first reach a set of load balancers (with the same VIP address) that dispatch them on a set of frontend instances (e.g., Apache, HHVM, Nginx). Once a session is established, the client sends HTTP requests to the server to get or modify data. The server then callbacks the backend servers to fetch the data and deliver them to the client. Each session has a different processing time that depends on the client transactions (e.g., data size, backend processing time...) and corresponds to the time from the establishment to the termination of the underlying TCP

connection. A server processes the received sessions in parallel. Its load can be estimated by its number of open connections.

We consider an architecture where the L4 load balancing policy can be modified to reduce the load imbalance between servers. In this architecture, the servers regularly feedback their number of open connections to a controller, which computes the best policy to apply in the next control period to reduce load imbalance between the servers and enforces it on the L4 load balancers.

We distinguish three main parts in this system depicted in Figure 5.3:

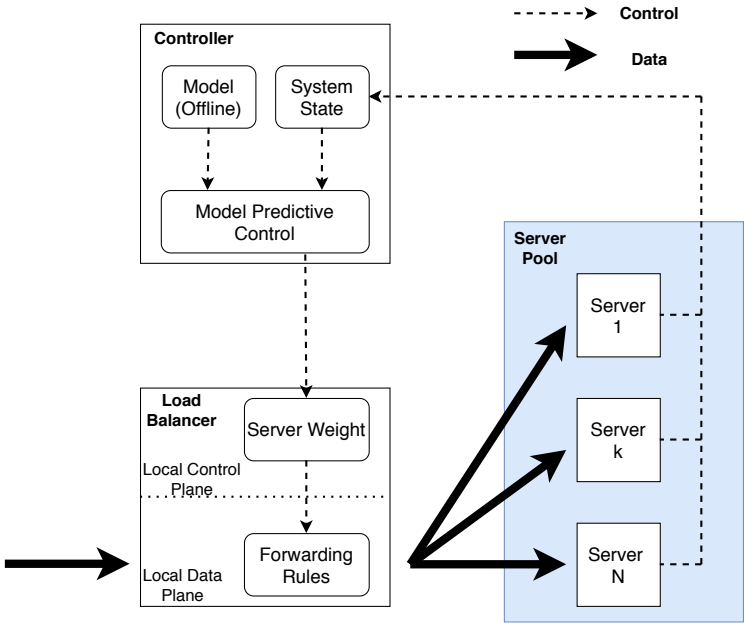


FIGURE 5.3: Load Balancing Architecture.

Server Pool

Constituted of S servers (physical or virtual machines), it processes the incoming connections and delivers service to clients. Each server may have different capacities due to different resource allocations (e.g., CPU). The servers monitor their open connections, which represent the objective to optimize. Each server's capacity weights the number of connections it has to process. For instance, if a subset of servers can process two times more connections than another subset, it should receive two times more connections. They send these data to a controller at a given pace that we call the control period (or frequency).

Load Balancer(s)

This element receives incoming connections and dispatches them on the server pool. When new connections arrive, it selects a server, stores its decision, and replaces the destination of every packet of this connection towards the associated DIP to forward them. We consider that the server selection is based on a server weight distribution D stored in the load balancer and updated by the controller. These weights represent the expected ratio of incoming connections to be forwarded to a server. A single or a pool of load balancers enforces the same distribution for every control period. This connection distribution can either be implemented with a weighted round-robin [104], a hashing space [48, 118], or a pseudo-random generator. How the server selection process is done in the dataplane is out of the scope of this work. Notice that even if the server weights are modified, the existing connections are still sent to the same server (connection consistency).

Controller

This part is in charge of computing the load balancing policy based on the current system state and then sending the computed server weight distribution (D) to the load balancer(s). This decision is devised based on two main inputs: the model, which represents the dynamics of the load, and the system state, which is the current load on the servers. Notice that the model can either be learned offline or updated online. In this work, we detail how such a controller can be built.

5.3.2 Modelization

We consider a time horizon divided into a set of consecutive control periods with duration $\Delta t \in \mathbb{R}^+$ that is denoted as $\tau = \{1, 2, ..., T\}$. We use $t = 0$ to denote the initial instant of the time horizon. We consider C the set of connections requests to the service during the time horizon τ. A connection c is characterized by three attributes: its arrival time t, its duration d, and the assigned server s. In practice, the connection duration can span from a few seconds to several minutes [91]. We call C_t the connections present at the instant t. At each instant $t \in \tau$, there is a set of new connections arriving on the service called $r_t = C_t \setminus (C_t \cap C_{t-1})$ that the load balancer has to assign to the server instances in the server pool.

Let S be the set of servers in the server pool. We consider that the server pool is static in the time horizon τ. Each server has a capacity of κ, which is

the number of connections it can process in parallel before suffering congestion. A service provider can tune this value based on a server benchmark. At each control period the server sends to the controller its current load $x_{s,t}$ formally described in Equation 5.1, where $\mathbb{1}(c,s)$ equals 1 if c is assigned to s and 0 otherwise. We call X_t the vector of the server states at time t.

$$\forall t \in \tau \quad \forall s \in S \quad x_{s,t} = \frac{\sum_{c \in C_t} \mathbb{1}(c,s)}{\kappa_s} \tag{5.1}$$

We consider that a load balancing decision at the instant $t \in \tau$ is represented by a vector u_t of normalized server weights. Each weight $w_{s,t}$ represents the probability for a new connection to be forwarded toward the server s during the control period t. The main challenge of connection load balancing is that neither the number of new connections nor their duration is known ahead of time. Thus, a load balancer can only decide based on the previous state of the system and the previous load balancing decision it has taken. The goal of our work is then to find the load balancing function Π_t that will minimize the load imbalance such that $u_t = \Pi_t(X_0, \dots X_t, u_0, \dots u_{t-1})$ for every control period t. Notice that Π_t is a constant function when the load balancing policy is static.

We now formalize a general control problem to minimize load imbalance with the Equations 5.2, 5.3, 5.4, 5.5. Equation 5.3 corresponds to a constraint that expresses the dynamics of the system over time and maps the future state of the system X_{t+1} with the previous state X_t and the load balancing decision u_t, i.e., the server weights. This state function is usually not known since it depends on the incoming traffic characteristics (r_t), i.e., the different arrival times and durations of the connections arriving at time t. Nonetheless, an accurate enough model can estimate this state function. Equation 5.3 also enforces that all the connections present during a control period (C_t) are allocated on the servers, meaning that $\sum_{s \in S} \kappa_s \times x_{s,t} = Card(C_t)$. Due to this constraint, we choose to minimize the norm of the vector X_t in our objective function (Equation 5.2). In fact, minimizing the norm of the vector $\|X_t\|_p$ (if $2 \leq p$) penalizes the distance from the mean, as well as the congestion of a server, since a higher load on a server will be more penalized. Indeed, minimizing $\|X_t\|_1$ could lead to sending the whole load on a single server (they are more likely to propose sparse solutions [12]). Notice that to have a strict minimization of the *systemic imbalance*, there should be $p = \infty$; however, it will be a more complex problem to solve. In this work, we choose $p = 2$ to simplify the formulation into a quadratic problem. Equation 5.4 shows

that the control decisions u_t can only depend on the previous system trajectory (i.e., the previous state and previous load balancing decision). Finally, Equation 5.5 gives the initial conditions of the considered system.

Objective:

$$\min \sum_{t=0}^{T} \|X_t\|_2 \tag{5.2}$$

Subject to:

$$\forall t \in [0, T-1], \quad X_{t+1} = f_t(X_t, u_t) \tag{5.3}$$

$$\forall t \in [0, T-1], \quad u_t = \Pi_t(X_0, \dots X_t, u_0, \dots u_{t-1}) \tag{5.4}$$

$$X_0 = X_{init} \tag{5.5}$$

This optimization problem aims to find the optimal set of control decisions to minimize the objective function. Two methods can be applied: 1) find the optimal set of control decision u_t supposing the dynamics of the system known (the approach we use) or 2) try to directly learn the policy function Π_t without considering the dynamics (see discussion in Section 5.7). Moreover, additional constraints could be added in the search of u_t or Π_t, such as the horizon size of the policy function to reduce the storage of numerous state or that the function is static to avoid having feedback (and the control traffic overhead). We let for future work the exploration of the whole load balancing policy design space and propose in the next section a model predictive controller approach to solving an instance of this general problem.

5.4 Model Predictive Load Balancer

In this section, we propose to solve this optimal control problem by using a Model Predictive Controller. We choose this approach since a perfect knowledge of network dynamics is hard to obtain for the entire future. Nevertheless, a reasonably accurate prediction can be obtained for a short horizon. Thus, we propose to use a Linear Time-Invariant Model to characterize the load dynamics (i.e., estimate the dynamics function f_t in Equation 5.3). We then reformulate the systemic imbalance minimization problem into a linear quadratic regulator problem. Finally, we solve this problem in a model predictive approach to minimize the model's prediction error.

We call *Service System* the system composed of the load balancer(s) and the server pool. This system has two inputs: the new connections in the system r_t and the load balancing policy u_t. The *Service System* can be represented as

TABLE 5.1: Notations.

Parameters	
$\mathcal{S}$	Set of servers in the server pool
Δt	Control period
T	Number of control periods in the time horizon
C_t	Number of active connections during a control period
α	Request decay of allocated connections during a control period
β	arrival of new incoming connections during a control period
κ_s	Capacity of the server s
Control and state variables	
$x_{s,t}$	Load of server s at instant t
X_t	State of the server pool at instant t. Vector of $x_{s,t}$ for s in $\mathcal{S}$
$w_{s,t}$	Server weight of server s at instant t.
u_t	Load balancing policy at instant t. Vector of $w_{s,t}$ for s in $\mathcal{S}$
U	Set of load balancing policy $(u_0, ..., u_T)$ in a time horizon

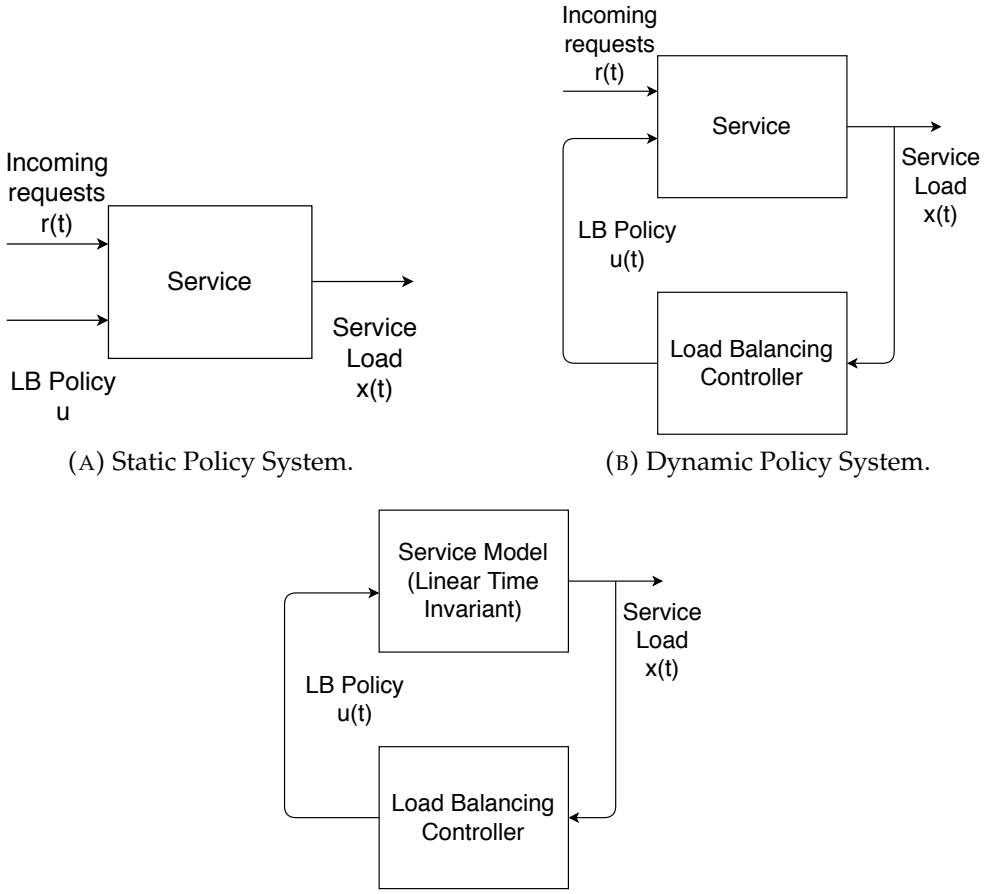
a function such that $X_{t+1} = f(r_t, u_t, X_t)$. This function f represents the open-loop behavior of the system. The static load balancing policy (Figure 5.4a) is an open-loop system: the control input u_t is independent of the output X_t and has a fixed value u. In the dynamic policy scenario (Figure 5.4b), the system becomes more complex. In this case, a controller is added, the service load X_t is fed back to the controller, which returns the load balancing decision u_t. As stated in the previous section, the connection duration and arrival rate are unknown ahead of time. We will thus embed an estimation of these parameters in the model such that $X_{t+1} = \hat{f}(u_t, X_t)$ (see Fig. 5.4c). We detail how we build such a function in the next paragraphs.

5.4.1 Linear Quadratic Regulator

We propose a Linear Quadratic Regulator problem to solve an instance of the problem described in Section 5.3.2. We first propose a linear time-invariant model to model the load dynamics and then formally define the dynamic load balancing problem based on this model.

Linear Time-Invariant Model

In this subsection, we propose a linear time-invariant model that describes the server load evolution over time. This model aims at finding a discrete linear expression linking the server load between two control periods. We will refer to this expression as *server load dynamics*.



(A) Static Policy System.

(B) Dynamic Policy System.

(C) Linear Time Invariant Model.

FIGURE 5.4: System Identification.

We distinguish two kinds of connections processed by the load balancer during a time sample t: the allocated connections and the incoming connections. The allocated connections are the ones that have arrived before sample t and are already allocated to a server (connection consistency). The incoming connections are the new ones arriving during the control period t, which will be allocated to the servers following the current load balancing decision.

During each control period t, new connections are allocated to the servers according to the server weight distribution. We call β the number of new connections arriving at the load balancer during sample t. The vector u_t is composed of the S weights and represents the load balancer policy during the period t. Besides, during a period t, some of the already allocated connections to the servers during the previous period $[0, t-1]$ expire. The parameter α represents the ratio between the number of connections being processed during sample $t-1$ and the number of those still present at period t. We call the parameter α connection decay.

From this model, we express the dynamics of a server load, that is expressing $x_{s,t+1}$ in function of $x_{s,t}$, and $w_{s,t}$ in Equation 5.6:

$$x_{s,t+1} = \alpha \times x_{s,t} + \beta \times w_{s,t} \tag{5.6}$$

We present a vectorize expression of the dynamics in Equation 5.7, where A and B are diagonal matrices of dimension $S \times S$ and can be expressed as $\alpha \times I_S$ and $\beta \times I_S$ respectively, I_S representing the identity matrix:

$$X_{t+1} = A \times X_t + B \times u_t \tag{5.7}$$

Notice that parameters α and β can be computed by analyzing the network traffic at the load balancers or at the servers (using the linux conntrack for instance [19]. We estimate α and β for each control period with the following formula: $\alpha_k = \frac{Card(C_t \cap C_{t-1})}{Card(C_{t-1})}$ and $\beta_k = Card(C_t) - Card(C_t \cap C_{t-1})$. We have then two series of α and β estimates. We can easily implement this estimation mechanism online with a system of 3 counters. Indeed during a control period, the servers can easily count the current number of connection during this period ($Card(C_t)$), the number of connection of the previous period ($Card(C_{t-1})$) and the connections arrived during the current period (β_t). By subtracting β_t to $Card(C_t)$, we get $Card(C_t \cap C_{t-1})$ and can deduce α_t.

Dynamic Load Balancing Problem

Now, we adapt the *systemic imbalance minimization problem* into a linear quadratic regulator problem.

Equation 5.8 represents the objective function we aim to minimize. This function is quadratic and is subject to linear constraints described in Equation 5.9, 5.10, and 5.11. The whole problem is a *quadratically constrained quadratic program* [12] and this formulation is more commonly known as a constrained linear quadratic regulator.

Objective:

$$min\ J(U) = \sum_{t=0}^{T} X_t^\top \times Q \times X_t + u_t^\top \times R \times u_t \tag{5.8}$$

Subject to:

$$\forall t \in [0, T-1] \quad X_{t+1} = A \times X_t + B \times u_t \tag{5.9}$$

$$\forall t \in [0, T] \quad \sum_{n=1}^{N} w_{n,k} = 1 \tag{5.10}$$

$$X_0 = X_{init} \tag{5.11}$$

The objective function in Equation 5.8 is composed of two parts representing two concurrent metrics. The first part, $X_k^\top \times Q \times X_k$, is the imbalance cost (see Section 5.3.2). It penalizes quadratically the load imbalance between servers. The second part, $u_k^\top \times R \times u_k$, is the control cost. It penalizes an "aggressive" control, i.e., the control values that vary strongly. The Q and R matrices are two diagonal matrices of dimension $S \times S$. The Q matrix represents the load deviation weights. It could be tuned if some servers need to have a more stable load than others. This paper considers that each server load has the same importance and that the Q matrix is the identity matrix. The R matrix represents the cost of control. This matrix ensures that the sequence of inputs (u_0, ... u_T) converges to a stable policy and avoids flip-flop behaviors. In practice, this matrix's weights are small compared to the Q matrix since the main objective is to reduce load imbalance. Equation 5.9 ensures that the *servers' load dynamics* are applied to link the load state with the load balancing policy. Equation 5.10 ensures that all the new connections are dispatched on the servers. It also ensures that the servers' weights are normalized. Finally, Equation 5.11 gives the initial state of the system. With the constraints expressed in Equations 5.9 and 5.11, the objective function J only depends on the load balancing policy $U = (u_0, ..., u_T)$.

5.4.2 Model Predictive Control Algorithm

The previous section described how to find the optimal policy on a given time horizon with our LTI model. This section describes how we can use the solution computed by the dynamic load balancing problem on a real system. So far, we can find the optimal load balancing policy by solving the previous problem, assuming that our LTI model prediction is accurate. In real systems, errors in measurements or approximations in the model can lead to a wrong decision. In this subsection, we propose leveraging a model predictive control approach to correct the prediction error.

In Figure 5.5, we explain how the model predictive control approach works. In this scheme, the controller gets feedback from the *Service System* at every control period. At $t = 0$, a first system state is received. This state is used

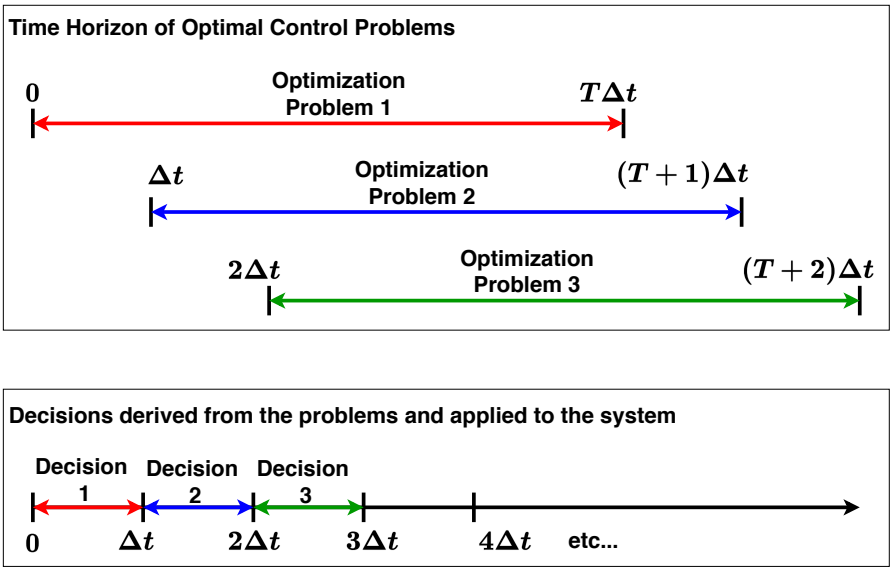


FIGURE 5.5: Receiding Horizon.

as the initial load of our dynamic load balancing problem, which is solved on a given time horizon of length T. The solution to this problem gives a set of servers' weights for the next T control periods in the time horizon. The controller takes the server weights for the next control period ($[0, \Delta t]$) and enforces it on the load balancer.At $t = \Delta t$, the controller gets again the system's current state, which can be more or less close to the previously predicted value in the first problem. This new state is used as the initial state of a new optimization problem, thus correcting the prediction. The problem is solved again, and the server weights are enforced. The same process is done at the end of each control period.

5.5 Evaluation Methodology

In this section, we explain the methodology used to evaluate our dynamic load balancer. Our goal is to evaluate our MPC solution in terms of load imbalance against various realistic network dynamics. We first describe the process used to generate synthetic traffic traces played in our simulation. Then we describe how we estimate the parameters of our LTI model from a traffic trace. Finally, we detail how a load balancing simulation is run. The experiment process is summarized in Figure 5.6. Each of these parts is implemented as a Python program (Python 3.7) and executed on a server with the following main characteristics: Intel Xeon Gold 6130 (Skylake, 2.10GHz, 4 CPUs/node, 16 cores/CPU), 768 GiB of memory.

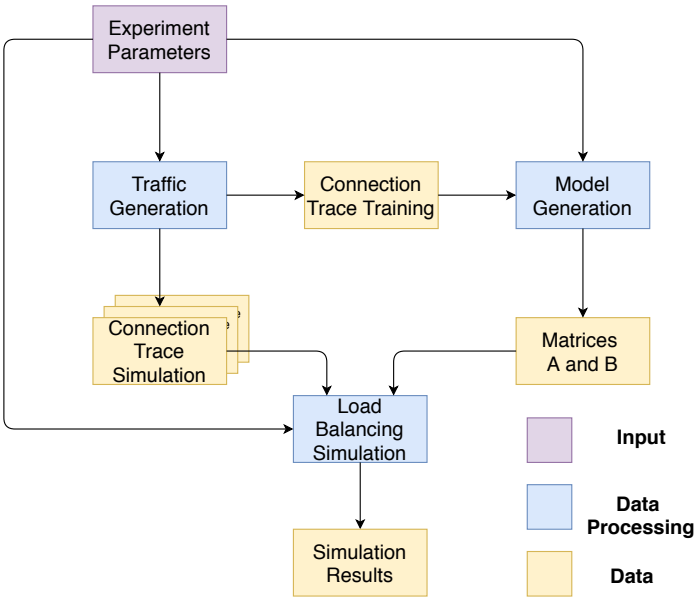


FIGURE 5.6: Simulation Workflow.

5.5.1 Traffic Traces Generation and Model Estimation

We generate traffic traces by using the Poisson Pareto Burst Process [119]. In this model, connections arrive following a Poison process with an arrival rate λ. Each of the generated connections has a duration that follows a Pareto distribution parameterized by two values: the mean duration (θ) and the Hurst parameter. We choose this model since web traffic is self-similar [20] and recent reports have shown that the connection duration follows a fat-tailed distribution [91]. We fix the Hurst parameter to 0.75. This parameter represents the long-range persistence of the traffic generation model. Since its value is comprised between 0.5 and 1, the traffic time series follows a trend. We choose a value of 0.75 so that the connections timeserie follows a long term trend but keeps a range of variability. We generate traffic traces with different dynamics patterns by varying λ and θ. The higher λ, the higher the load intensity since the server pool receives more connections during a control period. The higher θ, the higher the load stability. Following the results presented in [91], we consider that the mean web connections' duration is of the order of several seconds or tens of seconds. We generate traffic traces of 5 minutes.

We then use a traffic trace to estimate the parameters α and β of our model. We sample the traffic trace duration in control periods of size Δt. At the beginning of each control period, we enumerate the connections. Let us call this group C_t. We then compare two successive control periods, as explained in Section 5.4.1, to deduce series of α and β. We take the mean of

these series to build our model.

5.5.2 Load Balancing Simulation

Finally, we present how we run the load balancing simulation. First, we describe our MPC solver's implementation, which will be called a routine in the simulation. Then we detail the discrete event simulator we have implemented to simulate the dispatching of connections on a server pool.

Solving Dynamic Load Balancing Problem

In our MPC controller's implementation, we used *CVXPY* [23], a modeling language for convex optimization problems, to formally describe our dynamic load balancing problem. We have fixed the load deviation weights to 1 and the control weight to 0.001. We consider in these simulations that the server pool is uniform, meaning that every server has the same capacity κ. The routine's inputs are the parameters α and β as well as the current server load X_0. It solves the problem on a finite horizon with CPLEX. From a quick benchmark of our solution, we see that even for a considerable time horizon (30 control periods) and medium server pool (512 servers), our solver was able to give a solution in hundreds of milliseconds. These results could even be improved by using warm-start techniques or online optimization [105]. Still, for real systems implementation, off-the-shelf solvers might not be available due to licensing issues. In future work, we plan to find a heuristic solution to this problem.

Discrete Event Simulator

We use the parameters' estimation and a traffic trace to simulate a load balancing scenario. We fix the server pool size to 468 servers, similarly to recent works [8, 27]. We have implemented a simple discrete event simulator to do so. We consider three types of events: i) a connection arrival, ii) a connection end, iii) and a control update. When a connection arrives, the load balancer randomly picks a server in the pool based on the current servers' weights. The connection is then assigned to the chosen server. When a connection expires, it is removed from the server that was processing it. Finally, when a control event occurs, the load balancer gets the current state of the servers (number of current connections), solves the dynamic load balancing problem using CPLEX, and modifies the server weights following the MPC approach.

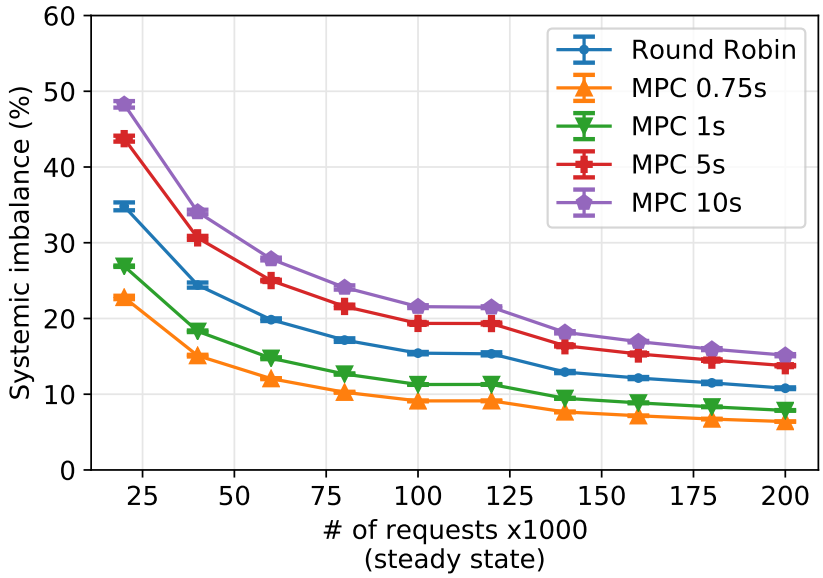


FIGURE 5.7: Impact of the arrival rate on the Systemic Imbalance.

Notice that in our implementation, we neglect the feedback loop delay (transmission time, server weight enforcement, and server weight computation), since we consider control periods significantly larger (in the order of a second) than the cumulated delay. Indeed, data transmission in a datacenter can be done in a few milliseconds, and it has been shown that servers' weight can be modified at line rate [48]. Even with our off-the-shelf solving approach, a solution can be found in a few hundreds of milliseconds.

We run simulations for varying several parameters: connection duration, connection arrival, control period, and control horizon. We run ten simulations for each combination of parameters with different traffic traces. The traces used for the model estimation have been generated with the traffic parameters (connection duration and arrival), but are never played in the simulations.

5.6 Evaluation

In this section, we evaluate the gain of using our model predictive control approach in terms of load imbalance. We exhibit how the traffic dynamics and control parameters impact the performance to find a tradeoff between load imbalance and control traffic. We compare our solution to a static and uniform load balancing baseline: the widely used round-robin policy.

We first analyze the effects of connections arrival rate on load balancing policies. We compare the round-robin policy with our MPC policy with several control periods. On Figure 5.7, the mean connection duration is fixed to 10s and the arrival rate varies from 2,000 to 20,000 connections per second. Thus, there is a mean number of connections distributed on the server spanning from 20,000 to 200,000. Notice that a front-end cluster in a Facebook datacenter can process several tens of thousands of active connections concurrently [67]. We can see that for each policy, the systemic imbalance decrease with the arrival rate. Indeed, when the arrival rate increases, each server receives a larger sample of new connections. With a larger sample, the active connections' duration distribution on the server is more likely to be similar to the duration distribution of the generated traffic. Thus, the more requests are distributed on the servers; the less imbalance is exhibited due to the big numbers law. Yet, increasing the number of connections per server to reduce imbalance is not an efficient solution since the servers need to be dimensioned to handle such a large number of connections. We can notice in Figure 5.7 that our model predictive approach outperforms the round-robin baseline for a control period of 0.75s and 1s while it exhibits worse performance for a period of 5 and 10s. Indeed, when the control period is close to the request mean duration, the controllable load becomes less stable (α tends to 0) since more and more requests last less than a control period. Thus the load prediction becomes less accurate. Still, we can see a 35% gain compared to the baseline with a control period of 750ms. Even in the worst scenario (20,000 requests in the system), one can expect a 10% reduction of the systemic imbalance.

Now we focus on the effect of the request duration on our load balancing policies. In this scenario, we vary the mean duration of the requests from 1s to 60s. We fix the mean number of requests to 60,000 and adjust the request arrival accordingly. We can see that the round-robin policy is not affected by this parameter since it dispatches the requests independently of the current server load. In this case, the systemic imbalance is 20% and slightly decreases due to the mean request number adjustment. As above, we can see that when the control period is too long compared to the mean request duration, our MPC exhibits worse performance compared to the static load balancing policy. For a control period of 10s and a mean duration of 1s, the MPC exhibits a 9% increase in load imbalance compared to the round-robin policy. Nonetheless, we observe that for a control period of 750ms and 1s the MPC performs better than the static policy from 5s request mean duration. The performance

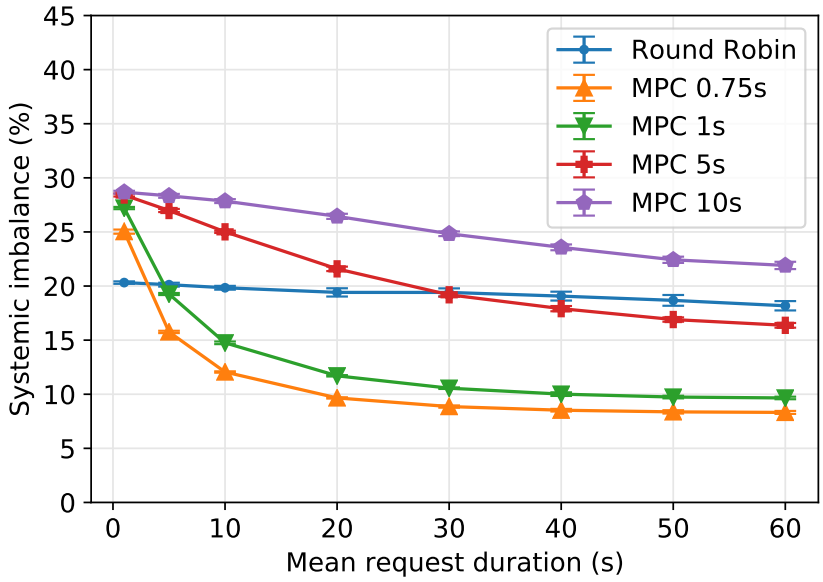
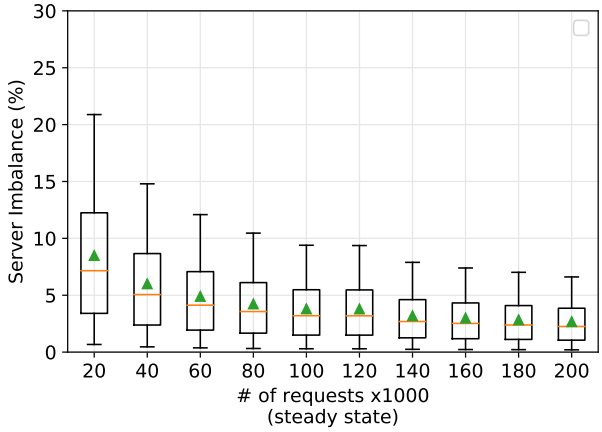


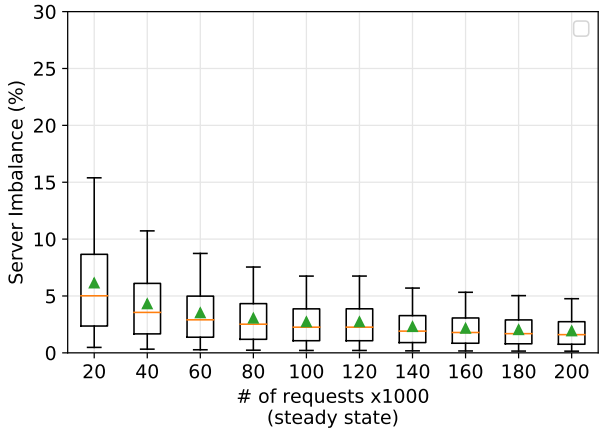
FIGURE 5.8: Impact of the connection duration on the Systemic Imbalance.

gain increases and finally reaches an asymptote around 10% of systemic imbalance. It seems that as soon as the mean duration is 10 times higher than the control period, the asymptote is reached. This might be a good rule of thumb to design the minimal control update period knowing the pattern of traffic for a performance gain. One could decide to reduce the control period. But, we see that the systemic imbalance gain slows down (the gain between 750ms and 1s is small), while the control traffic overhead still grows linearly. Knowing the traffic dynamics, one can choose a tradeoff in terms of systemic imbalance and traffic overhead.

In Figure 5.9 we compare the load imbalance distribution between the round-robin policy and the model predictive control (we fix the control period to 1s). We can see the same trend for both the MPC and the round-robin policy: the higher the number of requests, the lower the imbalance. Nonetheless, we can see that the imbalance distribution spread is lower for the model predictive approach, meaning that the load is more stable on every server. For instance, we see that 75% of the imbalance distribution is below 9% for the MPC approach with 20,000 requests in the system, while the round-robin policy is 3% higher for the same quartile. Logically we see that the extreme imbalance values are higher for the round-robin policy with 5% of the distribution higher than 21% of imbalance, while the MPC has 95% of its distribution below 15% of imbalance. Overall the mean imbalance is lower for the MPC compared to the round-robin policy for every request considered in the system.



(A) Static Policy System (round-robin).



(B) Dynamic Policy System (MPC).

FIGURE 5.9: Servers' Imbalance Distribution.

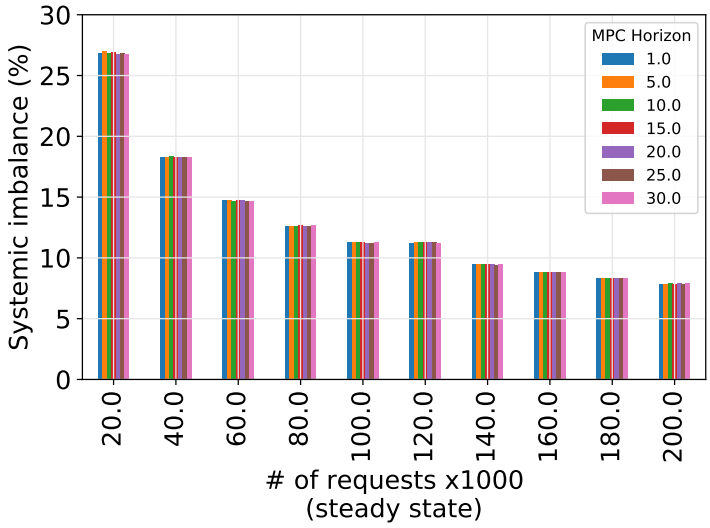


FIGURE 5.10: Impact of the horizon length.

Finally, in Figure 5.10, we look at the impact of the control horizon on the systemic imbalance. We see that our MPC approach performs the same with a control horizon spanning from 1s to 30s. It means that the short horizon solution is approximately the same in terms of systemic load imbalance compared to the long horizon. This may be due to the control cost being small compared to the imbalance cost, thus the control policy can be aggressive, and most of the imbalance is corrected in the first moves. Since an efficient strategy can be found with just a few moves, these results emphasize that a simple heuristic could solve the dynamic load balancing problem.

5.7 Discussion and Perspectives

Our results illustrate the potential gain that control theory can achieve if applied to networking problems. In this section, we discuss how our solution can adapt to operational constraints to bring these gains into the real world. We also emphasize the potential perspectives our work opens. Indeed the performance could be further improved by finding a more accurate dynamics estimator or integrating modeling errors into the control problem.

Does MPC support service scaling?

In this work, we have considered a fixed server pool size. Nonetheless, with the traffic fluctuations, servers' capacity can be modified to cope with demands. If a scaling event happens (e.g., adding more servers to the pool), the problem's parameters can be adapted to fit the new current situation for the next considered control period.

Is this solution limited to L4 load balancers?

In this paper, we have investigated the distribution of connections among servers. However, load balancing occurs in many other areas to mitigate congestion at the network or application level. The formulation of Section 5.3.2 could be adapted to fit the dynamics of the considered system, the main limitations being: What are the dynamics of the considered system? How fast can a good enough solution to the problem be found?

Is it possible to use another model?

Yes. In our work, we used a first principle approach to build a model, but other techniques could be used to predict the system's load dynamics. Nonetheless, using a linear time-invariant model simplifies the optimization problem in a quadratically constrained quadratic program that can be exactly solved in polynomial time. Machine learning could be used to estimate the system dynamics and solve the optimization problem [100]. Recht et al. discuss these solutions compared to the classical control approaches in [87] and show that they may perform poorly even on simple problem compared to classical solutions.

What if the load prediction is inaccurate?

Even if the MPC approach reduces prediction error as it takes feedback from the real system, it may still lead to short scale bursts or unstable control policy if the traffic is too bursty. Nevertheless, the optimization problem can be extended to a *finite-horizon stochastic optimal control problem* to take into account the load estimation error due to the traffic fluctuation. This problem aims to minimize the load imbalance, assuming that some states can take any value in a given range. We let the formulation of this robust MPC problem [15] for future work.

Is this solution scalable?

Yes, with some improvements. Our work shows that our optimization problem can be exactly solved in a few hundreds of milliseconds for a medium-size cluster (512 servers). Nonetheless, the problem-solving complexity is polynomial, making it unfeasible for large scale problems on short timescales if the dynamics are fast. Still, the optimal solution is not required here, and a good enough solution to the problem using a heuristic could be devised to reduce the time to found the next load balancing move. Moreover, centralizing the load balancing decision might not be an optimal architectural choice. Using a framework, such as the System Level Synthesis [25], enables the design of a distributed architecture of controllers. In this architecture, the problem is split into subproblems, which can be solved in parallel, thus reducing its complexity. This framework can be the right perspective for geo-distributed load balancing among multiple datacenters where load balancing decisions are taken in different sites in a hierarchical manner (e.g., Facebook load balancing [94]).

5.8 Conclusion

In this chapter, we have proposed an L4 load balancer control plane to reduce servers' load imbalance. We made the case that an efficient control plane for dynamic load balancing should focus on the load dynamics instead of the last state. We proposed to model a service cluster as a dynamical system whose load is controlled by its load balancing policy. This optimal control problem is, in general, hard to solve, since it needs an accurate prediction of the load dynamics. To solve this problem, we proposed a Model Predictive Controller, which finds optimal solutions on receding short horizons. We have detailed how this solution could manage load balancers and showed that a simple model could be estimated with only three counters. The evaluation of our solutions with realistic simulations shows that our approach can significantly reduce load imbalance even with a large control period (given that the control period is adapted to the dynamics). We show that the control loop should be adapted to the traffic dynamics more precisely, the control period should be shorter than the mean duration of the flow otherwise, it would lead to performance degradation compared to stateless uniform load balancing policy. In future work, we plan to find a good enough heuristic to solve the problem on a shorter timescale, thus improving the approach's efficiency and making it feasible for workload with fast dynamics. Moreover, other prediction and control solutions could be contemplated and compared to ours to further reduce the load imbalance.

Chapter 6

Conclusion and Perspectives

> Le vrai voyageur n'a pas de plan établi et n'a pas l'intention d'arriver.
>
> Lao Tseu

This thesis presented our three contributions that tackle dynamics in software networks. We covered two main topics: Service Function Chaining and Load Balancing. We conclude this thesis by providing a summary of our contributions (Sec. 6.1), and putting our thesis into perspective (Sec. 6.2).

6.1 Summary of Contributions

Today's network are mostly static as adapting them to new requirements is complex, requires human intervention and is error-prone. Network softwarization has been seen as the holy grail that would end this era and lead to network that can automatically adapt to the traffic dynamics. SDN, NFV as well as programmable dataplane let envision a network that can more finely monitor its state and adapt to a change. In this thesis, we have explored to what extent these new systems could react to the network dynamics in two domains: Service Function Chaining and Load Balancing.

6.1.1 Service Chaining

In our first contribution, we questioned some of the assumptions made by the state of the art concerning service chaining. First, we proposed to augment the network routing layer with the notion of service. Instead of relying on a centralized orchestrator, which would monitor both service functions and forwarding devices to build a map of the network, we instead envision autonomous NFV-Routers that would host functions and announce

them through their routing protocol. Our approach allows to build a distributed map of both the network topology and service functions with their related metrics. Our proposal can rely on the robustness and scalability of routing protocols and can be deployed incrementally in existing networks. Our evaluation shows that our approach due to its anycast addressing induced almost no overhead on the routing system. Second, we proposed to distribute the chaining decision. Orchestrators used to centralize the chaining decision and push static rules on forwarding elements. Instead, we argue that this decision can be taken segment by segment autonomously by NFV-Rs. This distributed chaining decision allows to share the path computing and increase the robustness of service chains. Third, our approach can support any path computation algorithm for service chain. Based on the announced metrics, service aware routing tables are built and updated autonomously by NFV-Rs. We showed two types of scenario: a multipath routing approach (WCMP) and the classical shortest path algorithm. Even if we focused on hop-by-hop routing algorithms, the augmented network view can also be leveraged to drive a source routing decision. Our large emulations have demonstrated that NFV-Routers can efficiently steer traffic through service chains at the granularity of UDP flows to more finely adapt to the network dynamics. Moreover, our solution succeeds in finely balancing the load among the service function instances.

The design of the NFV-R let envision new possibilities regarding the way the service paths are computed. In our second contribution, we tried to answer the following question: should a centralized decision be preferred compared to a hop-by-hop one? Indeed, due to its myopic view, a hop-by-hop decision may take longer path and increase the load on links or service functions. Yet a distributed decision significantly increases the control architecture robustness and reactivity. We proposed an analytical evaluation of these two approaches to assess what was the cost of a distributed control. We introduced two Integer Linear Programs to compare these two control architectures in terms of overall cost, path length and link use. We show that the distributed decision is close to the centralized one on realistic topologies, while the distributed architecture due to its local control loop can react more quickly to network events (e.g., link failure). This makes a strong case to prefer a distributed hop-by-hop chaining decisions compared to centralized ones.

6.1.2 Load Balancing

In our third contribution, we have explored to what extent network softwarization can improve load balancing in datacenters by adapting its decision to the load dynamics. So far, load balancers apply static and uniform policies to spread incoming connections. Nonetheless, this approach induces a non uniform load distribution. To cope with this issue and reduce overprovisioning, we made the case for a dynamic load balancing policy that would adapt to the current load on backend servers. We detailed how such a solution can be technically achieved due to recent innovations in programmable data planes. We then proposed a control theory based approach to reduce the server load imbalance: Model Predictive Control. In this vision, the servers' load is modeled as a dynamic system driven by the load balancing policy (i.e., traffic splitting ratio). We proposed a Linear Quadratic Regulator model to predict the load evolution and compute the optimal set of traffic splitting input to reduce imbalance. This model is then applied on a receding horizon so to directly get feedback of the current server load. We showed with our large scale simulations that MPC can significantly reduce load imbalance if the control period is adapted to the load dynamics. We have also demonstrated that this solution induces a small control overhead and could be applied as is in datacenters. We believe that this proposal is a first step to close the loop in load balancing systems so to better adapt to the traffic dynamics.

6.2 Perspectives

Along the way, several research directions have been opened in this thesis. We summarize below the possible research opportunities that we have identified, which cover Service Chaining and Dynamic Load Balancing.

Service Chaining

The NFV-R demonstrates that service chaining can be achieved by relying on distributed routing protocols. One interesting future work would be to port this approach to BGP so to build multi-domain service chains. We identified two ways to do so: rely on BGP community to stamp vSF anycast routes or extend the existing work at the IETF, which leverages on BGP Address Family Identifier to directly announce vSF without piggybacking classical IP routes. This opens interesting business perspectives as well as a number of questions on how a vSF could be chosen to deliver a service in a

multi-domain environment (e.g., vSF providers could provide pay as you go offers). One other interesting perspective would be to design metrics related on both network links and vSF state as well as composition rules to drive a routing decision. In our work, we focused on finely steering traffic through already instantiated vSFs to balance the load and deliver a service. The NFV-Router could also leverage on its distributed view to take vSF instantiation, scaling or deletion decisions to cope with traffic variation on a larger timescale (e.g., diurnal pattern). Taking this decision in a distributed manner with a limited amount of information is challenging but would also significantly improve service chains resiliency and elasticity. For instance, if a network partition occurs and that vSFs are no longer present to deliver a service, based on an updated view of the network, NFV-Rs could decide autonomously to instantiate the missing vSFs and restore the service. This opens many questions regarding how such a system could be designed: classical control theory and hysteresis mechanisms could be leveraged as well as more trendy Machine Learning approaches.

Dynamic Load Balancing

Our MPC proposal showed that dynamic load balancing policies could reduce servers' load imbalance in datacenters. Our solution is modular and could be improved. Innovation could be made in the model design as well as the objective function to more finely forecast the load evolution and take more insightful decisions. To handle the stochastic variation of the load one could reformulate the problem into a finite-horizon stochastic optimal control problem to increase the system stability. Moreover, interesting works in the control community propose System Level Synthesis approach to split a large problem into a hierarchy of sub problems. This approach could be extremely interesting for geo-distributed services, the control plane could be divided in a hierarchical manner and distributed among the different sites to better optimize the resource use at a global scale. In addition to these approaches, one could use Machine Learning methods to i) forecast the load evolution and ii) take traffic splitting decision to reduce load imbalance. A reinforcement learning approach could also avoid model estimation and directly try to solve the optimization problem by getting direct feedback of the infrastructure, when it applies a decision. Yet, all these new methods are not well understood. Their main issue is that they replace well defined hypothesis that can be verified or tested by data biases that are hard to detect. Even

if they can perform well in the lab, due to their black box nature it is difficult to guarantee their potential performance on real problems.

Publications

International journals with peer review

- A. Wion, M. Bouet, L. Iannone and V. Conan, "Change in Continuity: Chaining Services with an Augmented IGP", *IEEE Transactions on Network and Service Management, Volume 16, Number 4, Pages 1332-1344, 2019*

International conferences with peer review

- A. Wion, M. Bouet, L. Iannone and V. Conan "Distributed Function Chaining with Anycast Routing", *Proc. of the 2019 ACM Symposium on SDN Research (SOSR)*

- A. Wion, M. Bouet, L. Iannone and V. Conan "Let there be chaining: How to augment your IGP to chainyour services", *Proc. of the IFIP Networking conference, 2019*

Posters in international conferences with peer review:

- A. Wion, M. Bouet, L. Iannone and V. Conan "Finding Next Service Hop with NFV-Routers", *38th Annual IEEE International Conference on Computer Communications (INFOCOM), 2019*, **Best Poster Paper Runner-up**

Under review

- A. Wion, M. Bouet, L. Iannone and V. Conan, "Using Model Predictive Control to balance Service Load in Data Centers", *IEEE Transactions on Network and Service Management*

Patent

- A. Wion, M. Bouet, L. Iannone and V. Conan, "Système de Gestion distribuée pour un réseau de communication comportant une pluralité de fonctions réseau virtualisées", *Reference: EP3474492*.

Bibliography

[1] A. Abdelsalam, F. Clad, C. Filsfils, et al. "Implementation of virtual network function chaining through segment routing in a linux-based NFV infrastructure". In: *Proc. of the IEEE Conference on Network Softwarization (NetSoft)*. 2017, pp. 1–5.

[2] Ibrahim Afolabi et al. "Network slicing and softwarization: A survey on principles, enabling technologies, and solutions". In: *IEEE Communications Surveys & Tutorials* 20.3 (2018), pp. 2429–2453.

[3] Mohammad Alizadeh et al. "CONGA: Distributed congestion-aware load balancing for datacenters". In: *Proceedings of the 2014 ACM conference on SIGCOMM*. 2014, pp. 503–514.

[4] Saeed Akhoondian Amiri et al. "Charting the Algorithmic Complexity of Waypoint Routing". In: *ACM SIGCOMM Computer Communication Review* 48.1 (2018), pp. 42–48.

[5] Joao Taveira Araujo et al. "Balancing on the edge: Transport affinity without network state". In: *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. 2018, pp. 111–124.

[6] Alia Atlas et al. *An Architecture for the Interface to the Routing System*. RFC 7921. June 2016. DOI: 10.17487/RFC7921. URL: https://rfc-editor.org/rfc/rfc7921.txt.

[7] Daniel Balouek, Carpen Amarie, et al. "Adding Virtualization Capabilities to the Grid'5000 Testbed". In: *Cloud Computing and Services Science*. Vol. 367. Communications in Computer and Information Science. Springer International Publishing, 2013, pp. 3–20.

[8] Tom Barbette et al. "A High-Speed Load-Balancer Design with Guaranteed Per-Connection-Consistency". In: *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. Santa Clara, CA: USENIX Association, Feb. 2020, pp. 667–683. ISBN: 978-1-939133-13-7. URL: https://www.usenix.org/conference/nsdi20/presentation/barbette.

[9] Theophilus Benson, Aditya Akella, and David A Maltz. "Network traffic characteristics of data centers in the wild". In: *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*. 2010, pp. 267–280.

[10] Pat Bosshart, Dan Daly, Glen Gibb, et al. "P4: Programming protocol-independent packet processors". In: *ACM SIGCOMM Computer Communication Review* 44.3 (2014), pp. 87–95.

[11] Pat Bosshart et al. "Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN". In: *ACM SIGCOMM Computer Communication Review* 43.4 (2013), pp. 99–110.

[12] Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[13] Sebastian Brandt, Klaus-Tycho Förster, and Roger Wattenhofer. "On consistent migration of flows in SDNs". In: *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*. IEEE. 2016, pp. 1–9.

[14] Scott W. Brim and Brian E. Carpenter. *Middleboxes: Taxonomy and Issues*. RFC 3234. Feb. 2002. DOI: 10.17487/RFC3234. URL: https://rfc-editor.org/rfc/rfc3234.txt.

[15] Eduardo F Camacho and Carlos Bordons Alba. *Model predictive control*. Springer Science & Business Media, 2013.

[16] Lianjie Cao et al. "Nfv-vital: A framework for characterizing the performance of virtual network functions". In: *Proc. of the IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*. 2015, pp. 93–99.

[17] Subhrendu Chattopadhyay et al. "Amalgam: Distributed Network Control With Scalable Service Chaining". In: *2020 IFIP Networking Conference (Networking)*. IEEE. 2020, pp. 519–523.

[18] Xiaoqi Chen et al. "Fine-grained queue measurement in the data plane". In: *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*. 2019, pp. 15–29.

[19] *Conntrack website*. https://conntrack-tools.netfilter.org/manual.html. Accessed: 2020-29-09.

[20] Mark E Crovella and Azer Bestavros. "Self-similarity in World Wide Web traffic: evidence and possible causes". In: *IEEE/ACM Transactions on networking* 5.6 (1997), pp. 835–846.

[21] *Data Plane Developement Kit*. https://www.dpdk.org/. Accessed: 2019-07-12.

[22] *DEFO website*. https://sites.uclouvain.be/defo/. Accessed: 2019-07-12.

[23] Steven Diamond and Stephen Boyd. "CVXPY: A Python-embedded modeling language for convex optimization". In: *The Journal of Machine Learning Research* 17.1 (2016), pp. 2909–2913.

[24] *DISTributed systems EMulator (DISTEM)*. https://distem.gforge.inria.fr/. Accessed: 2019-07-12.

[25] John C Doyle et al. "System level synthesis: A tutorial". In: *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*. IEEE. 2017, pp. 2856–2867.

[26] Sevil Dräxler and Holger Karl. "SPRING: scaling, placement, and routing of heterogeneous services with flexible structures". In: *2019 IEEE Conference on Network Softwarization (NetSoft)*. IEEE. 2019, pp. 115–123.

[27] Daniel E Eisenbud et al. "Maglev: A fast and reliable software network load balancer". In: *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*. 2016, pp. 523–535.

[28] Adrian Farrel et al. *BGP Control Plane for the Network Service Header in Service Function Chaining*. Internet-Draft draft-ietf-bess-nsh-bgp-control-plane-18. Work in Progress. Internet Engineering Task Force, Aug. 2020. 71 pp. URL: https://datatracker.ietf.org/doc/html/draft-ietf-bess-nsh-bgp-control-plane-18.

[29] *Fast data – Input/Output (fd.io)*. https://fd.io/. Accessed: 2019-07-12.

[30] Seyed Kaveh Fayazbakhsh et al. "Flowtags: Enforcing network-wide policies in the presence of dynamic middlebox actions". In: *Proc. of the ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*. 2013, pp. 19–24.

[31] Klaus-Tycho Foerster, Stefan Schmid, and Stefano Vissicchio. "Survey of consistent software-defined network updates". In: *IEEE Communications Surveys & Tutorials* 21.2 (2018), pp. 1435–1461.

[32] P. Francois, M. Shand, and O. Bonaventure. "Disruption Free Topology Reconfiguration in OSPF Networks". In: *Proc. of the IEEE International Conference on Computer Communications (INFOCOM)*. 2007, pp. 89–97.

[33] Pierre Francois et al. "Achieving sub-second IGP convergence in large IP networks". In: *ACM SIGCOMM Computer Communication Review* 35.3 (2005), pp. 35–44.

[34] Ruomei Gao, Constantinos Dovrolis, and Ellen W Zegura. "Avoiding Oscillations Due to Intelligent Route Control Systems." In: *INFOCOM*. 2006.

[35] Aaron Gember et al. "Stratos: A Network-Aware Orchestration Layer for Middleboxes in the Cloud". In: *CoRR* abs/1305.0209 (2013). arXiv: 1305.0209. URL: http://arxiv.org/abs/1305.0209.

[36] Aaron Gember-Jacobson et al. "OpenNF: Enabling innovation in network function control". In: *ACM SIGCOMM Computer Communication Review*. Vol. 44. 4. 2014, pp. 163–174.

[37] Milad Ghaznavi et al. "Distributed service function chaining". In: *IEEE Journal on Selected Areas in Communications* 35.11 (2017), pp. 2479–2489.

[38] Vijay Gill et al. *SWAN: achieving high utilization in networks*. US Patent 8,977,756. 2015.

[39] Jochen W Guck et al. "Unicast QoS routing algorithms for SDN: A comprehensive survey and performance evaluation". In: *IEEE Communications Surveys & Tutorials* 20.1 (2017), pp. 388–415.

[40] Arpit Gupta et al. "Sonata: Query-driven streaming network telemetry". In: *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. 2018, pp. 357–371.

[41] Joel M. Halpern and Carlos Pignataro. *Service Function Chaining (SFC) Architecture*. RFC 7665. 2015. DOI: 10.17487/RFC7665. URL: https://rfc-editor.org/rfc/rfc7665.txt.

[42] Joel M. Halpern et al. *Forwarding and Control Element Separation (ForCES) Protocol Specification*. RFC 5810. Mar. 2010. DOI: 10.17487/RFC5810. URL: https://rfc-editor.org/rfc/rfc5810.txt.

[43] Renaud Hartert et al. "A declarative and expressive approach to control forwarding paths in carrier-grade networks". In: *ACM SIGCOMM computer communication review*. Vol. 45. 4. ACM. 2015, pp. 15–28.

[44] Davit Harutyunyan et al. "Latency-aware service function chain placement in 5G mobile networks". In: *2019 IEEE Conference on Network Softwarization (NetSoft)*. IEEE. 2019, pp. 133–141.

[45] Juliver Gil Herrera and Juan Felipe Botero. "Resource allocation in NFV: A comprehensive survey". In: *IEEE Transactions on Network and Service Management* 13.3 (2016), pp. 518–532.

[46] Chi-Yao Hong et al. "B4 and after: managing hierarchy, partitioning, and asymmetry for availability and scale in google's software-defined WAN". In: *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. 2018, pp. 74–87.

[47] Christian Hopps. *Analysis of an Equal-Cost Multi-Path Algorithm*. RFC 2992. Nov. 2000. DOI: 10.17487/RFC2992. URL: https://rfc-editor.org/rfc/rfc2992.txt.

[48] Kuo-Feng Hsu et al. "Adaptive Weighted Traffic Splitting in Programmable Data Planes". In: *Proceedings of the Symposium on SDN Research*. 2020, pp. 103–109.

[49] Kuo-Feng Hsu et al. "Contra: A programmable system for performance-aware routing". In: *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. 2020, pp. 701–721.

[50] Nanyang Huang et al. "Software-Defined Label Switching: Scalable Per-Flow Control in SDN". In: *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*. IEEE. 2018, pp. 1–10.

[51] Qun Huang, Patrick PC Lee, and Yungang Bao. "Sketchlearn: Relieving user burdens in approximate measurement with automated statistical inference". In: *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. 2018, pp. 576–590.

[52] Nicolas Huin, Brigitte Jaumard, and Frédéric Giroire. "Optimal network service chain provisioning". In: *IEEE/ACM Transactions on Networking* 26.3 (2018), pp. 1320–1333.

[53] Nicolas Huin et al. "Energy-efficient service function chain provisioning". In: *Journal of Optical Communications and Networking* 10.3 (2018), pp. 114–124.

[54] J. M. Jaffe. "Algorithms for finding paths with multiple constraints". In: *Networks* 14.1 (1984), pp. 95–116.

[55] Sushant Jain et al. "B4: Experience with a globally-deployed software defined WAN". In: *ACM SIGCOMM Computer Communication Review* 43.4 (2013), pp. 3–14.

[56] Xin Jin et al. "Dynamic scheduling of network updates". In: *ACM SIGCOMM Computer Communication Review* 44.4 (2014), pp. 539–550.

[57] Murad Kablan et al. "Stateless Network Functions: Breaking the Tight Coupling of State and Processing". In: *Proc. of the USENIX Conference on Networked Systems Design and Implementation (NSDI)*. 2017, pp. 97–112. ISBN: 978-1-931971-37-9.

[58] Naga Katta et al. "Hula: Scalable load balancing using programmable data planes". In: *Proceedings of the Symposium on SDN Research*. 2016, pp. 1–12.

[59] Wolfgang Kellerer et al. "Adaptable and data-driven softwarized networks: Review, opportunities, and challenges". In: *Proceedings of the IEEE* 107.4 (2019), pp. 711–731.

[60] Junaid Khalid and Aditya Akella. "Correctness and Performance for Stateful Chained Network Functions". In: *Proc. of the 16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. USENIX Association, 2019, pp. 501–516. ISBN: 978-1-931971-49-2. URL: https://www.usenix.org/conference/nsdi19/presentation/khalid.

[61] V. F. Kolchin, B. A. Sevastianov, and V. P. Chistiakov. *Random allocations / Valentin F. Kolchin, Boris A. Sevastyanov, Vladimir P. Chistyakov ; translation ed., A. V. Balakrishnan*. English. V. H. Winston ; distributed solely by Halsted Press Washington : New York, 1978, xi, 262 p. ; ISBN: 0470993944.

[62] Sameer G Kulkarni et al. "Nfvnice: Dynamic backpressure and scheduling for nfv service chains". In: *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. 2017, pp. 71–84.

[63] Bob Lantz, Brandon Heller, and Nick McKeown. "A network in a laptop: rapid prototyping for software-defined networks". In: *Proc. of the ACM SIGCOMM Workshop on Hot Topics in Networks*. 2010, p. 19.

[64] Tony Li, Ravi Chandra, and Paul S. Traina. *BGP Communities Attribute*. RFC 1997. 1996. DOI: 10.17487/RFC1997. URL: https://rfc-editor.org/rfc/rfc1997.txt.

[65] Nick McKeown et al. "OpenFlow: enabling innovation in campus networks". In: *ACM SIGCOMM Computer Communication Review* 38.2 (2008), pp. 69–74.

[66] David Meyer and Randy Bush. *Some Internet Architectural Guidelines and Philosophy*. RFC 3439. Dec. 2002. DOI: 10.17487/RFC3439. URL: https://rfc-editor.org/rfc/rfc3439.txt.

[67] Rui Miao et al. "Silkroad: Making stateful layer-4 load balancing fast and cheap using switching asics". In: *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. 2017, pp. 15–28.

[68] Nithin Michael and Ao Tang. "Halo: Hop-by-hop adaptive link-state optimal routing". In: *IEEE/ACM Transactions on Networking* 23.6 (2014), pp. 1862–1875.

[69] Michael Mitzenmacher. "The power of two choices in randomized load balancing". In: *IEEE Transactions on Parallel and Distributed Systems* 12.10 (2001), pp. 1094–1104.

[70] Ghada Moualla, Thierry Turletti, and Damien Saucez. "An availability-aware SFC placement algorithm for fat-tree data centers". In: *2018 IEEE 7th International Conference on Cloud Networking (CloudNet)*. IEEE. 2018, pp. 1–4.

[71] Priyanka Naik, Dilip Kumar Shaw, and Mythili Vutukuru. "NFVPerf: Online performance monitoring and bottleneck detection for NFV". In: *Proc. of the IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. 2016, pp. 154–160.

[72] Thanh Dang Nguyen, Marco Chiesa, and Marco Canini. "Decentralized consistent updates in SDN". In: *Proceedings of the Symposium on SDN Research*. 2017, pp. 21–33.

[73] M. Obadia et al. "Revisiting NFV orchestration with routing games". In: *Proc. of the IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. 2016, pp. 107–113.

[74] Vladimir Olteanu et al. "Stateless Datacenter Load-balancing with Beamer". In: *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. Renton, WA: USENIX Association, Apr. 2018, pp. 125–139. ISBN: 978-1-939133-01-4. URL: https://www.usenix.org/conference/nsdi18/presentation/olteanu.

[75] *ONOS SDN Controller*. https://onosproject.org/. Accessed: 2019-07-12.

[76] *Open Platform for NFV (OPNFV)*. https://opnfv.org. Accessed: 2019-07-12.

[77] *Opendaylight SDN Controller*. https://www.opendaylight.org/. Accessed: 2019-07-12.

[78] *P4 software switch*. https://github.com/p4lang/behavioral-model. Accessed: 2019-07-12.

[79] Shoumik Palkar, Chang Lan, Sangjin Han, et al. "E2: A Framework for NFV Applications". In: *Proc. of the Symposium on Operating Systems Principles (SOSP)*. 2015, pp. 121–136. ISBN: 978-1-4503-3834-9.

[80] Aurojit Panda et al. "Cap for networks". In: *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*. 2013, pp. 91–96.

[81] Parveen Patel et al. "Ananta: Cloud scale load balancing". In: *ACM SIGCOMM Computer Communication Review* 43.4 (2013), pp. 207–218.

[82] Rahul Potharaju and Navendu Jain. "Demystifying the dark side of the middle: a field study of middlebox failures in datacenters". In: *Proceedings of the 2013 conference on Internet measurement conference*. 2013, pp. 9–22.

[83] Zafar Ayyub Qazi, Cheng-Chun Tu, Luis Chiang, et al. "SIMPLE-fying Middlebox Policy Enforcement Using SDN". In: *Proc. of the ACM SIGCOMM*. 2013, pp. 27–38. ISBN: 978-1-4503-2056-6.

[84] P. Quinn, U. Elzur, and C. Pignataro. *Network Service Header (NSH)*. RFC 8300. IETF, 2018.

[85] Paul Quinn and Thomas Nadeau. *Problem Statement for Service Function Chaining*. RFC 7498. IETF, 2015.

[86] Shriram Rajagopalan et al. "Split/Merge: System Support for Elastic Execution in Virtual Middleboxes". In: *Proc. of the USENIX Conference on Networked Systems Design and Implementation (NSDI)*. 2013, pp. 227–240.

[87] Benjamin Recht. "A tour of reinforcement learning: The view from continuous control". In: *Annual Review of Control, Robotics, and Autonomous Systems* 2 (2019), pp. 253–279.

[88] Yakov Rekhter, Susan Hares, and Tony Li. *A Border Gateway Protocol 4 (BGP-4)*. RFC 4271. 2006. DOI: 10.17487/RFC4271. URL: https://rfc-editor.org/rfc/rfc4271.txt.

[89] Jerome H Saltzer, David P Reed, and David D Clark. "End-to-end arguments in system design". In: *ACM Transactions on Computer Systems (TOCS)* 2.4 (1984), pp. 277–288.

[90] *Scapy, packet manipulation python library*. https://github.com/secdev/scapy. Accessed: 2019-07-12.

[91] Brandon Schlinker et al. "Internet Performance from Facebook's Edge". In: *Proceedings of the Internet Measurement Conference*. IMC '19. Amsterdam, Netherlands: Association for Computing Machinery, 2019, 179–194. ISBN: 9781450369480. URL: https://doi.org/10.1145/3355369.3355567.

[92] Anees Shaikh, Jennifer Rexford, and Kang G. Shin. "Load-sensitive Routing of Long-lived IP Flows". In: *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*. ACM, 1999, pp. 215–226.

[93] Justine Sherry et al. "Making Middleboxes Someone else's Problem: Network Processing As a Cloud Service". In: *Proc. of the ACM SIGCOMM Conference*. 2012, pp. 13–24. ISBN: 978-1-4503-1419-0.

[94] Patrick Shuff. "Building A Billion User Load Balancer". In: Dublin: USENIX Association, May 2015.

[95] Neil Spring, Ratul Mahajan, and David Wetherall. "Measuring ISP topologies with Rocketfuel". In: *ACM SIGCOMM Computer Communication Review* 32.4 (2002), pp. 133–145.

[96] T. Taleb et al. "On Multi-Access Edge Computing: A Survey of the Emerging 5G Network Edge Cloud Architecture and Orchestration". In: *IEEE Communications Surveys Tutorials* 19.3 (2017), pp. 1657–1681.

[97] Shu Tao et al. "Exploring the performance benefits of end-to-end path switching". In: *Proceedings of the 12th IEEE International Conference on Network Protocols, 2004. ICNP 2004.* IEEE. 2004, pp. 304–315.

[98] Nicolas Tastevin, Mathis Obadia, and Mathieu Bouet. "A graph approach to placement of Service Functions Chains". In: *Proc. of the IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. 2017, pp. 134–141.

[99] *The FRRouting Protocol Suite*. https://frrouting.org/. Accessed: 2019-07-12.

[100] Mathukumalli Vidyasagar and Rajeeva L Karandikar. "A learning theory approach to system identification and stochastic adaptive control". In: *Probabilistic and randomized methods for design under uncertainty*. Springer, 2006, pp. 265–302.

[101] *View on 5G Architecture*. Tech. rep. 5G PPP Architecture Working Group, June 2019.

[102] Stefano Vissicchio, Laurent Vanbever, and Olivier Bonaventure. "Opportunities and research challenges of hybrid software defined networks". In: *ACM SIGCOMM Computer Communication Review* 44.2 (2014), pp. 70–75.

[103] Stefano Vissicchio et al. "Central control over distributed routing". In: *ACM SIGCOMM Computer Communication Review*. Vol. 45. 4. ACM. 2015, pp. 43–56.

[104] Weikun Wang and Giuliano Casale. "Evaluating weighted round robin load balancing for cloud web services". In: *2014 16th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*. IEEE. 2014, pp. 393–400.

[105] Yang Wang and Stephen Boyd. "Fast model predictive control using online optimization". In: *IEEE Transactions on control systems technology* 18.2 (2009), pp. 267–278.

[106] Zheng Wang and Jon Crowcroft. "Bandwidth-delay based routing algorithms". In: *Proc. of the IEEE Global Telecommunications Conference (GLOBECOM)*. Vol. 3. 1995, pp. 2129–2133.

[107] A. Wion et al. "Change in Continuity: Chaining Services With an Augmented IGP". In: *IEEE Transactions on Network and Service Management* 16.4 (2019), pp. 1332–1344.

[108] A. Wion et al. "Distributed Function Chaining with Anycast Routing". In: *Proc. of the Symposium on SDN Research*. ACM. 2019.

[109] A. Wion et al. "Let there be Chaining: How to Augment your IGP to Chain your Services". In: *Proc. of the IFIP Networking conference* (2019).

[110] Thomas Wirtgen et al. "The Case for Pluginized Routing Protocols". In: *2019 IEEE 27th International Conference on Network Protocols (ICNP)*. IEEE. 2019, pp. 1–12.

[111] Thomas Wirtgen et al. "xBGP: When You Can't Wait for the IETF and Vendors". In: *Proceedings of the 19th ACM Workshop on Hot Topics in Networks*. 2020, pp. 1–7.

[112] Shinae Woo et al. "Elastic Scaling of Stateful Network Functions". In: *Proc. of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. 2018.

[113] Li Yizhou et al. *Architecture of Dynamic-Anycast in Compute First Networking (CFN- Dyncast)*. Internet-Draft draft-li-rtgwg-cfn-dyncast-architecture-00. Work in Progress. Internet Engineering Task Force, Oct. 2020. 15 pp. URL: https://datatracker.ietf.org/doc/html/draft-li-rtgwg-cfn-dyncast-architecture-00.

[114] Pamela Zave et al. "Dynamic service chaining with dysco". In: *Proc. of the Conference of the ACM Special Interest Group on Data Communication*. 2017, pp. 57–70.

[115] Cheng Zhang et al. "HyperV: A high performance hypervisor for virtualization of the programmable data plane". In: *2017 26th International Conference on Computer Communication and Networks (ICCCN)*. IEEE. 2017, pp. 1–9.

[116] Ying Zhang, Neda Beheshti, Ludovic Beliveau, et al. "Steering: A software-defined networking for inline service chaining". In: *Proc. of IEEE Network Protocols (ICNP)*. 2013, pp. 1–10.

[117] Peng Zheng, Theophilus Benson, and Chengchen Hu. "P4visor: Lightweight virtualization and composition primitives for building and testing modular programs". In: *Proceedings of the 14th International Conference on emerging Networking EXperiments and Technologies*. 2018, pp. 98–111.

[118] Junlan Zhou et al. "WCMP: Weighted cost multipathing for improved fairness in data centers". In: *Proc. of the European Conference on Computer Systems*. 2014, p. 5.

[119] Moshe Zukerman, Timothy D Neame, and Ronald G Addie. "Internet traffic modeling and future technology implications". In: *IEEE INFOCOM 2003. Twenty-second Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE Cat. No. 03CH37428)*. Vol. 1. IEEE. 2003, pp. 587–596.

ECOLE DOCTORALE

Titre : Plan de contrôle dans les réseaux logiciels dynamiques

Mots clés : Réseaux IP, Chaînage de service, Équilibrage de Charge, Théorie du contrôle, Routage

Résumé : Au cours de ces dernières années, les réseaux se sont transformés passant d'une infrastructure à base de matériel Au cours de ces dernières années, les réseaux se sont transformés passant d'une infrastructure à base de matériel dédié implémentant des fonctions statiques à des solutions logicielles plus flexibles. D'un côté, le SDN (Software Defined Networks) permet de contrôler les opérations de transmission, alors que de l'autre le NFV (Network Function Virtualization) crée des fonctions élastiques qui peuvent s'adapter à la demande. Jusqu'à présent, ces solutions ont été utilisées pour simplifier la gestion et l'exploitation des réseaux mais elles laissent également envisager un réseau qui peut automatiquement réagir à des événements réseaux. Dans cette thèse, nous explorons dans quelle mesure ces nouveaux réseaux logiciels peuvent être utilisés pour s'adapter à la dynamique inhérentes aux réseaux. Notre première contribution s'intéresse au chaînage de service, c'est à dire la capacité de diriger des flux de données à travers un ensemble de points intermédiaires, qui hébergent des fonctions, avant d'atteindre leur destination. Nous montrons qu'un plan de control distribué, qui s'appuie sur les protocoles de routage existants et qui est constitué par des noeuds autonomes, peut dynamiquement diriger le trafic à travers des chaines de services. Notre solution adapte sa décision au trafic sur le réseau et équilibre automatiquement la charge induite sur les fonctions présentes sur le réseau. De plus, notre proposition, au contraire des solutions existantes, peut être déployée progressivement dans les réseaux actuels. Dans notre seconde contribution, nous comparons deux types de décision de chaînage : une approche centralisée avec une vue de bout en bout de la chaîne et une approche distribuée qui dirige uniquement les flux d'une fonction à l'autre. Nous montrons que ces deux décisions sont proches dans des topologies réalistes. Ainsi, un chaînage saut par saut pourrait être utilisé sans affecter les performances du réseau. Finalement, nous explorons comment les réseaux logiciels peuvent réagir à la dynamique des réseaux dans les centres de données. Jusqu'à présent, des équilibreurs de charges utilisaient des politiques statiques afin de répartir le trafic sur les serveurs, ce qui amenait du déséquilibre et gâchait des ressources. Nous proposons d'asservir le système et d'adapter dynamiquement la politique à la variation de charge des serveurs. Notre approche MPC (Model Predictive Control) est efficace afin de réduire le déséquilibre de charge à une basse fréquence de contrôle améliorant ainsi le nombre de requêtes qu'un ensemble de serveur peut traiter.

Title : Control Plane in Dynamic Software Network

Keywords : IP Networks, Service Function Chaining, Load Balancing, Control Theory, Routing

Abstract : During the last years, network infrastructure has moved from dedicated-hardware solutions implementing fixed functions to more flexible software based ones. On one hand, SDN (Software Defined Network) can flexibly control forwarding operations, while on the other, NFV (Network Function Virtualization) creates elastic functions that can scale with the user demands. So far, these solutions have been used to simplify network management and operations, but they let envision a network that can automatically react to network events. In this thesis, we explore to what extent these new software networks can be used to react and adapt finely to the network dynamics. Our first contribution focuses on service chaining : the ability to steer flows through a set of waypoints hosting functions before they reach their destinations. We show that a distributed control plane that relies on existing routing protocols and is constituted by autonomous nodes can dynamically steer traffic through chains of services. Our solution finely adapts its decision to the network traffic and automatically balances the induced load on the functions present in the network. Moreover, our proposal, contrary to existing solutions, can be incrementally deployed in today's network. In our second contribution, we compare two types of chaining decisions : a centralized one with an end-to-end view of the chain and a distributed approach that solely routes flow from a function to another. We show that the two decisions are close in realistic topologies. Thus, hop-by-hop chaining could be used without affecting chaining performance. Finally, we explore how software networks can react to network dynamics in datacenters. So far, load balancers use static policies to spread incoming traffic on servers, which leads to imbalance and overprovisioning. We propose to close the loop and dynamically adapt the policy to the server load variation. Our MPC (Model Predictive Control) approach proved to be efficient to reduce load imbalance at a slow pace, thus improving the number of requests a cluster can process.

Institut Polytechnique de Paris
91120 Palaiseau, France