Une nouvelle architecture d'automatisation des réseaux : de la détection d'anomalie à la reconfiguration dynamique

Alessio Diamanti

HESAM UNIVERSITÉ

le cnam

École doctorale Sciences des Métiers de l'Ingénieur

Centre d'études et de recherche en informatique et communications

THÈSE

présentée par : **Alessio DIAMANTI**
soutenue le : **21 Décembre 2021**

pour obtenir le grade de : **Docteur d'HESAM Université**
préparée au : **Conservatoire National des Arts et Métiers**
Discipline : **Informatique**
Spécialité : **Informatique**

Une nouvelle architecture d'automatisation des réseaux : de la détection d'anomalie à la reconfiguration dynamique

Jury

M. Philippe OWEZARSKI	Professeur, CNRS	Rapporteur
M. Adrien LEBRE	Professeur, IMT Atlantique	Rapporteur
Mme Thi-Mai-Trang NGUYEN	Maître de conférences, Sorbonne Université	Présidente
M. Keun-Woo LIM	Maître de conférences, Telecom Paristech	Examinateur
M. Roberto RIGGIO	Professeur associé, Università politecnica delle Marche	Examinateur
M. Nikaein NAVID	Professeur, Eurecom	Examinateur
M. Stefano SECCI	Professeur, CNAM	Directeur de thèse
M. José Manuel SÁNCHEZ VÍLCHEZ	Ingénieur de recherche, Orange	Co-encadrant
M. Laurent CIAVAGLIA	Ingénieur de recherche, Rakuten	Invité
M. Bertrand DECOCQ	Team and Program Manager, Orange	Invité

THÈSE

Affidavit

Je soussigné , Alessio Diamanti, déclare par la présente que le travail présenté dans ce manuscrit est mon propre travail, réalisé sous la direction scientifique de Stefano Secci et de José Manuel Sanchez Vilchez, dans le respect des principes d'honnêteté, d'intégrité et de responsabilité inhérents à la mission de recherche. Les travaux de recherche et la rédaction de ce manuscrit ont été réalisés dans le respect de la charte nationale de déontologie des métiers de la recherche. Ce travail n'a pas été précédemment soumis en France ou à l'étranger dans une version identique ou similaire à un organisme examinateur.

Fait à Paris, le 27/10/2021

Signature 

Affidavit

I, undersigned, Alessio Diamanti, hereby declare that the work presented in this manuscript is my own work, carried out under the scientific direction of Stefano Secci and of José Manuel Sanchez Vilchez, in accordance with the principles of honesty, integrity and responsibility inherent to the research mission. The research work and the writing of this manuscript have been carried out in compliance with the French charter for Research Integrity. This work has not been submitted previously either in France or abroad in the same or in a similar version to any other examination body.

Paris, date 27/10/2021

Signature 

To my Family

Résumé

Les technologies de softwarisation des réseaux entraînent de nouvelles architectures de réseau qui remettent en question les systèmes de gestion des défaillance existants et la caractérisation de la résilience. En effet, la coordination entre les différents composants logiciels pour, par exemple, l'orchestration, la commutation et la gestion des machines virtuelles et des conteneurs, implique différents points de supervision et de nouvelles sources de défaillance et de bogues. Dans cette thèse, nous proposons un framework d'automatisation de réseau qui détecte les anomalies et caractérise l'état de résilience d'un service de réseau virtualisé. Un algorithme basé sur les Long Short Term Memory Autoencoder analyse une série temporelle multidimensionnelle construite à partir de centaines de métriques collectées au niveau des couches physique, virtuelle et de service. Il apprend les conditions de fonctionnement nominales de l'infrastructure, sur la base desquelles les déviations (anomalies) par rapport à la référence apprise sont détectées et analysées. Le framework produit une caractérisation des déviations utilisée pour élaborer le graphe d'état et la visualisation sous forme de radiographie. Tandis que cette dernière visualise de manière compacte la propagation des anomalies à travers les trois couches composant un réseau virtualisé, le graphe d'état vise à établir l'état de résilience de la plateforme comme base d'un algorithme de réorchestration qui s'appuie sur une nouvelle technique de gestion de la résilience basée sur la réputation. Le framework est implémenté et validé par des tests expérimentaux sur la plateforme Kubernetes hébergeant un services de cœur de réseau virtualisé conteneurisé et open-source.

Mots-clés : Automatisation des réseaux, détection d'anomalies , reconfiguration , NFV, machine learning, réputation

Abstract

Legacy and novel network services are expected to be migrated and designed to be deployed in fully virtualized environments which lead to novel network architectures that challenge legacy fault management systems and resilience characterization. Indeed, the coordination among the different software components for, e.g., orchestration, switching, and virtual machine and container management creates different monitoring points, besides novel sources of faults and bugs. In this thesis, we propose a network automation framework that detects anomalies and characterizes the resiliency state of a virtualized network service. A Long-Short-Term-Memory-Autoencoder-based algorithm analyzes a multidimensional time-series built from hundreds of metrics collected at the physical, virtual, and service layers. It learns the nominal working conditions of both the infrastructure and the service, and for each type of resource (i.e., CPU, network, memory, and disk); it then detects and analyzes deviations (anomalies) from the learned reference. The produced deviations characterization is finally used to generate both the transition state graph and the innovative radiography visualization. The latter compactly visualizes the propagation of anomalies across all the layers down from the physical and up to the service, highlighting the temporal evolution as well. The former aims at establishing the virtualized platform state as the basis for a re-orchestration algorithm that leverages a novel reputation-based resiliency management technique. We implement and validate the proposed framework through experimental tests on the Kubernetes platform hosting a containerized, open-source, and virtualized network core service.

Keywords : Network automation, anomaly detection, reconfiguration, NFV, machine learning, reputation

Contents

List of tables

List of figures

Chapter 1

Resumé

Les infrastructures de réseaux de communication connaissent aujourd'hui une profonde évolution avec la "softwarisation"de tous leurs composants de base, à partir des fonctions du cœur de réseau à celles de l'accès radio, en passant par les dispositifs mobiles et les objets connectés : fonctions de contrôle d'accès, transport, routeurs, commutateurs et terminaux, notamment les dispositifs de l'Internet des objets (IoT). Comme le montre la figure 2.1, le nombre de dispositifs mobiles et IoT explose déjà, avec des applications liées à la surveillance d'une multitude d'objets allant des services de sécurité, de santé et de contrôle industriel aux maisons intelligentes, aux villes et aux services agricoles.

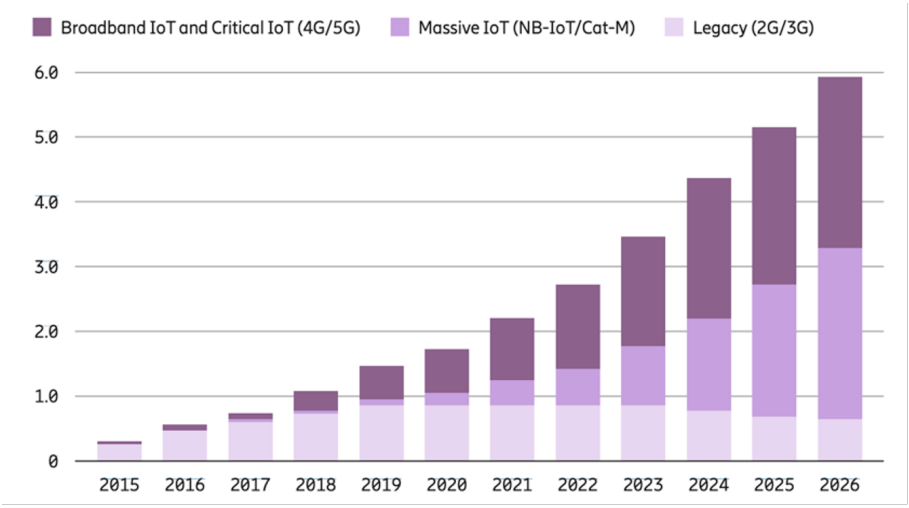


Figure 1.1: Billions of IoT devices [2]

Ces appareils sont de plus en plus conçus comme une composition fonctionnelle de briques logicielles modulaires, qui peuvent chacune suivre des modèles de conception et de développement indépendants,

mais qui doivent finalement fonctionner dans une infrastructure composite qui doit être la plus fiable et la plus efficace possible. Les opérateurs de réseaux doivent donc concevoir des infrastructures flexibles qui doivent simultanément (i) être adaptables en fonction du nombre de dispositifs [1], (ii) garantir le respect des accords de niveau de service (SLA), (iii) être rentables et (iv) assurer une haute disponibilité et une faible latence des services de communication. Pour les services à faible latence et à haute fiabilité, en particulier ceux liés à la sécurité publique, la gestion des SLAs est fondamentale pour pouvoir faire la distinction entre les slices de réseau offrant différents niveaux de service. Puisque les technologies de virtualisation des réseaux et de softwarisation promettent une exploitation rentable des services de réseau avec des SLA garantis, les opérateurs de réseau ont commencé à virtualiser en profondeur les composants de leur infrastructure, ce qui conduit au déploiement de ressources virtuelles sur des plates-formes basées sur des machines virtuelles (VM), des conteneurs et/ou server-less. Essentiellement, la virtualisation des fonctions de réseau (NFV) met en œuvre des fonctions de réseau par le biais de techniques de virtualisation logicielle et les exécute sur du matériel commun (c'est-à-dire des serveurs, du stockage et des commutateurs standard), en découplant le logicielle des fonctions de réseau du matériel sous-jacent. Cela permet d'améliorer la flexibilité des services tout en réduisant le délai de commercialisation d'un nouveau service qui peut être rapidement adapté aux besoins du client grâce à de nouveaux cycles d'innovation comme, par exemple, le DevOps (développement de logiciels, Dev, et les opérations informatiques, Ops).

Malgré cette complexité accrue, les réseaux du futur devront être facilement maintenables et leurs capacités devront être continuellement améliorées et mises à l'échelle en dépendant le moins possible de l'intervention humaine [11]. En ce sens, l'apprentissage automatique (ML) et l'intelligence artificielle (AI) en général, ainsi que la virtualisation et la softwarisation, sont l'un des principaux moteurs de l'automatisation des réseaux. Grâce à la facilité d'accès à des ressources de traitement massives (CPU et GPU), que ce soit sur du matériel on-premise à faible coût ou par le biais de ressources en Cloud à la demande, chaque secteur d'activité peut aujourd'hui optimiser ses processus commerciaux grâce aux algorithmes de Machine Learning (ML). En outre, les plus grands acteurs du cloud privé, comme Amazon et Google, permettent aux entreprises d'exploiter des modèles et algorithmes ML sous la forme de boîtes noires instanciées et utilisées à la demande, sans pratiquement aucun investissement dans la conception et la mise en œuvre. Ainsi, l'accès facile et généralisé au ML ouvre la voie à l'automatisation cognitive des réseaux, pour laquelle la softwarisation fournit les interfaces nécessaires

aux composants du réseau.

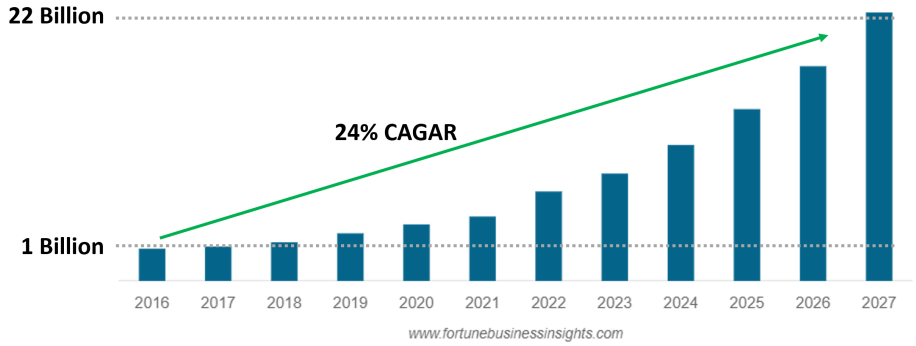


Figure 1.2: North America network automation market size from 2016 to 2027 (USD Billion) [3]

Selon l'étude Market Research Future [3] *la taille du marché mondial de l'automatisation des réseaux était de 4,00 milliards de dollars en 2019 et devrait atteindre 22,58 milliards de dollars d'ici 2027, affichant un taux de croissance annuel composé (TCAC) de 24,2% au cours de la période de prévision.* De même, la recherche sur les algorithmes d'automatisation des réseaux est aujourd'hui dans une large mesure un domaine vert, sans bonnes pratiques établies et sans même un état de l'art suffisant. Des études récentes se sont concentrées sur la conception de ce que l'on appelle la "boucle cognitive", qui fait référence à une boucle de processus composée des phases de détection, d'apprentissage, de décision, de politique et d'action CN1,CN2. Néanmoins, à notre connaissance, il n'existe aucun travail mature qui conçoit une boucle cognitive complète pour les infrastructures de réseau virtualisées.

Dans les sections suivantes, nous résumons brièvement le contenu des chapitres qui composent cette thèse. Pour plus de détails, veuillez vous référer aux chapitres correspondents en anglais.

1.1 Détection et caractérisation des anomalies des systèmes virtualisés

Les anciens et nouveaux services réseau seront migrés et conçus pour être déployés dans des environnements entièrement virtualisés. À partir de la 5G, le NFV devient une brique formellement requise dans les spécifications, pour les services intégrés dans les réseaux des fournisseurs d'infrastructures. Cette évolution conduit au déploiement de ressources virtuelles sur des plates-formes basées sur des machines virtuelles (VM), des conteneurs et/ou des serveurs sans serveur, ce qui nécessite une virtualisation profonde des composants de l'infrastructure. Une telle "softwarisation"du réseau entraîne également une virtualisation plus poussée du réseau logique, facilitant les services multicouches, multi-

acteurs et multi-accès, de manière à pouvoir répondre aux exigences de haute disponibilité, de sécurité, de confidentialité et de résilience. Cependant, l'hétérogénéité accrue des composants rend la détection et la caractérisation des anomalies difficiles, et de même maniere la relation entre la détection des anomalies et la reconfiguration correspondante de la pile NFV pour atténuer les anomalies.

L'automatisation des réseaux est un domaine de recherche dynamique qui vise à déployer de nouvelles solutions dans les réseaux opérationnels au cours des prochaines années. Bien que les premières recherches sur l'automatisation des réseaux remontent en fait à quelques décennies, la véritable automatisation des réseaux alimentée par l'intelligence artificielle (IA) et l'apprentissage automatique (ML) n'est devenue que récemment une possibilité tangible pour les services opérationnels, grâce notamment aux nouvelles technologies liées au Software Defined Networking (SDN) - avec la spécification d'interfaces de configuration ouvertes - et à la Network Functions Virtualization (NFV) - qui rompt le couplage entre les fonctions réseau et le matériel d'hébergement.

Au cours des dernières décennies, la communauté a relevé des défis liés à la manière de laisser des ensembles distribués d'agents s'auto-organiser, découvrir automatiquement les états du réseau et opérer la reconfiguration nécessaire du réseau. Cela a été l'objet de nombreux projets de recherche dans le domaine des réseaux autonomes [161]. Nous pouvons également citer les activités de normalisation liées à l'automatisation des réseaux, comme par exemple celles liées aux protocoles de signalisation autonomes entre les agents décisionnels distribués [162]. Néanmoins, il manquait à ces activités de recherche pionnières une architecture technique de référence stable à partir de laquelle un framework décisionnel pourrait être développé et déployé à grande échelle, par exemple pour résoudre des problèmes d'optimisation du routage ou de l'allocation des ressources. Avec l'avènement des technologies de virtualisation des réseaux, les éléments de référence pour les infrastructures 5G, et au-delà, sont aujourd'hui clairement spécifiés et adoptés. D'une part, la maturité relative des systèmes NFV-SDN a concentré les efforts de spécification de l'industrie sur les interfaces requises pour l'automatisation des réseaux, répondant en quelque sorte à l'attente des anciennes recherches sur les réseaux autonomes, mais avec désormais un environnement opérationnel prêt pour leur intégration. Les groupes Zero-Touch Network and Service Management et Experiential Networked Intelligence de l'ETSI répondent à ce besoin et ont récemment produit un ensemble de spécifications [163, 164]. D'autre part, des plates-formes d'automatisation des réseaux ont récemment vu le jour, notamment l'Open Network Automation Platform, choisie par de nombreux opérateurs comme plate-forme de

référence pour l'automatisation des réseaux [165, 166, 167]. Plus récemment que pour le segment du cœur, le segment radio subit une softwarisation croissante, avec de nouvelles plateformes comme l'Open Radio Access Network one ORAN,ONAP. Ces activités ouvrent la voie à des décisions d'orchestration pour lesquelles il existe un besoin critique d'algorithmes et de méthodes d'automatisation pour (i) déterminer comment l'état d'une infrastructure entièrement virtualisée et programmable, composée d'une variété de modules logiciels, doit être modélisé, (ii) inférer en temps d'exécution, et (iii) supporter l'orchestration automatisée du réseau.

Dans ce chapitre, nous présentons la première brique dans cette direction et proposons une méthodologie pour détecter des anomalies dans l'espace d'état du réseau plutôt non identifié, composé d'un très grand nombre de composants logiciels. Ces composants peuvent être caractérisés par un grand nombre de métriques, changeant en nombre et en comportement dans le temps, qui peuvent être corrélées ou non entre elles, selon les conditions du réseau. Cet environnement indéfini et variable nous motive à proposer un framework d'apprentissage automatique non supervisé pour la détection d'anomalies dans les infrastructures NFV. Le framework a été appelé SYRROCA (SYstem Radiography and Root Cause Analysis). La figure 4.5 visualise le framework de manière schématique.

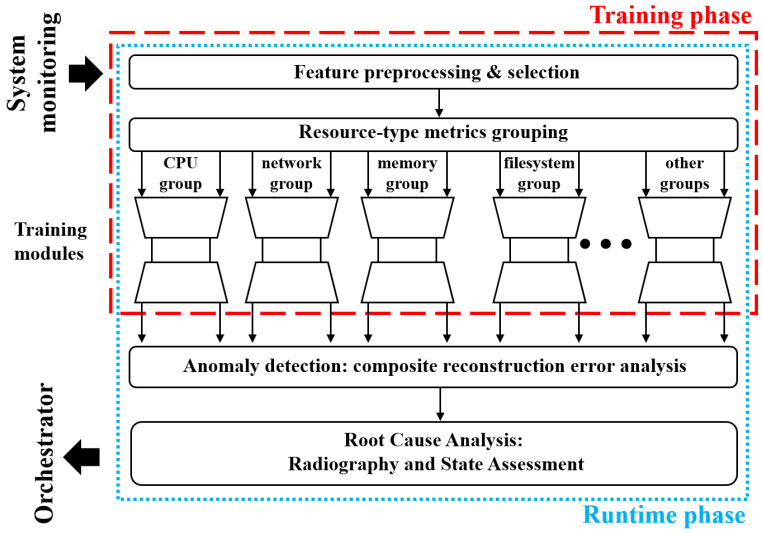


Figure 1.3: SYRROCA framework functional diagram

Nous avons effectué des tests dans une architecture virtualisée Ip Multimedia Subsystem IP (IMS) visualisé dans la figure 4.2, le framework traditionnel utilisé pour le routage et le traitement du trafic de la voix sur IP. Les distributions d'appels simulées et les ensembles de données utilisés sont disponibles

à l'adresse suivante [170].

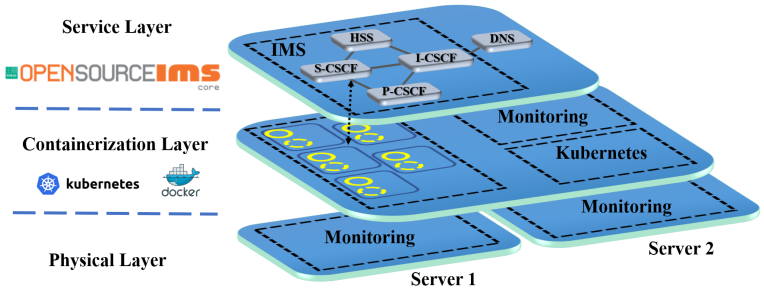


Figure 1.4: Testbed

1.2 Évaluation de l'état du système virtualisé

La softwarisation des réseaux facilite l'adoption d'approches dites "cognitives", c'est-à-dire d'un processus en boucle fermée composé de phases de détection, d'apprentissage, de décision, de politique et d'action [12, 13]. Les observations capturées par les capteurs (sense) aident à construire un modèle à partir des observations utiles (learn), qui est à son tour utilisé par un module de décision pour choisir (decide) les actions à entreprendre sur la base des mouvements possibles et de l'expérience acquise. Les actions potentielles, c'est-à-dire les stratégies stockées dans le module de politique (policy), sont présélectionnées par le module de planification, de sorte que, finalement, les actionneurs exécutent (act) les reconfigurations sélectionnées [11]. Les politiques développées par la boucle cognitive visent à atteindre un objectif final de bout en bout dicté par les exigences de l'entreprise et/ou de l'utilisateur, comme le maintien d'une certaine qualité de service (QoS) pour respecter un accord de niveau de service (SLA). Construire un modèle suffisamment précis de l'état du réseau à partir des données détectées est une étape primordiale dans la boucle cognitive/automatique. Dans un environnement logiciel, des outils de surveillance avancés récents (par exemple, Prometheus) permettent de récupérer des milliers de métriques à différents niveaux pour détecter une plateforme connectée composée de composants informatiques et de réseaux. Cependant, l'extraction automatique de caractéristiques pertinentes à partir d'une telle quantité de données pour évaluer l'état du système est un défi que nous avons abordé dans le chapitre précédent, où nous avons posé les bases du framework SYRROCA. Dans ce chapitre, nous développons davantage le framework SYROCCA, en proposant une caractérisation de l'état du système à travers les couches. Nous montrons comment cette nouvelle analyse permet de

caractériser les anomalies à n'importe quelle couche et leur propagation à travers les couches. Nous effectuons des tests dans la même architecture vIMS conteneurisée que nous avons présentée dans le chapitre précédent. Nous proposons également une formalisation du problème de remédiation des anomalies autonomes sous la forme d'un algorithme d'apprentissage par renforcement. Enfin, nous proposons un modèle de confiance et de réputation pour quantifier la contribution des entités MANO à la résilience des systèmes autonomes.

1.3 Méthode de reconfiguration automatisée

Dans les chapitres précédents, nous avons présenté le framework SYRROCA qui couvre la phase de détection de la boucle d'automatisation du réseau en détectant et en caractérisant les anomalies à travers les couches physiques et virtuelles. Cette phase consiste à apprendre les conditions de l'état nominal pour ensuite identifier et caractériser quelles ressources du réseau (groupe de ressources, unité de calcul) et dans quelle mesure sont déviées. Dans la section suivante, nous présentons l'architecture d'une méthodologie de récupération automatisée capable de compenser la déviation en s'appuyant sur l'évaluation de l'état du système SYRROCA. L'approche proposée est basée sur un algorithme d'apprentissage par renforcement (RL) qui apprend à sélectionner les actions de remédiation les plus appropriées en ciblant les ressources dont la caractérisation de l'état est la plus éloignée des conditions de travail nominales. Nous montrerons comment SYRROCA, enrichi d'un modèle de réputation et d'un agent RL, complète le framework d'automatisation cognitive en boucle fermée. Ce chapitre décrit principalement le brevet et la conception possible du système.

1.4 Remarques conclusives et perspectives

Dans cette thèse, nous avons présenté et évalué expérimentalement une boucle de contrôle autonome complète pour la gestion de la résilience des réseaux virtualisés.

La figure 7.1 résume notre proposition sous forme de différents blocs appliqués à l'architecture fonctionnelle NFV. Le cœur de notre proposition est le framework SYRROCA, qui est une approche basée sur un LSTM-autoencodeur pour détecter (B) et caractériser (C) les anomalies d'une série temporelle multivariée composée de centaines de métriques collectées aux niveaux physique et de virtualisation (A) d'une plateforme logicielle supportant un service de réseau virtualisé. Nous avons montré com-

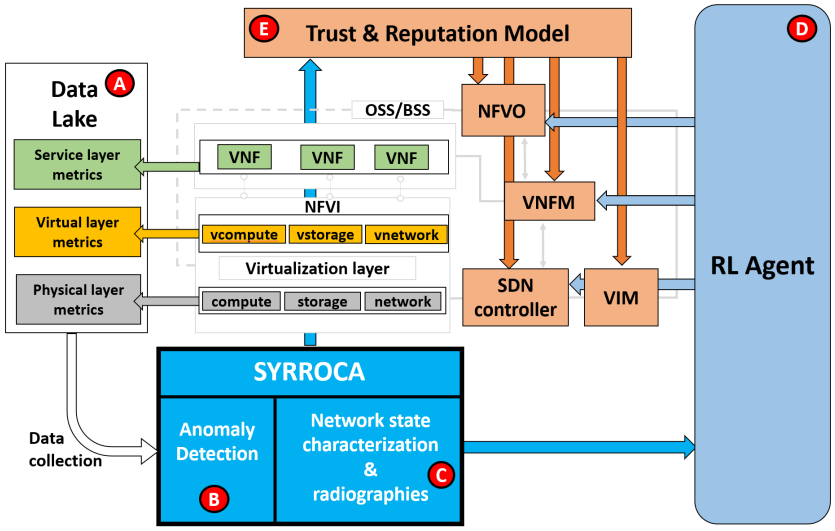


Figure 1.5: SYRROCA closed loop automation on ETSI NFV architecture

ment les métriques brutes peuvent être agrégées et prétraitées pour constituer un jeu de données d'entraînement sur lequel les autoencodeurs profonds LSTM peuvent apprendre une représentation compacte des conditions nominales de fonctionnement du système softwarisé. En effet, grâce aux propriétés de compression des auto-codeurs, la série temporelle d'entrée de haute dimension est comprimée dans un sous-espace de plus faible dimension (l'espace latent) où les échantillons anormaux semblent significativement différents des échantillons nominaux. Pour améliorer les performances de l'auto-codeur, nous divisons l'ensemble de données d'entrée en groupes de mesures par type de ressource et nous alimentons chaque groupe avec un auto-codeur dédié. En considérant les types de ressources CPU, réseau, mémoire et disque, nous avons un ensemble de quatre AE pour la couche physique et quatre autres pour la couche de virtualisation. Pour analyser correctement les métriques à croissance monotone et celles de forme arbitraire, nous ne conservons que les incréments des premières tout en normalisant à une gamme uniforme les deux catégories de métriques. Nous avons choisi les RNN LSTM comme cellules d'auto-codage de base car ils permettent de traiter des séries temporelles arbitrairement longues sans perdre la mémoire des échantillons loin dans le passé comme cela se produit dans les RNN standard. Nous avons comparé les auto-codeurs SYRROCA basés sur les LSTM avec un RNN de base et une approche de forêt d'isolement (ISF), prouvant que les auto-codeurs RNN améliorent à la fois le rappel et les scores F2 par rapport aux ISF, qui ne parviennent pas à extraire d'informations sur les données non étiquetées. Nous avons également montré que les LSTM augmentaient encore les performances des auto-codeurs par rapport aux RNN grâce à leur capacité à stocker des dépendances

à long terme.

Ensuite, nous avons montré comment exploiter l'EQM des AE pour caractériser les anomalies détectées à la fois par l'ensemble des métriques les plus déviantes et par la visualisation du roman radiographique. En particulier, nous avons démontré comment l'EQM des auto-codeurs est corrélée à l'intensité de l'anomalie, ce qui permet à notre framework d'évaluer la gravité de l'anomalie. Nous avons ensuite expliqué comment les radiographies peuvent visualiser les anomalies de réseau sur un espace euclidien bi-dimensionnel où la densité des fonctions bi-variées $f(MSE^l, g, MSE^{l+1,g})$ de deux couches consécutives AE erreurs quadratiques moyennes, repère l'état du système le plus récurrent sous la fenêtre temporelle considérée. Nous avons présenté deux versions d'un ensemble de radiographies $(L-1) \times G$ pour G groupes de ressources d'un système composé de L couches. La version de radiographie à fenêtre glissante met en évidence l'évolution de l'anomalie dans le temps tandis que la version à fenêtre croissante montre la propagation de l'anomalie à travers des couches consécutives. Dans les deux versions, les seuils des AE identifient une zone rectangulaire qui correspond aux conditions normales de fonctionnement.

La détection d'anomalies basée sur l'autoencodeur et la caractérisation de l'état du système ont toutes deux été évaluées expérimentalement sur une plateforme vIMS gérée par Kubernetes. Nous avons injecté un trafic réaliste basé sur des profils d'appels réels extraits d'un LAC (Location Area Code) donné du réseau 3G d'Orange. Nous avons également injecté trois types d'anomalies pour évaluer les performances de détection et de caractérisation de SYRROCA. Le jeu de données généré est disponible à l'adresse [170].

Nous avons ensuite proposé une méthodologie pour évaluer l'état de fonctionnement du système logiciel (C) en se basant sur la caractérisation des anomalies produite à l'étape B. La méthodologie produit un graphe d'état qui est destiné à alimenter un moteur de décision dans un système d'automatisation en boucle fermée : comme chaque état dégradé est caractérisé par l'ensemble des métriques les plus déviées, les politiques d'orchestration peuvent être adaptées à ces déviations. Selon En effet, l'apprentissage par renforcement correspond parfaitement au processus cognitif qu'un réseau autonome devrait mettre en œuvre, et les diverses entités logicielles qui composent la pile SDN/NFV constituent une accroche idéale pour le processus cognitif. En conséquence, l'agent RL intègre toutes les entités actives du système dans un agent de niveau supérieur qui gère la résilience du système softwarisé par le biais d'intentions de haut niveau, mises en œuvre avec une ou plusieurs actions

d'entités. L'état de l'agent RL est obtenu par le bloc C d'évaluation de l'état du SYRROCA, qui calcule également la récompense des intentions à partir des radiographies à fenêtre coulissante comme une évaluation de la qualité de l'intention. Plus précisément, la récompense est obtenue comme la somme des contributions de chacune des $(L-1) \times G$ radiographies. Chaque contribution est inversement proportionnelle à la distance de la région à haute densité par rapport à l'origine de l'espace euclidien de la radiographie et directement proportionnelle à un facteur $m_{\hat{x}'}^{l,g} + m_{\hat{y}'}^{l,g}$.

Chapter 2

Introduction

Content

2.1 Background and motivation

Communication network infrastructures are nowadays experiencing a deep evolution with the softwarization of all their basic components, from core network functions to radio access ones, including mobile devices and connected objects: access control functions, transport, routers, switches, and terminals in particular Internet of Things (IoT) devices. As depicted in figure 2.1, the number of mobile and IoT devices is already exploding, with usages on the monitoring of endless examples of objects wearable or not varying from security, health, and industry-related control services to smart homes, cities, and farming services.

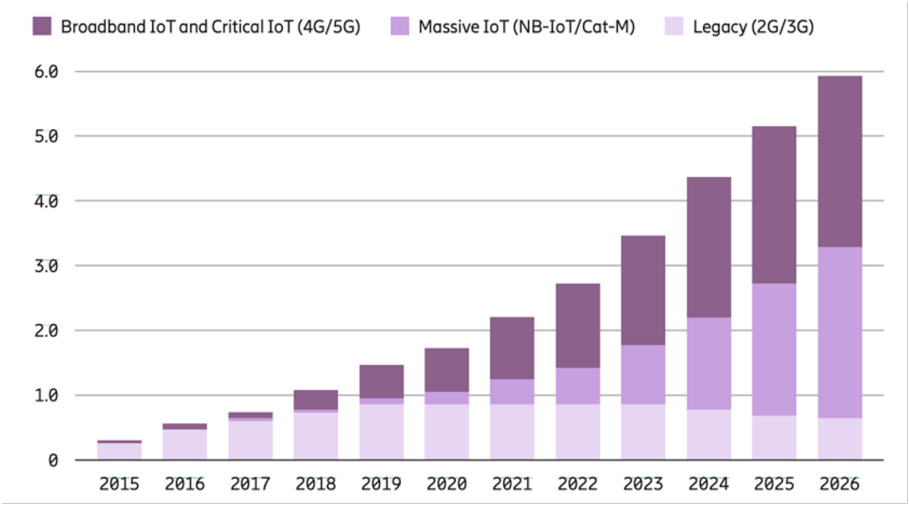


Figure 2.1: Billions of IoT devices [2]

These end-devices are more and more designed as a functional composition of modular software bricks, each of them potentially following independent design and development patterns but eventually having to work in a composite infrastructure that must be as reliable and efficient as possible. Thereby, network operators need to design flexible infrastructures that simultaneously have (i) to be scalable with the number of devices [1], (ii) to guarantee Service Level Agreement (SLA) compliance, (iii) to be cost-effective and (iv) to ensure high availability and low latency to communication services. For low-latency and high-reliability services, especially those related to public safety, SLA management is fundamental to be able to discriminate among network slices offering different grades of service. As network virtualization and softwarization technologies promise a cost-effective operation of

network services with guaranteed SLAs, network operators started deep virtualization of their infrastructure components which leads to the deployment of virtual resources Virtual-Machine (VM)-based, container-based and/or server-less platforms. Essentially, Network Function Virtualization (NFV) implements network functions through software virtualization techniques and runs them on commodity hardware (i.e., industry-standard servers, storage, and switches), decoupling the software implementation of network functions from the underlying hardware. This brings improved flexibility of service provisioning while decreasing the time to market of a new service that can be rapidly tailored to customer needs thanks to novel innovation cycles like, for instance, software development (Dev) and IT operations (Ops).

In spite of the increased complexity, future networks should be easily maintainable and their capabilities should be continuously improved and upgraded by relying as little as possible on human intervention [11]. In that sense, machine learning (ML) and artificial intelligence (AI) in general, along with virtualization and softwarization, are one of the key enabler for network automation. Thanks to easy access to massive processing resources (CPU and GPU) both on low-cost on-premise hardware or through on-demand cloud resources, every industry can nowadays optimize its business process through ML algorithms. Furthermore, recently the largest private cloud players like Amazon and Google, allow companies to take advantage of ML models and algorithms in the form of black-boxes instantiated and used on-demand with almost no investment in design and implementation. Thereby, widespread and easy access to ML paves the way to cognitive network automation for which softwarization provides the necessary interfaces to network components. Indeed, according to the Market Research Future [3] *the global network automation market size was USD* 4.00 *billion in 2019 and is projected to reach USD* 22.58 *billion by 2027, exhibiting a Compound annual growth rate (CAGR) of* 24.2% *during the forecast period.*

Likewise, research in network automation algorithms is today to a large extent a green field, with no established good practices and not even sufficient state-of-the-art. Recent studies focused on the conception of the so-called 'cognitive loop', which refers to a process loop consisting of sense, learn, decide,policy and act phases [12, 13]. Nonetheless, to the best of our knowledge, there is no mature work that conceives a full cognitive loop for virtualized network infrastructures.

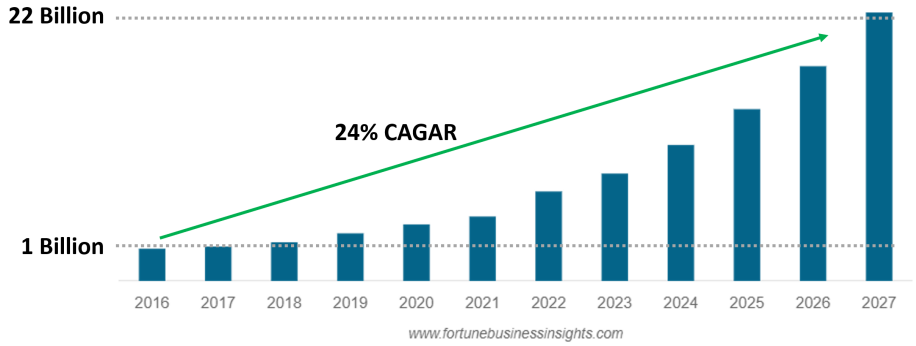


Figure 2.2: North America network automation market size from 2016 to 2027 (USD Billion) [3]

2.2 Contributions and thesis outline

This manuscript is organized as follows.

In Chapter 3 we first introduce all the concepts, background and related work fundamental to understanding network automation closed loops for resiliency management. We first present network virtualization with reference to network function virtualization, software defined network and cloud computing, outlining the synergy among the three technologies as network operators envision it for the networks of the future. We then define network resiliency as long as presenting a brief overview of the state of the art modeling efforts with a focus on automated resiliency management loops requirements. After introducing the concept of reputation, we conclude the chapter with a brief overview of machine learning techniques used in the dissertation.

In Chapter 4 we describe the proposal for the first brick towards a fully automated network resiliency management framework. We detail the machine learning pipeline we proposed to process raw datasets composed of network metrics collected at any layer composing a softwarized network infrastructure. We also showcase the anomaly detection algorithm we designed to characterize deviation from the learned nominal working conditions. The novel radiography visualization is also introduced as a tool to visualize anomalies propagation among layers.

Chapter 5 is dedicated to the network system state assessment we proposed in [14]. The virtualized system state is classified whether nominal or degraded. In the latter case, it is characterized through the most deviated computation units and resources.

In Chapter 6 we describe the mechanism we propose to master distributed decision process of multi-

agent resiliency management of softwarized infrastructures. We formalize the anomaly remediation action selection as a reinforcement learning problem, which is then used to build the reputation assessment system. We deposited patent [15] for this proposal.

Chapter 7 concludes the dissertation with perspectives for future work improvements of the presented solutions.

Annexes A presents the dataset metric names we used for the thesis work. Annexes B and C report a study on traffic classification conducted during the thesis and a technical report on ONOS and ODL Software Defined controllers performance comparison.

2.3 Publications

Journals

Diamanti A., Vilchez J.M.S. and Secci S.,"An AI-empowered framework for cross-layer softwarized infrastructure state assessment". IEEE Transactions on Network and Service Management, Major revision.

Conferences

Diamanti A., Sanchez Vilchez J.M. and Secci S., "LSTM-based radiography for anomaly detection in softwarized infrastructures "in 32nd International Teletraffic Congress, 2020.

Diamanti A., Sanchez Vilchez J.M. and Secci S.,"The SYRROCA AI-empowered network automation platform."in 24th Conference on Innovation in Clouds, Internet and Networks and Workshops, 2021.

Aureli D., Cianfrani, A., Diamanti A., Sanchez Vilchez J.M. and Secci, S.,"Going beyond diffserv in ip traffic classification". In IEEE/IFIP Network Operations and Management Symposium, 2020.

Technical reports

Secci S., Diamanti A., Sanchez Vilchez J.M., et al., "Security and Performance Comparison of ONOS and ODL controllers.", Open Networking Foundation, Informational Report, September 2019.

Patents

Diamanti A., Sanchez Vilchez, J.M. and Secci S., "Procédé de contrôle d' une entité d'orchestration dans un réseau logiciel ". French Patent, 2021. Submitted under number FR2107239.

Chapter 3

Related Work

Content

This chapter introduces the concepts, the background, and the related work that are fundamentally correlated to virtualized network resiliency management. We describe the main virtualization techniques used by network operators focusing on the introduced architectural novelties and challenges. We then review state-of-the-art closed-loop automation paradigms focusing on crucial challenges of cognitive resiliency management. Trust and reputation are then introduced according to the state of the art definition and models. To conclude, we outline an overview of the main machine learning techniques used in this manuscript.

3.1 Network softwarization

The idea of virtualizing physical resources through a software-based abstraction in order to optimize available resources utilization was first introduced in the late 60s by IBM with the CP/CMS (Control Program/Cambridge Monitor System) discontinued time-sharing operating system [16]. The OS allowed the sharing of the available computing resources, dividing the activity of the CPU into time intervals. Since then, virtualization greatly evolved to encompass plenty of different paradigms which allow the abstraction of several hardware types other than classic computing resources.

In the context of telecommunications operators, three virtualization paradigms mainly shaped the evolution of network infrastructures: the Network Function Vritualization (NFV), the Software Defined Networking (SDN) and the Cloud Computing. In the following paragraphs we detail each of the three paradigms.

3.1.1 Network Function Virtualization

Traditionally, the provisioning of a telecommunication network service was based on the deployment of proprietary and vendor-locked specialized hardware according to a chain of functions specifically tailored to each service. This lack of elasticity in network infrastructures coupled with highly demanding service quality, security and stability impose heavy deployment and reconfiguration process which in turn bring high CApital EXpenses (CAPEX) and OPerating EXpenses (OPEX). Nevertheless, in the last decade, the demand for broadband network connectivity has been increasing dramatically not only because of the increase in the number of Internet-connected mobile devices but also for new (short-lived) services with high data rates, like in sensor networks and machine-to-machine (M2M) connectivity.

To cope with all of these challenges, in October 2012, a group of telecom operators proposed a *Call for Action* [17] which led to the creation of the Network Functions Virtualization (NFV) Industry Specification Group (ISG) within the European Telecommunications Standards Institute (ETSI). This ISG was initiated by several leading telecommunication carriers, including AT&T, BT, China Mobile, Deutsche Telekom, Orange, Telefónica, and Verizon. It has quickly attracted broad industry support, and had over 150 members and participants by the end of 2013, ranging from network operators to equipment vendors and IT vendors. NFV, as intended by the ISG, leverages virtualization technology,

such as Virtual Machines (VM) and containers, to decouple physical equipment from the functions that run on them. This way, a given Network Service (NS) can be decomposed into a set of Virtual Network Functions (VNFs) which are deployed as plain software on commodity off-the-shelf hardware which could be located in data centers, distributed network nodes or at end-user premises. A VNF can be further decomposed into multiple components (VNFC) when the different functionalities of a VNF feature specific performance, security, availability or scalability requirements.

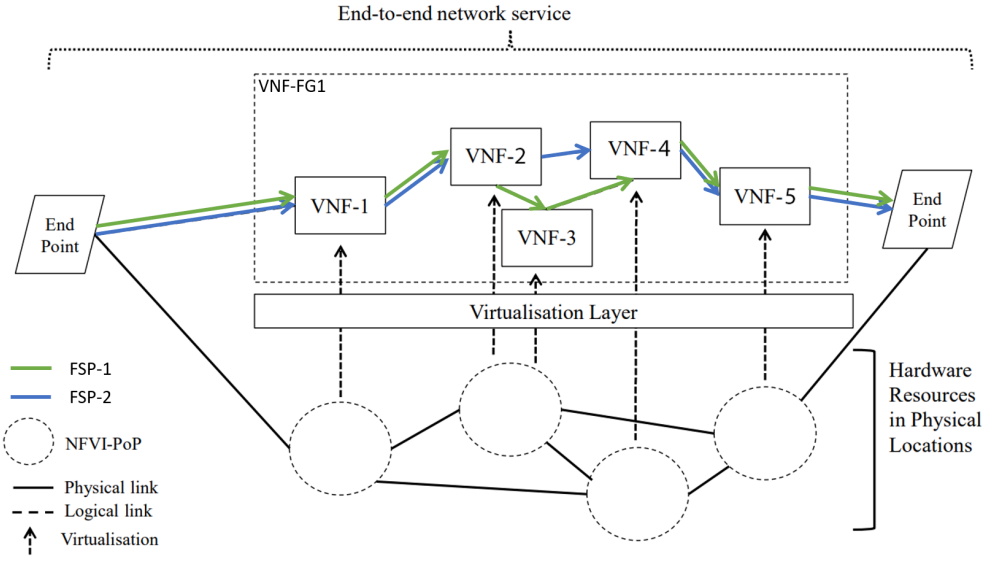


Figure 3.1: NFV-FG composed of two FSP [4]

To describe an end-to-end NS, the NFV ETS ISG introduced the concept of Network Function Forwarding Graph (VNF-FG); as depicted in figure 3.1 the graph is composed of nodes representing the VNFs and edges for the logical links defining the existing connection between VNFs. As different packet flows may need to be processed according to specific policies, the same VNF-FG may include multiple sets of function chains, referred as Forwarding Service Paths (FSP). Classifiers are thus used to match incoming packets with pre-defined rules in order to map them to the correct FSP.

To operate virtual network services with the required flexibility, VNF should run on a framework that includes dynamic initiation and orchestration of VNF instances [18]. Therefore, the ETSI NFV ISG defined the functional NFV architectural framework, using functional entities and reference points, without any indication of a specific implementation. As depicted at figure 3.2, the architecture leverages on two main blocks:

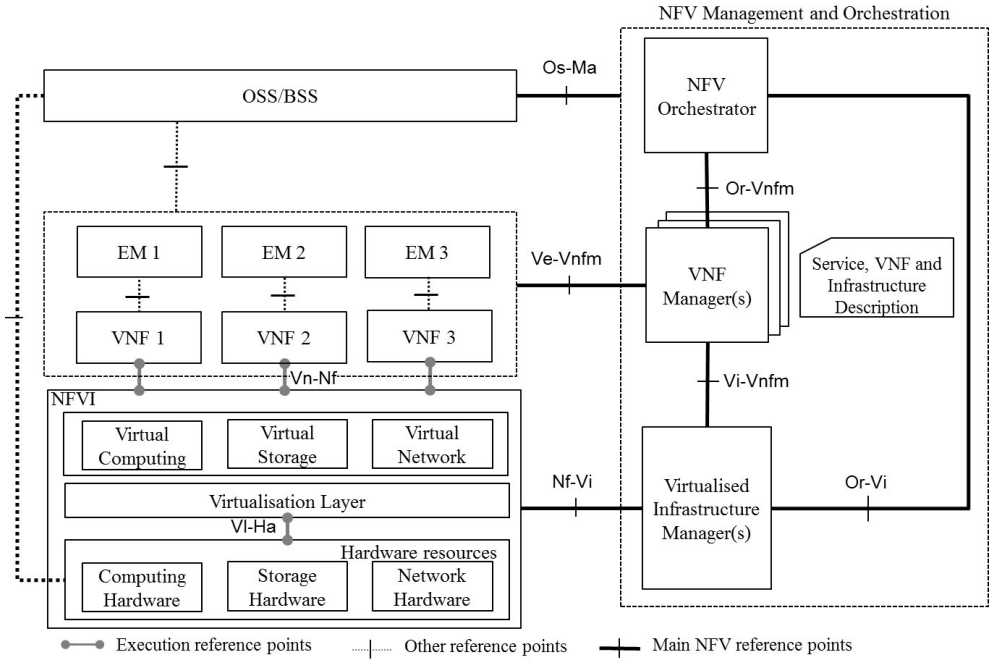


Figure 3.2: The functional NFV architectural framework [4]

- Network Function Virtualization Infrastructure (NFVI): represents the infrastructure of an NFV environment, i.e. it provides all the hardware and software resources for the instantiation of network functions. Managed resources may be both virtualized and non-virtualized, supporting partially VNFs as well;

- NFV MANagement and Orchestration (NFV MANO): encloses all the entities that have the role to manage the NFVI and orchestrate the allocation of resources needed by the Network Services (NS) and VNFs.

When deploying VM-based VNFs, the NFVI virtualization software is the *hypervisor*, which is the piece of software in charge of abstracting the host physical resources to guest operating systems [19]. However, this flexibility of hardware usage comes at a price. Firstly, VM creation and instantiation require a non-negligible amount of time: after the hypervisor allocates resources to the VM and the desired OS is installed on the VM, the entire boot process needs to be performed, like for a bare-metal machine. Furthermore, run-time performance not only depends on the amount of allocated resources, but each operation, whether an I/O or any instruction, needs to be translated by the

hypervisor in order to be correctly mapped to the host hardware. Even using dedicated hardware to accelerate virtualization or directly access some host components like it happens with network card in Single Root I/O Virtualization (SR-IOV), VNF scalability and portability are greatly impacted by the heavy virtualization burden. On the contrary, as depicted in figure 3.3, container-based virtualization provides a different level of abstraction: rather than executing a full OS on top of the virtualized hardware, containers implement isolation of processes at the OS kernel level [20]. In particular, Linux containers are implemented through *control groups* (cgroups [21]) and *namespaces* [22] kernel features. While the latter provides a mechanism to restrict the view of certain resources to the container, the former feature allows processes to be organized into hierarchical groups whose usage of resources can be limited and monitored. Due to that, containers instantiation time is greatly reduced compared to VM, code can execute as if it was run on the host machine and I/O devices are directly accessed. Nonetheless, containers do not isolate resources as hypervisors do because the host kernel is exposed to the containers, which can be an issue for multi-tenant security [23].

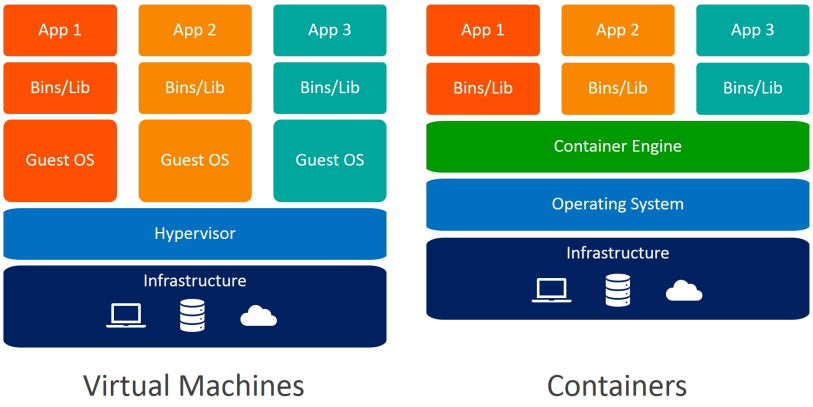


Figure 3.3: Container and VM based virtualization [5].

As the virtualization principle stimulates a multi-vendor ecosystem where the different components of NFVI, VNF software, and NFV-MANO architectural framework entities are likely to follow different lifecycles (e.g. on procurement, upgrading, etc.), ETSI proposed a set of interoperable and standardized interfaces to let the different entities intercommunicate. Furthermore, as NFV-MANO alone cannot deliver all the NFV business benefits, the architecture also includes an interface to integrate and inter-work with Operations support systems and Business support systems (OSS/BSS). While the OSS is responsible for maintaining the Quality of Service (QoS) in accordance with the

Service Level Agreement (SLA) and provides assistance to the customer in case of problems, the BSS deals with business-related issues such as customer order processing and payment. Communications among those interfaces are facilitated through a shared resource abstraction model that is built up leveraging on five different descriptors [24]:

- VNF Descriptor (VNFD): characterizes the identification details of the VNF (e.g., Name, ID, Provider), its connection points, and any resource requirements for its execution;
- Virtual Link Layer Descriptor (VLD): describes links between VNFs, end-points and VNFCs;
- VNF Forwarding Graph Descriptor (VNFFGD): models the Forwarding Service Paths (FSP) and the classifiers used to define a chain of VNFs;
- Physical Network Function Descriptor (PNFD): describes the Physical Network Functions (PNFs) regarding their interconnection and performance requirements;
- NS Descriptor (NSD): models the end-to-end NS as an aggregation of the others type of descriptors.

As proposed by ETSI NS descriptors (NSD) and VNF Descriptors (VNFD) should also contain a set of N *monitoring_parameter* that describe the different metrics that should be collected. In particular, for each VNFD, three attributes require the specification of related monitoring metrics: Virtual Deployment Unit (VDU), VNF Virtual Link Descriptor (VnfVirtualLinkDesc) and VNF Deployment Flavour (VNDF). The VDU is a construct supporting the description of the deployment and operational behavior of a VNFC mapping it to to a single virtualization container (e.g. a VM). Hence, the monitored parameter covers the single VFC. The VnfVirtualLinkDesc information element supports providing information about the internal VNF Virtual Links (VLs); monitoring parameters concern then metrics about intra-VNF and intra-VNFC virtual links. The VnfDf describes a specific deployment version of a VNF, thus the monitoring parameters concern the virtualized resource-related performance metrics to be tracked by the VNFM. Similarly, at the service level, each NSD defines for each Network Service Deployment Flavour (NsDf) the set of performance metrics to be monitored on an NS level (e.g. calls-per-second, number-of-subscribers, no-of-rules, flows-per-second, etc.). Figure 3.4 depicts the structure of the *monitoring_parameter* attribute we find in the descriptors.

Attribute	Qualifier	Cardinality	Content	Description
id	M	1	Identifier	Unique identifier of the monitoring parameter.
name	M	0..1	String	Human readable name of the monitoring parameter.
performanceMetric	M	1	String	Identifies the virtualised resource performance metric.
collectionPeriod	M	0..1	Not specified	An attribute that describes the periodicity at which to collect the performance information.

Figure 3.4: Monitoring Parameter information element from ETSI specification [6]

3.1.2 NFV Management and Orchestration

The MANagement and Orchestration (MANO) components of the ETSI NFV architecture are in charge to coordinate network resources for cloud-based applications and the lifecycle management of virtual network functions (VNFs) and network services. As such, it is crucial for ensuring rapid, reliable NFV deployments at scale.

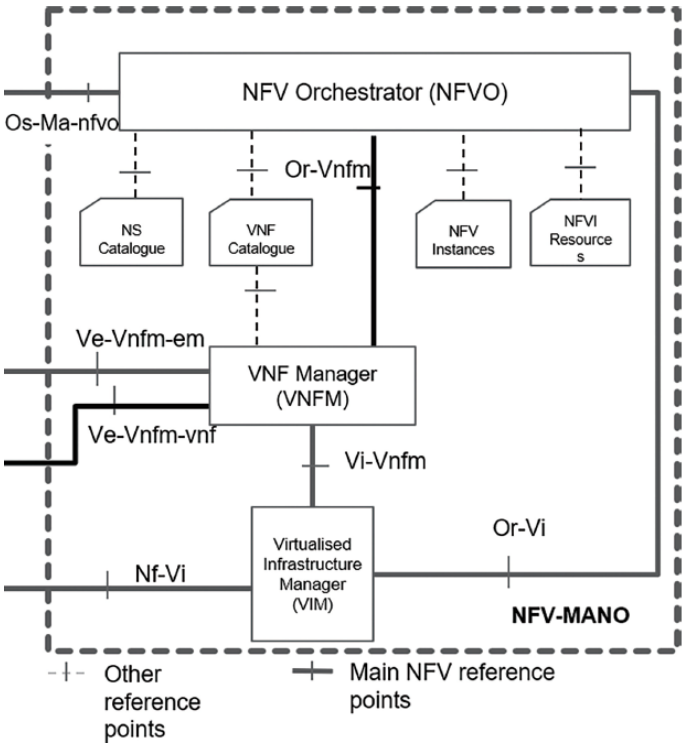


Figure 3.5: ETSI MANO architecture [4]

As depicted in Figure 3.5, the main MANO functional component are:

- NFV Orchestrator (NFVO): it is in charge of managing NS life-cycle instantiating, scaling,

modifying and terminating NS. It is composed of two sub-components: the Resource Orchestrator (RO) and the Network Service Orchestrator (NSO). While the former communicates with the VNFM[s] (through the Or-Vnfm interface) to correctly compose the VNF to obtain the desired NS, the latter communicates with the VIM[s] (through Or-Vi interface) to orchestrate the NVFI resources;

- VNF Manager (VNFM): it manages the life-cycle (that is setup, configuration, maintenance and tearing down) of a single VNF or a set of multiple VNFs. In some implementations, it also provides FCAPS (Fault, Configuration, Accounting, Performance and Security) functions generally provided by a dedicated component called Element Management System (EMS). When VNFs require proprietary interfaces to fulfill FCAPS operations, the EMS is functionally separated from the VNFM and it is provided by the VNF vendor. In this case, the VNFM features two standard open interface/reference points, one to communicate with the EMS (Ve-Vnfm-em) and another with the VNF (Ve-Vnfm-vnf);

- Virtualized Infrastructure Manager (VIM): it manages the association of the virtualized resources to the physical compute, storage and networking resources of the NFVI. Furthermore, it collects and forwards to the NFVO performance measurements and events.

It is worth noticing that the ETSI MANO architecture also includes the databases that are used to store the information and data models which define both deployment as well as life-cycle properties of functions, services, and resources. With an effective MANO architecture, service onboarding is straightforward and can be performed by a telco operator relying on template specification and onto an automated mechanism to translate specification in the desired deployment. As we will see in the next paragraphs, this automation pattern is an essential enabler and a driver towards future autonomous cognitive networks.

In both the industry and academia, there exist several projects closed or open-source which try to implement the elements of the MANO software stack. Open Source MANO (OSM) [25] is an ETSI-hosted open source community delivering a production-quality MANO stack for NFV, capable of consuming openly published information models, available to everyone, suitable for all VNFs and VIM-independent (API interaction). OSM is aligned to NFV ISG information models while providing first-hand feedback based on its implementation experience. Similarly, Tacker [26] is an official

OpenStack [27] project building a Generic VNFM and an NFVO to deploy and operate NS and VNFs on an NFVI platform like OpenStack. While OpenStack is a set of interrelated software components that control diverse, multi-vendor hardware pools of processing, storage, and networking resources throughout a data center, Tacker completes the stack with a VNFM and a NFVO. Open Baton [28] is another similar effort that supports several VNFM and VIM solutions. It provides a generic VIM (combined with its own Generic EM System) which can be used for managing VNF Packages. Furthermore, it is built to be extensible either over a message bus using the pub/sub mechanism or using a RESTFul API to support additional external VNFMs.

The Kubernetes container orchestrator

Kubernetes is an open-source container-orchestration system that aims to provide a platform for automating deployment, scaling, and operations of application containers across clusters of hosts. It implements the pattern of multiple cooperating processes which form a cohesive unit of service through the 'Pod' abstraction, which constitute a group of containers sharing storage and network.

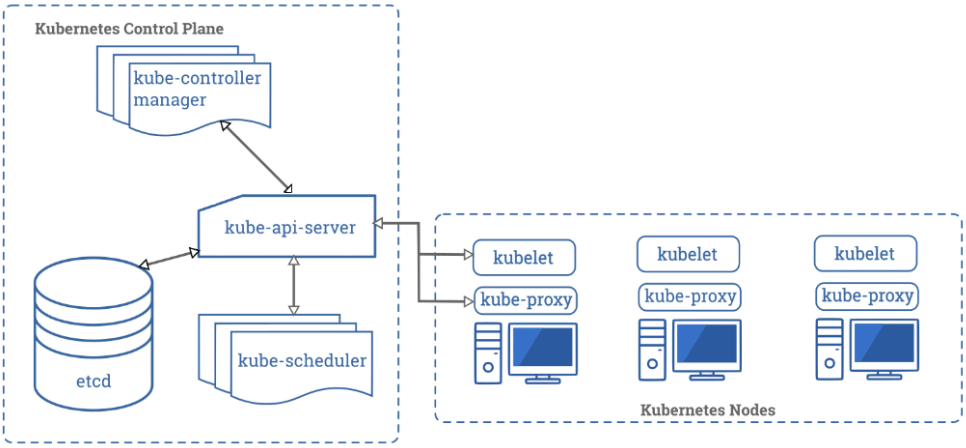


Figure 3.6: Kubernetes architecture [7]

As depicted in figure 3.6, the Kubernetes control plane expose an API server that provides the front-end to the cluster's shared state and the interface for the users and the other components to manage, create, and configure Kubernetes clusters resources and workloads. A workload is an application running on Kubernetes that can be composed a set of workload resources:

- Deployment. The former is a good fit for managing a stateless application workload on the

cluster, where any Pod in the Deployment is interchangeable and can be replaced if needed;

- StatefulSet lets users run one or more related Pods that do track state somehow. For example, if your workload records data persistently, you can run a StatefulSet that matches each Pod with a PersistentVolume. Your code, running in the Pods for that StatefulSet, can replicate data to other Pods in the same StatefulSet to improve overall resilience;
- DaemonSet defines Pods that provide node-local facilities. These might be fundamental to the operation of the cluster, such as a networking helper tool, or be part of an add-on. Every time you add a node to your cluster that matches the specification in a DaemonSet, the control plane schedules a Pod for that DaemonSet onto the new node;
- Job and CronJob define tasks that run to completion and then stop. Jobs represent one-off tasks, whereas CronJobs recur according to a schedule.

Their specification and desired state are expressed in a declarative fashion, through either .yaml configuration files or the *kubectl* command line tool, and they are set through API calls. A distributed key-value store called *etcd* stores resources desired state and specification into Kubernetes *objects*, which also records the actual state for the resource the object describes. Desired state enforcement is performed through the so-called *controllers*, which implements several core control loops all of them managed by the *kube-controller-manager*. A controller tracks at least one object and issues command towards the API server to make the actual state as close as possible to the desired state. For instance, the *Node Controller* is responsible for node life-cycle managements which encompass nodes CIDR block assignment, health monitoring, available resource tracking etc... When deploying pods on the physical nodes composing the cluster, Kubernetes relies on a component called scheduler which finds feasible nodes for a Pod selecting the optimal one. When a pod is created on a node, the *kubelet* node agent daemon uses *probes* to ensures that the containers in it run healthy. A probe describes a health check to be performed against a container to determine whether it is alive or ready to receive traffic. If the associated user-defined conditions are not meet, the kubelet daemon can react to restart, replace or kill containers.

Networking among cluster pods is strictly regulated by a fixed model which is stated through the Container Network Interface (CNI) specification. CNI is a Cloud Native Computing Foundation project, consists of a specification and libraries for writing plugins to configure network interfaces in

Linux containers. CNI promotes interoperability among container runtimes and orchestrators providing a common interface between the network plugins and container execution. The CNI imposes that pods on a node can communicate with all pods on all nodes without NAT and agents on a node (e.g. system daemons, kubelet) can communicate with all pods on that node. Those two fundamental requirements enable low-friction porting of apps from VMs to containers. Another fundamental principle that the CNI imposes is the "IP-per-pod" model, in which containers within a Pod share their network namespaces, including their IP and MAC addresses. This means that containers within a Pod can all reach each other's ports on localhost and that thus they must coordinate port usage, as it happens for processes in a VM. In the context of Kubernetes, there exists plenty of CNI plugins which implements those specifications in a number of way and leveraging on different technologies, like for example Linux bridges (Calico), openVSwitches (ovs-cni) or AWS/GCP/Azure virtual routers. External pod exposure as a network service is instead managed by the *service* abstraction. Pods are inherently ephemeral, which means that the set of Pods running in one moment in time could be different from the set of Pods running that application a moment later. Accordingly, the service abstraction constitutes a single entry-point for a logical set of Pods and a policy by which to access them. The set of Pods targeted by a Service is determined by a *selector*. The *kube-proxy* network proxy which runs on each node, is the control-plane component in charge to load-balance traffic that is destined for services to the correct back-end pods. To do that, it can rely on three different technologies (userspace, iptables and IPVS) which may require the CNI plugin to make the traffic accessible to them.

3.1.3 Software Defined Networking

Despite their widespread adoption, traditional IP networks are complex and hard to manage [29]. In fact, the implementation of a given network policy requires a low-level configuration of the vendor-closed network devices. Furthermore, the decision on how to route traffic and the mechanism used to forward packets according to the desired policies, are bundled together inside the network devices. This inherent rigidity of the network makes challenging to implement any kind of automation or auto-reconfiguration while hindering innovation and evolution of the infrastructure [30].

Software Defined Networking (SDN) was then proposed to bring more elasticity breaking the vertical integration of control and data planes through the software-based Controllers (SDNC) which runs on a commodity server and centralize the management into a logical layer separated from the data

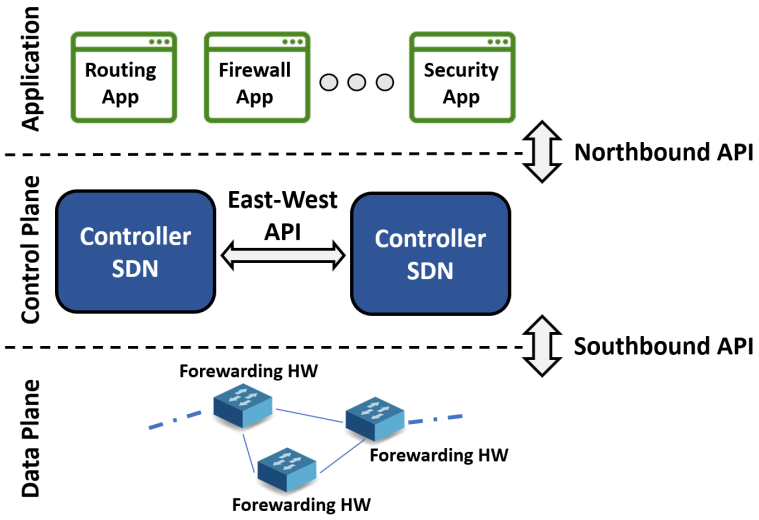


Figure 3.7: Software Defined Networking architecture

forwarding function. Figure 3.7 depicts a high level architecture as it was conceived by a research collaboration between Stanford and the University of California at Berkeley [31] which proposed the OpenFlow southbound Application Programming Interface (API) and protocol as a way for the SDNC to control the flow of traffic over a network. The key idea of OpenFlow is to program the forwarding hardware to match incoming data-plane packets against the forwarding rules that the controller pushes through the OpenFlow itself. Each rule associates to the match conditions certain actions (like dropping, forwarding, modifying, etc...) that are performed on the matching traffic (flow). Rules are organized in tables, which can be chained in order to implement different forwarding policies and make the hardware devices behave like different network components (like firewalls, routers, switches etc..). Even though OpenFlow is the first and probably the most used southbound API specification and protocol, there exists other southbound protocols and associated APIs like the Network Configuration Protocol (NetConf) which uses an Extensible Markup Language (XML) to communicate with the forwarding hardware. In general, a southbound API not only enables control over the forwarding operations but also lets the switching hardware exposing event notifications, statistical reporting and advertise the capabilities of the network. An SDN controller also exposes a northbound API which allows programming SDN Applications to access an abstraction of network functions, to consume the network services and dynamically configure the network. SDN applications can for example implement enable basic network functions like path computation, loop avoidance or put in place more sophisti-

cated services like load balancers, software defined security and orchestration applications across cloud resources [32].

Even thought SDN holds great promise in terms of simplifying network deployment, maintenance and evolution, many challenges remain to be addressed. When using general-purpose forwarding hardware, high-level programming language enables fast development and the highest level of design abstraction [33]. But this flexibility comes at a price: CPUs are indeed limited to several tens of Gb/s of throughput when processing network packets [34]. Optimized processor architectures for network processing, like Network flow processors (NPUs/NFPs), were introduced to increase processing speed up to 200 Gb/s; however, the flexibility of implementation is reduced as the definition of the processing functions requires detailed knowledge of the device architecture. Further, less flexible but more energy efficient and performing, Application-specific Integrated Circuits (ASICs) are also broadly used as forwarding hardware, mainly along with NPUs/CPUs to compose a hybrid architecture that provides a desirable programmability/performance trade-off [35]. Additionally, the SDN controller software implementation may also affect the processing performance: for example, in [36] authors showed how the Open Network Operating System (ONOS) controller turns out to be more efficient at processing Openflow packets than the OpenDayLight controller.

Other broadly discussed challenges that arise from the centralization of the control plane in one logical node, concern the security and the high-availability/resilience of the SDN controller. Despite SDN controllers can be clustered through the east-west API to assure availability in case of one node fault, the softwarized nature of the SDN controller opens the way for new types of faults and threats that did not exist in legacy hardware-based networks. At the controller-application level, questions have been raised around authentication and authorization mechanisms for applications and users as without proper securing an attacker may masquerade as a controller and carry out malicious activities. Furthermore, the adoption of open interfaces and protocols allows a complete knowledge of the same by eventual attackers [37].

The flexibility and the network programmability offered by of the SDN paradigm, pave the way to the so-called intent-based networking [38]. Firstly introduced by the ONOS and OpenDayLight open source projects, intents should allow a network application to declare the desired network behavior in plain natural language without being responsible for implementing that behavior itself [39]. The actual intent rendering, that is the translation into device-specific rules and courses of action, should

be automatically managed by the SDN controller. Even though the paradigm was conceived for the SDN, it is particularly interesting with reference to network automation and declarative network configuration as it allows the specification of policies rather than mechanisms. For example, when a given service request is received, the VNF-MANO framework has to convert that request into a suitable service graph and pass it to the relevant VIMs involved in the SFC composition. In this context, intents allow to provide an abstracted yet flexible definition of the requested service graph, without knowledge of the technology-specific details [40]. Furthermore, intents can be useful to express all the goals – including those that may have been considered "common sense" in human-operated systems- that a cognitive loop needs to target at: intents, indeed, give the system the flexibility to explore various solution options and find the optimal one through learning [41].

3.1.4 Cloud Computing for networking

With the advent of the Internet and the progressive development of computing resource virtualization, the locus of computation shifted outward to distant data centers [42]. Consequently, cloud computing was at first intended as the possibility for users to rent virtual computer resources to execute their own applications. Since then, the concept of cloud computing greatly evolved to encompass other service models, which demonstrated to be the best candidate that offers a chance of achieving the efficiency and expense reduction that are motivating TSPs towards NFV [43]. In particular cloud computing offers three service models [44]:

- Software as a Service (SaaS): users can access and exploit applications through a web browser or a programming interface. The software is hosted and fully managed by the cloud provider;
- Platform as a Service (PaaS): customers are provided with a complete cloud platform—hardware, software, and infrastructure—for developing, running, and managing applications without the cost, complexity, and inflexibility that often comes with building and maintaining that platform on-premises;
- Infrastructure as a Service (IaaS): users can access compute, network, and storage resources on-demand, over the internet, and on a pay-as-you-go basis.

As IaaS is the most open-ended cloud service providing access to virtual and general-purpose computing, networking and storage resources, it constitutes a cost-effective approach for telecommu-

nication operators to implement the NFVI layer. Fully managed cloud infrastructures not only relieve the operators from the burden to set up and maintain the whole virtualization infrastructure, but it gives a cost-effective way to shrink and scale resources to meet the variable business requirements, eliminating up-front capital expenditures or unnecessary owned infrastructure. Similarly, the PaaS model will push even further the possibility for telco operators to deploy and maintain services, providing all the required software to instantiate, maintain, reconfigure, aggregate and monitor network services.

Even if both IaaS and PaaS seem to be suitable to facilitate the migration to fully virtualized networks, cloud environments still need an adaption so as to obtain carrier-class behavior. In fact, telecommunication networks call for very strict performance requirements like low data-plane latency [45] and infrastructure five-nine availability and reliability [46], among others, which are not inherently granted by a generic cloud environment. For example, one of the architectural tenets of cloud platforms is that both scalability and reliability are achieved via massive horizontal scale [47]. This pushes the High Availability (HA) requirement from the infrastructure itself as in old-fashioned network, up to the application: recognizing that failures are bound to occur, an application with many instances across many hosts in a distributed cloud can grant the five-nine availability requirements.

When it comes to design and develop cloud-native scalable applications in dynamic environments such as public, private, and hybrid cloud, technologies such as containers, service meshes, microservices, immutable infrastructure, and declarative APIs are commonly adopted [48]. These techniques enable loosely coupled systems traditionally designed according to Service Based Architectures (SBA) like microservices or Service-Oriented Architecture (SOA). SBA patterns place a heavy emphasis on services as the primary architecture component used to implement and perform business functionality [49]. SOA promotes the reuse of services which are accessed through synchronous RESTFull APIs and synchronously obtain and alter data directly at its primary source. Microservices instead are a specialization of an implementation approach for SOA used to build flexible, independently deployable software systems [50]: code reuse is preferred, accepting duplicate data to help improve decoupling. Furthermore interaction patterns based on asynchronous communication, like publish/subscribe event sourcing, are adopted to enable a microservices component to remain up to date on changes happening to the data in another component. Following the introduction of DevOps, which combines a set of best practices of software development (Dev) with IT operations (Ops), microservices begun the most

popular architecture for building continuously deployed systems [51]. In fact, as experienced in IT, when developing several units potentially deployable to any environment, Continuous Delivery (CD) and Continuous Integration (CD) DevOps best practices, among others, are of paramount importance to decrease the time between changing a system and transferring that change to the production environment [51].

3.1.5 Fully softwarized networks

By allowing network functions to be virtualized and run on commodity hardware, NFV enables cloud-native service models, like SBA, to be applied to operators network services. In this direction, the 3rd Generation Partnership Project (3GPP) standardized 5g as a service-based architecture in order to conceive the future operator core network as a cloud-native framework. As mentioned in the previous section, cloud-like environments are particularly suited to implement SBA since VMs and containers provide a perfectly suited abstraction to deploy independent services. Furthermore, the cloud management and orchestration schemes coupled with SBA provide all the elasticity that NFV requires for the instantiation, migration and reconfiguration of VMs running specific network services according to the VNFM policies. In that sense, the cloud management and orchestration fulfill the functionality expected from a VIM and thus integrates into the NFV architecture. Likewise, SDN seems to be the natural solution to obtain a coherent fully softwarized infrastructure where each VNF is managed by the NFV MANO through the cloud orchestrator and the connectivity among the different cloud nodes and tenants that hosts the virtualization units (container or VMs) is configured through the SDN controller. Furthermore, NFV and the cloud provide the SDN with the possibility of implementing network functions through the software on commodity servers. Accordingly, the SDN controller can be virtualized so that it can be easily maintained, extended and integrated in the MANO pipeline.

As the future of all network services is directed towards fully softwarized architectures both at the level of network functions and at the level of connectivity management, an open research question is how to manage these infrastructures in order to ensure the same quality of the service as traditional networks while providing all those functionalities that the future networks, such as 5G, require. As we will show in the following paragraphs, one of the methods that have been hypothesized is to build a *self-conscious* network, that is therefore, able to self-regulate starting from a consciousness of its

state that uses as a base to understand autonomously how to reconfigure itself in case of anomalies or fluctuations of the service in order to guarantee the desired level of service.

3.2 Network resilience

Definitions

Service continuity is not only a customer expectation but often a regulatory requirement, as telecommunication networks are considered to be part of critical national infrastructure. Moreover, system failure not only results in loss of revenue for the provider, but it can seriously damage its reputation. A Highly Available (HA) network is one that ensures that the network and its services are nearly always accessible. An HA of five nines (99.999%, meaning that on average, the service is never down for more than five minutes in a one-year period) or six nines is the minimum requirement that telco operators often want to enforce. However, it is not enough: a network and its services also need to be *Resilient*. Resilience stems from the Latin *resilio*, which literally means "jump backwards", accordingly a resilient network has the ability to provide and maintain an acceptable level of service recovering from various faults and challenges to normal operation [52]. It is also seen as a combination of HA and the ability to maintain QoS Service Level Agreement (SLA). Consider a routing service that can grant a level of 99.9999% HA but which is not resilient: it will not go down for more than 31.5 seconds a year but if it does it may not capable of restoring active sessions without any degradation. On the contrary, a resilient routing system will manage faults and continue the service without any disruption or degradation in performance, maintaining the minimum level of service as defined in the SLA. Networks resiliency is a very broad property, widely studied both in the research community and in the industrial environment, winch is in turn defined as a composition of several sub-properties:

- **Survivability**: The capability of a system to fulfill its mission, in a timely manner, in the presence of threats such as attacks or large-scale natural disasters [53, 54];
- **Disruption tolerance** represents the capability of tolerating episodic loss of connectivity among its components;
- **Traffic tolerance** as the ability of the system to to tolerate traffic beyond the design parameters without a significant drop in carrier load;

- **Reliability**: is the probability that a system or service remains operable for a specified period of time. It is mathematically defined as the probability of the system or service to stay operational at time t and it is expressed as: $R(t) = \exp -(\int_0^t \lambda(\tau)d\tau)$, where $\lambda(\tau)$ is the instantaneous failure rate. Through the reliability it is possible to compute the Mean Time Between Failure (MTBF), i.e. the predicted elapsed time between inherent failures, as $MTBF = E(T_{BF}) = \int_0^{+\infty} R(t)dt$. When the instantaneous failure rate is a constant $\lambda(t) = \lambda$, the reliability is $R(t) = e^{-\lambda t}$; hence the $MTBF = 1/\lambda$. Similarly, given a constant repair rate μ, the Mean Time To Repair (MTTR) is defined as $MTTR = \frac{1}{\mu}$ [55].
- **Availability**: readiness for correct service [56], i.e., delivery of service in compliance with the service specification. Measured as the probability of the readiness to provide service compliant with the requirements. It is mathematically defined as $A(t) = \frac{MTBF}{MTBF+MTTR} = \frac{\mu}{\lambda+\mu}$;
- **Security**, declined as the property of a system, and the measures taken such that it protects itself from unauthorized access or change, subject to policy [57];
- **Performability**: is the property of a system to deliver the service according to specification [58].

To correctly define system resilience, it is important to understand how challenges can impact its working condition. In particular, a fault is a flaw in the system that can cause an error. This can either be an unforeseen design flaw (such as a software bug), or a foreseeable flaw due to constraints that permit an external challenge to cause an error (such as not designing a sufficiently robust system due to cost constraints). On the other end, an error is a deviation between an observed value or state and its specified correct value or state that may lead to a subsequent service failure. A service failure (frequently shortened to failure) is a deviation of the service from the desired system functioning such that it does not meet its specification or expectation. Accordingly, a fault may be triggered to cause an observable error, which may result in a failure if the error is manifest in a way that causes the system not to meet its service specification.

Resiliency modeling

Effectively and comprehensively modeling the resiliency of a network is an open state of the art challenge. The major complexities come from the varied nature of services that the network provides, the numerous layers and their parameters over which these services depend, and the plethora of

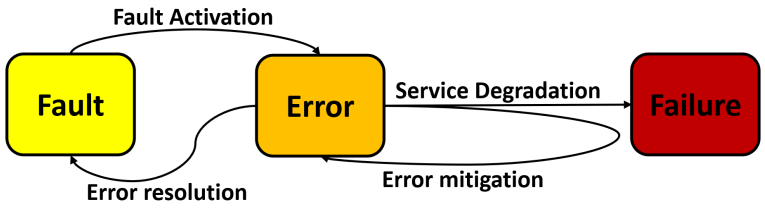


Figure 3.8: Fault-error-failure chain

adverse events and conditions that present challenges to the network as a whole. For instance, [59] formalizes the notion of Level-of-Resilience (LoR) in IP network. This measure was based on the Level-of-Stability-Reduction (LoSR), as measured by the percentage of IP traffic dropped, the Level-of-Performance-Reduction (LoPR) as measured by the percentage of reduction in application Quality-of-Service (QoS) latency, and the network Recovery-Time (RT) as measured by convergence time under various attack scenarios. Gertsbakh et al. [60] formalizes the notion of probabilistic resilience in a network N with n components as the largest number of component failures such that N is still operational with probability $1 - \beta$.

Authors in [52, 61, 62] model networks resilience as a function of the deviation between two network states caused by anomalies. The network state S_k is represented as a tuple $(\mathbb{N}_k, \mathbb{P}_k)$ where $\mathbb{N}_k$ represents the operational state of the network derived from a set of operational M metrics $\{p_1, ..., p_l\}$ (e.g. number of link failures, link capabilities or congestion etc..). $\mathbb{P}_k$, instead, characterizes the service delivered by that layer (e.g. IP connectivity) and is obtained from the set $\{n_1, ..., N_s\}$ of service related metrics (e.g. delay, jitter, etc.). Accordingly, they defined the bi-dimensional state space depicted in figure 3.9. For the seek of simplicity, they divided the Service dimension in *Acceptable*, *Impaired* and *Unacceptable* regions, while the Operational states are divided in *Normal Operation*, *Partially Degraded* and *Severely Degraded*. In this space, network resilience is then characterized as a function of the deviation between the nominal state S_0, which represents the normal working conditions for the network, and a degraded state S_c, which results from an anomaly. In particular, they defined a resiliency score as $\mathbb{R} = 1 - A(S_n, S_d)$, where $A(S_n, S_d)$ is the area under the resilience trajectory $S_0 ß S_c$ depicted in figure 3.9 as a the shaded triangle.

What is particularly interesting about this model is its multi-layer nature: considering all the layers composing a network, it is possible to obtain a resiliency state-space like the one of figure 3.9 for each boundary $B_{i,j}$ between layers i and j. The service parameters at the boundary $B_{i,j}$ become

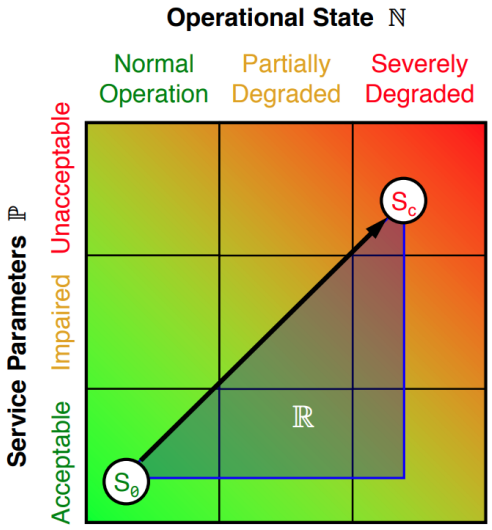


Figure 3.9: Sterbenz resiliency space [8]

the operation metrics at boundary $B_{i+1,j+1}$. By beginning at the bottom level and progressing up the service layers, the overall multilevel resilience value can be computed, and by composing these across all scenarios of interest for a given network architecture, it may be possible to derive a single resilience value.

Resilience considerations for NFV

Resiliency is one of the central topic studied by the e ETSI NFV ISG which produced a documents [9] to define the requirements a resilient NFV virtualized network environment needs to fulfill, and a Proof of Concept (PoC) [63] to demonstrates how virtualized network functions (VNFs) from multiple vendors can be easily deployed in a highly resilient software infrastructure environment.

According to ETSI, there are a number of alternative means by which the virtualized applications can achieve their resiliency objectives: on the one end, relying totally on the capabilities of the NFV-MANO entities to detect errors and effect remediation, or the resiliency could be managed by the virtualized application itself and relies only partly or not at all on the NFV-MANO functionality. In both cases, as the NFV-MANO entities could be in turn virtualized, they need to be at list HA (they should support recreation to original state prior to failure). In fact, as they are involved in orchestrating the infrastructure as well as managing the VNF lifecycle, they need to be always at least available.

As each NS could require a different degree of resiliency, the network service descriptors and the VNF descriptors should specify the reliability, availability and scalability SLAs to be supported. Furthermore, when onboarding NS and setting up the related VNFs, it is expected the Orchestrator and/or the VNF Manager set up monitoring and register for notifications to the VIM. The monitoring should span the whole stack, down from the physical service, passing through the virtualized resources and VNFs, and up to the NS. Indeed, in the event of faults, monitoring triggers should either activate VIM remedial actions or report to the NFVO. Thereby, anomalies affecting an NFV system may trigger fault notifications at different levels, several of which could have causal relationships to one or more faults (for example, a low memory condition in a virtual machine could lead to several faults at the application level). To avoid individual handling of the single notifications that may led to unnecessary concurrent actions, ETSI propose a fault correlation framework (figure 3.10) composed of *system-wide* (SC) and *local correlators* (LC). The latter runs at different layers in the system and collect failure

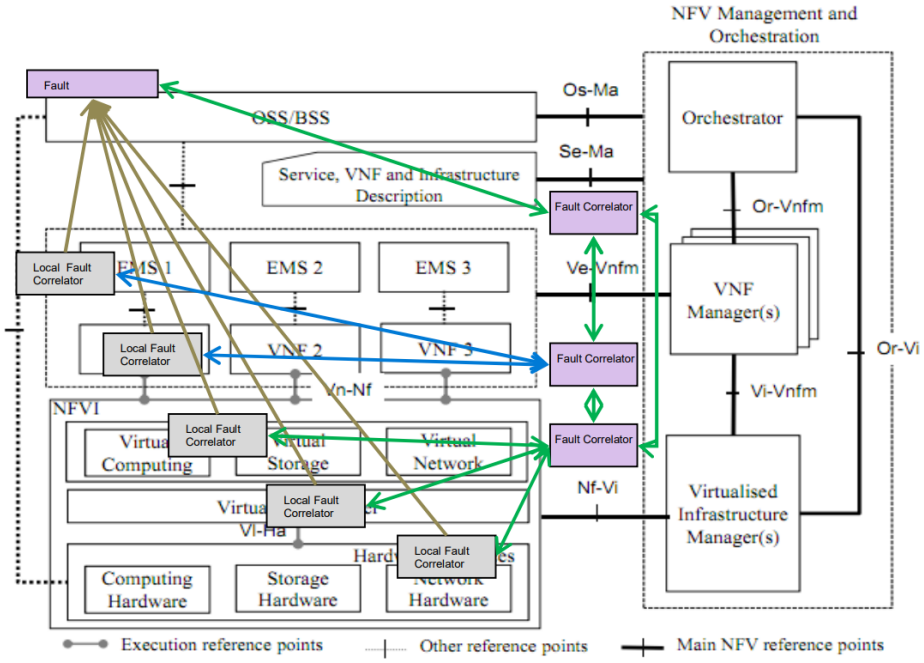


Figure 3.10: ETSI NFV fault correlation framework [9]

information occurring at different components within that layer. They apply well-defined correlation rules to select one or more root cause candidates that might have caused all other errors reported at that layer. The former, collect reports of LCs and other SCs to apply correlation rules based on a common fault precedence graph. One of the main limitations of this proposal is how to define

correlation rules for LCs and the fault precedence graph. In spite of that, the important takeaway is that ETSI highlighted the need for coordination in analyzing errors and/or anomalies detected at each layer to perform a joint analysis. As we will se in the next chapter, this is design specification is important for cognitive control loops as well.

To the best of our knowledge, there exist only very few works that try to implement the ETSI NFV recommendations. For example, the OPNFV Doctor [64] project aims at building a fault management and maintenance framework for the high availability of Network Services on top of virtualized infrastructure. The key feature is immediate notification of unavailability of virtualized resources from VIM, to process recovery of VNFs on them. However, the project does not provide for any automatic recovery mechanism. Similarly, the Open Network Automation Platform (ONAP) is a software platform for the orchestration, management, and automation of network and edge computing services for network operators, cloud providers, and enterprises. Its real-time, policy-driven orchestration and automation of physical and virtual network functions enables rapid automation of new services and complete lifecycle management critical for 5G and next-generation networks. It integrates with Openstack, Kubernetes and other different public clouds. Cloudify [65] is an open source cloud orchestration framework that allows you to model applications and services, automate their entire life cycle, monitor and detect issues and failure.

3.2.1 Cognitive closed loop automation for resiliency management

In spite of the future networks increased complexity we highlighted in the previous paragraphs which appears to be approaching the limits of human capability, they are expected to be easily maintainable with as little as possible of human intervention. In March 2001 during a keynotem Paul Horn introduced the idea of *Autonomic Computing* referring to computing systems that can manage themselves given high-level objectives from administrators [10]. As depicted in figure 3.11, an autonomic manager was presented as a component that continually executes a *Monitor-Analyze-Plan-Execute* (MAPE) loop in which it observes sensor readings, analyzes and plans an appropriate management decision, and then executes that them via effectors. The model encompasses a central knowledge base as well: it contains an explicit system model, which estimates how the management decisions would affect subsequent states of the managed element and their effectiveness in achieving the manager objectives.

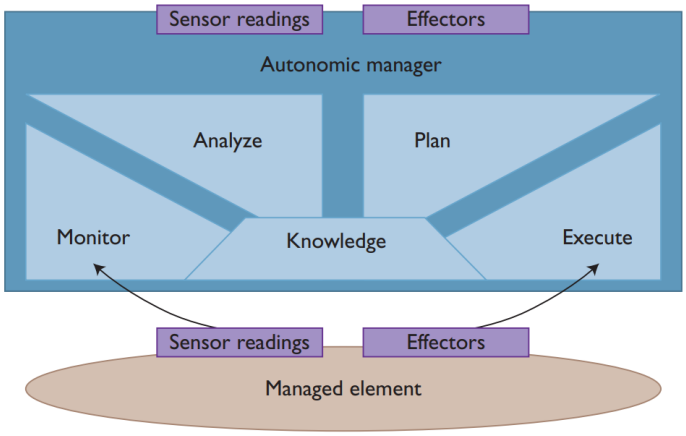


Figure 3.11: MAPE control loop [10]

Inspired by this publication, the research community applied the autonomic computing concept to several domains and further developed a set of properties, referred to as *self-**, that are associated to autonomic systems [66]:

- **self-protecting**: refers to the capability of defending the system against correlated failures such as external attacks, massive disasters, or cascading effects. Supports self-healing systems when those are not able to deal with such problems;
- **self-healing**: it is the capability of detecting abnormalities, diagnose them and identify the root cause, i.e. the element or elements origin of the abnormality. The abnormality may imply a service failure, a simple fault having non-disastrous consequences on the service layer, or even a mismatch in a given operational parameter that could evolve in a service failure;
- **self-configuring**: it regards the automatic configuration of the network equipment by means of high-level policies. This implies that, when new equipment is connected to the network, it should self-advertise by sending its capabilities and the system should configure it automatically as well as providing its capabilities to the rest of network equipment to be aware of this new equipment;
- **self-optimizing**: the automatic setting of parameters of the equipment installed in the network in order to optimize the global behavior of the network.

In the context of networking, autonomic computing was first studied by Mitola [8] which proposed the so-called *cognitive radio architecture*, depicted in figure 3.12. The cognitive loop enables the radio

system to sense the external environment, learn from history and make intelligent decisions to adjust its transmission parameters according to the current state of the environment. Likewise, a general cognitive closed loop automation refers to a closed-loop process consisting of sense, learn, decide, policy and act phases [12, 13]. The observations captured by the sensors (sense) help to build a model from the useful observations (learn), which is in turn used by a decision-making module to choose (decide) the actions to be taken based on possible moves and learned experience. Potential actions, i.e. strategies stored in the policy module (policy), are shortlisted by the planning module, so that, finally, the actuators execute (act) selected re-configurations [67].

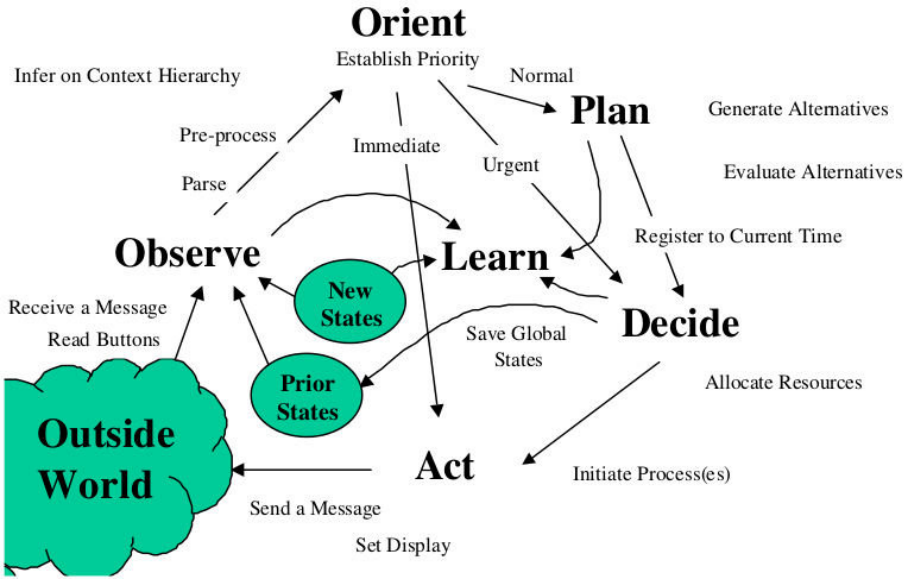


Figure 3.12: Cognitive cycle from Mitola [8]

One of the key concepts of a generic cognitive closed loop is its cross-layer design referred to as the ability to share information among layers and actively exploiting the dependence between the protocol layers to obtain performance and achieve high adaptivity [68]. Classic computer networks have been designed following the principle of protocol layering, so network functionalities are designed in isolation of each other (separate layers) and interfaces between layers. Each layer uses the services provided by the layer below it and provides services to the layer above it. Inter-layer communication happens only between adjacent layers and is limited to procedure calls and responses. When implementing a cognitive control loop each layers parameters need to be aggregated and analyzed as a whole to help the loop determine the best adaptation rules with regard to the current network state. For networks that employ a wide variety of layers and protocols, a cognitive loop can provide a mechanism to view

and learn from the network as a whole [69].

Since Mitola first proposed an application of autonomic computing to networks, more demanding new applications and services have been introduced, which in turn pushed the growth of the complexity and heterogeneity of network and equipment. In this regard, autonomics is seen as the next generation of management solutions for telco operators, mainly for its self-healing property which enables discovery of malfunctions through fault-detection, diagnosis and appropriate actions triggering to prevent disruptions. The idea to build a control loop to manage the resilience of the network operator system, is not novel in the literature. In [52] Sterbenz et. al. propose a double control-loop, known as $D^2R^2 + DR$: Defend, Detect, Remediate, Recover, and Diagnose and Refine. The framework is based on two strategies: first to heal the net-work at run-time with short-term measures (D^2R^2) and to refine and improve the network with medium-term or long-term measures by a background loop (DR). Those long-term measures may be architectural changes or the inclusion of new mechanisms and algorithms, or new protocols to face new vulnerabilities and challenges. Both control loops are continuously interacting in order to take into account the feedback of the D^2R^2 control loop to foresee new updates and upgrades of the resilience system. Similarly, [70] describes an integrated framework for the design, evaluation, and deployment of network resilience strategies. These strategies describe the management behavior of a number of federated, policy-controlled resilience mechanisms, such as monitoring and detection systems. The framework allows the generalization of the most effective policy configurations into reusable management patterns, which can then be rapidly deployed in the network infrastructure. The proposed framework leverages on four main abstractions: (i) a challenge analysis component which outputs challenge events indicating that a new challenge has been identified; (ii) The resilience manager, which realizes a policy-driven feedback control-loop, which is based on the challenges observed and mechanism instances available; (iii) a Policy-based RESilience simulaTor (PReSET) which allows the modeling of resilience strategies, facilitates the off-line analysis of a range of anomalies and attack behaviors, and permits the evaluation of resilience policies to detect and mitigate security threats; (iv) management patterns which can be seen as a generalized resilience strategy that is used to address a particular type of network challenges, and consists of a policy configuration between a set of resilience mechanisms and their relationships.

[71] presents the BioRAC (Biologically-inspired Resilient Autonomic Cloud) framework which employs biologically inspired techniques and multi-level tunable redundancy techniques to increase attack

and exploitation resilience in cloud computing. The solution leverage on a Cooperative Autonomic Manager (CAM), which is assigned to a cluster of resources and is in charge of setting up, running and maintain the cloud service on its cluster as defined at the organization and functional levels. Furthermore, the CAM is able to self-adapt in a timely manner to the security policies and re-configure the cloud resources in order to eliminate newly discovered threats and stop their propagation.

When it comes to implementing a cognitive loop on existing and future networks, on of the hindrance than make challenging to implement a MAPE-like control loop, is the complexity in engineering a sufficiently accurate knowledge module that can achieve acceptable performance in deployed systems. Recent advances in network softwarization and programmability through SDN and NFV, the proliferation of new sources of data, and the availability of low-cost and seemingly infinite storage and compute resources from the cloud are paving the way for the adoption of ML to realize this cognitive network management in support of autonomic networking. Cognitive control loops are then generally empowered with machine learning algorithms [72, 73]. Regardless of the implementation, not only the agent needs to observe the result of the chosen action, but also to learn from that result and optimize his behavior in such a way as to achieve a final end-to-end goal dictated by the business and/or user requirements such as maintaining a certain Quality of Service (QoS) to fulfill a Service Level Agreement (SLA).

3.2.1.1 Network state assessment

In cognitive network approaches the sensing of the environment and learning the state is the preliminary step to inference. Network monitoring has always been the simplest way for network operators to understand the current behavior of a network from heterogeneous data related to the various subsystems at stake. When moving to softwarized or cloud-like environments, monitoring is inherently supported by the management and orchestration which normally expose several performance measurements metrics regarding the different managed resources, like computing units, network, and storage. In an ETSI NFV compliant architecture, the VIM should collect and aggregate metrics and events from physical and virtual resources and communicate these metrics to the Orchestrator. Considering for example OpenStack, most of the VIM monitoring functionalities can be realized leveraging OpenStack's Ceilometer module which reliably collects measurements of the utilization of physical and virtual resources [74]. In addition to computing resources, the network assets of the NFVI (physical

and virtual switches and routers) need to be monitored as well [75]. Generally, networking in NFVI is assumed to be based on SDN; in this case, metrics can be retrieved directly either from the SDN elements themselves or by the SDN controller. For example, Prometheus [76], one of the most commonly used monitoring framework, features an OpenFlow metrics collector while in the OpenDaylight controller the Statistics API exposed metrics for the managed switches. Finally, a comprehensive snapshot of the system state requires service metrics as well: aiming to maintain an acceptable level of service, its quality must be perceived through some performance indicators. Depending on the type of the service, telco operators define a set of Key Performance Indicators (KPI) and a threshold on each KPI above which the quality of the service is considered degraded. In general, the continuous collection of metrics results in the so-called Time-Series (TS), which is defined as a sequence taken at successive equally spaced points in time. When the TS has more than one time-dependent variable, it is called Multivariate Time-Series (MTS).

TS analysis is a very broad topic that encompass several statistic and machine learning techniques appropriate for different purposes. For example, one of the most known TS parametric forecasting technique is the Autoregressive Integrated Moving Average (ARIMA) models, which are widely recognized as an accepted framework to build traffic forecast model [77]. These univariate models are used to better understand a single stationary TS and to predict future data points of variables. Extensions of these classes to deal with vector-valued data are available under the heading of multivariate time-series models and sometimes the preceding acronyms are extended by including an initial "V" for "vector". Parametric approaches can achieve good performance when traffic shows regular variations, but the forecast error is obvious when the traffic shows irregular variations. Non-parametric regression [78] and machine learning approaches are usually used for irregular time series.

No matter what source of knowledge the closed-loop leverages to sense the environment, the obtained TS needs to be studied to extract knowledge about the system state. The notion of system 'state' can be declined in different ways according to the addressed problem. In our dissertation, we are interested in a closed cognitive loop that aims to manage the resiliency of a softwarized network to reconfigure the system in case of deviations from the nominal working conditions. Consequently, the knowledge that we need to extract from the monitoring metrics is two-folded: first, the loop needs to build up a representation of the nominal working conditions; then, it needs to be able to detect a deviation from the nominal state. In the state of the art, the problem of finding patterns in data

that do not conform to expected behavior is commonly addressed as *Anomaly Detection* (AD). Ad is at the basis of *reactive diagnosis*, which aims at understanding which fault caused the given failure observed at present time and only act once the service failure has occurred. Conversely, *proactive diagnosis* mechanisms are in charge of identifying faults and errors in order to predict and avoid any future service failure. For example, in [79] Bayes networks are exploited to learn about old failures so that future subtle changes can be detected before the actual failure occurs. In the following section, we introduce state of the art techniques for network baseline creation and AD.

3.2.1.2 Network baseline and anomaly detection for telecommunication networks

Before a system can detect anomalies and make adjustments necessary to restore itself to the normal working conditions, the nominal state has to be identified. For very complex and multi-layered systems, not only it is challenging to model nominal conditions for each component but also the standards of what constitutes acceptable behavior in a system may vary both with time and from one user to another [80]. When the system is coupled with a monitoring solution, one of the biggest challenges is the establishment of a nominal baseline range for each/group of monitored metrics. According to the classic approach, the upper and lower baseline limits for a metric are chosen by the network administrator through manual observation over a given time and using the observed maximum or minimum values. As networks become more complex and dynamic, manual observations and re-calibrations of baselines are not feasible. For this reason, some network baselines are either only determined at network element installation or set using vendor default values, and are not changed throughout the lifetime of the element leading to not up-to-date settings [81]. Another well-known approach consists of the application of a set of association rules and frequent episode patterns to classify events as an anomaly or normal. While rules tend to be intuitive, they are difficult to maintain, and in some cases, are inadequate to represent many types of anomalies, as is the case for softwarized networks due to the immense variety of faults and threatens. Inductive rule generation algorithms have been proposed to overcome this limitation, as for example genetic algorithms [82].

Statistical theory outcomes are also widely used to detect anomalies. For example, in [83] the chi-square (χ^2) test statistic value is used as a distance measure to detect anomalies: when an observation chi-square value is greater than a fixed threshold, the observation is tagged as an anomaly. In [84] multivariate correlation analysis (MCA) is used to extract the geometrical correlations between a

baseline network traffic profile and generic features. In [85] authors present the Statistical Packet Anomaly Detection Engine (SPADE) as is a statistical anomaly detection system. A simple frequency-based approach is used to calculate the 'anomaly score' of a packet: the fewer times a given packet was seen, the higher was its anomaly score. Once the anomaly score crossed a threshold, the packets were forwarded to a correlation engine that was designed to detect port scans. [86] uses Markov models to characterize the "normal" behavior of the sensor network. In particular, a series of Markov models are developed and for each model, an anomaly-free probability law is estimated from past traces. Then, recent observations are studied with the Large deviations theory [87] to understand if the empirical measure takes very unlikely values.

3.2.1.3 Machine Learning methods

With the growth in availability and in the amount of monitoring data that characterize the latest and future networks, machine learning (ML) started to be applied to anomaly detection as well. Even though most of the ML AD techniques first profile normal instances and then identifies anomalies as instances that do not conform, Liu et al. [88] proposed the Isolation Forest (IF) method that explicitly isolates anomalies using binary trees. They observed that anomalous instances in a dataset are easier to separate from the rest of the sample compared to normal points. In order to isolate a data point, the algorithm recursively generates partitions on the sample by randomly selecting a dimension and a split value between its minimum and maximum values. From a mathematical point of view, recursive partitioning can be represented by a tree structure named Isolation Tree. The number of partitions required to isolate a point can be interpreted as the length of the path, within the tree, to reach a terminating node starting from the root. Consequently, the anomalous points are those with the smaller path length (easier to isolate, hence). The algorithm has a linear time complexity with a low constant and a low memory requirement, which works well in high dimensional problems that have a large number of irrelevant attributes, and in situations where the training set does not contain any anomalies. Authors in [89] leverage IF and Spark parallel computing capabilities to construct multiple trees simultaneously aiming to detect anomalies on network traffic traces.

When dealing with high dimensional data, one of the most widely used approaches is to project the data into a lower dimensionality sub-space, spot in this sub-space spontaneous clusters of data and then tag as anomaly those data which fall apart from the clusters. Principal Component Analysis (PCA) and

K-Means are the most used algorithms respectively for dimensionality reduction and clustering [90, 91]. For example, authors in [92] trained a k-mean algorithm on unlabeled flow records to cluster normal traffic. Then the distance from clusters centroids is used to affect to samples an anomaly score. Similarly, authors in [93] develop a two-stage anomaly detection algorithm based on feature selection and Density Peak-Based Clustering to handle large-scale, high dimensional, and unlabeled network data. In [94] instead, first PCA is applied to the KDD99 dataset for network IDS, and then a Support Vector Machine (SVM) is used to classify anomalous and nominal samples. However, clustering-based algorithms showed to be sub-optimal, mainly for a high false-positive rate [95]. Furthermore, PCA is recognized to fail in capturing temporal correlation [96] and in analyzing non-linear correlated metrics. Even though several non-linear approaches were proposed in the literature, it is broadly recognized that Deep Neural Networks (DNN) are very flexible and they can introduce a theoretically infinite level of non-linearities by using non-linear activation functions [97].

The main reason for their great success comes from their proven ability to approximate a very wide range of continuous functions, that is almost all the functions of interest [98]. Furthermore, early work showed that standard NN cannot be a universal classifier, but that a DNN with at least one hidden layer of unbounded width can [99]. DNNs are used in a wide range of problems (speech/text recognition, natural language processing, recommendation systems, mobile advertising, fraud detection, etc...) and many variants on the standard architecture there exist for each problem. In anomaly detection, one of the most widely used architecture is the Deep Autoencoder (AE) one [100, 101, 102].

When training AEs with anomaly-free data, anomalous samples projected into the latent space look significantly different from nominal samples; as a result, their reconstruction error is greater when compared to nominal samples. Leveraging that, authors in [101] demonstrated that AEs clearly outperform PCA in terms of accuracy and computation time. They also showed that autoencoders learn the normal state properly in the hidden layers and that they activate differently with anomalous input. Stacking several encoder and decoder layers to build the so-called Deep AE (DAE) is commonly proposed in the literature to extract more general properties of data than a single layer [103]. Even though DAE are employed to compress high dimensional vectors, several studies showed that it can struggle to efficiently learn with high-dimensional input vectors [104, 105, 106]

Standard AE architecture is also used in combination with other techniques: in [107] authors propose a model consists of two stages, an Unsupervised Feature Learning (UFL) stage using sparse

AEs, followed by a classification stage that uses the learned features with soft-max regression (SMR). The 2-class classification achieves a higher accuracy of 88.39% compared to 78.06% of SMR, and outperforms SMR with respect to recall and F-measure. However, SMR outperforms STL in precision. AEs are considered as an auto-supervised neural network, as the target value used to train is the input itself, so no labels are required in the training phase. For this reason, AEs are particularly suitable for anomaly detection in softwarized network infrastructures: labeling anomalies for such an environment is a time-consuming and error-prone task, due to the great extent of faults and threats that can affect NFV environments [108, 109].

It is worth mentioning that Variational AEs (VAEs) have emerged as another popular unsupervised learning approach for heterogeneous input distributions [110]. VAEs are generative models, which aim is to learn how the data is generated: they use a variational approach for latent representation learning, which results in an additional loss component and a specific estimator for the training algorithm, called the Stochastic Gradient Variational Bayes (SGVB) estimator [111]. It assumes that the data is generated by a directed graphical model and that the encoder is learning an approximation to the posterior distribution [112]

As already said, AE is just a specific composition of two specular groups of NN; thereby, several types of NN can be used according to the type of data the AE works on. We already pointed out that the monitoring component of a cognition control loop generally produces a multivariate time series. Consequently, when designing an AE-based anomaly detection system for state assessment of softwarized networks, it is important that the NNs composing the AEs catch both the time dynamics of each variable, and the cross-dependencies among variables, to effectively grasp knowledge from the input data. When analyzing MTS, in the literature, Recurrent neural networks (RNNs) are generally widely recognized as a suitable method to capture both the temporal and spatial evolution of the TS [113]. In particular, the Long-Short-Term Memory (LSTM) RNN showed the ability to learn long term correlations in complex multivariate data sequences [114], thus they have been used in a wide range of applications including speech/handwriting recognition [115, 116], forecasts [117] and time-series prediction/anomaly detection [118]. In network traffic and load forecasting, LSTMs demonstrated to outperform non-ML and other DNN approaches [119, 120, 121, 122]. In [123] authors propose a mechanism to scale 5G core resources by anticipating traffic load changes through LSTM and deep neural networks forecasting. They show that LSTM-based anomaly detection can be more

accurate, thanks to its ability to store data patterns without degradation over time. Authors in [114] propose a stacked LSTM architecture to detect anomalies within time series data by evaluating the deviation of predicted outputs based on variance analysis. In [124], a compound architecture is presented. Here, the LSTM network predicts regular system dynamics and a support vector machine is applied as a classifier for anomalies to realize an adaptable and self-learning detection mechanism.

3.2.1.4 Anomaly diagnosis and characterization

The advantage of learning the nominal working condition is that any deviations from it, even unforeseen ones, can be easily detected. On the other hand, not all deviations imply a degradation, some deterioration could be more severe than others and each deviation require a different remediation action. Furthermore, it is important to define the criteria for a "healthy"system and identify thresholds indicative of the need to initiate a healing process. In general, the distinction between a healthy and a faulty system is indistinct and fuzzy, as the transition between these two states is gradual. It is important that a self-healing system discern this progressive decline and that it identifies a definite threshold to initiate remediation. Therefore, a diagnosis function is required to characterize the detected anomalies and further analyze the discovered symptoms.

To this end, authors in [125] propose to use a set of KPIs to classify anomalies according to specific combinations of their values, the so-called *anomaly patterns*. Then, the observed anomaly pattern is compared against the already analyzed and labeled anomalies stored in the diagnosis knowledge-base. The label specifies the *type* of the anomaly for each KPI pattern. The closest matching labeled anomaly pattern or patterns are found and given as the most likely automated diagnosis. Labeling of anomaly patterns is performed by a human expert. In [126] authors pointed out that for each detected anomaly, the severity level of the consequences should be diagnosed first: critical anomalies in fact should be quickly handled, in order to avoid serious outages of the service. In particular, they estimate that the magnitude of the deviation from the nominal state is directly proportional to the reaction speed. They also propose the use of Naive Bayes Classifier to infer the possible consequences of the deviations and maps each classifier outputs to a *prescription*, that is the remediation action. The classifier is re-trained after each action execution to improve its accuracy. Ciocarlie et. al. [127] propose a framework consists of an anomaly detector and a diagnosis component for telecommunication cells. The former monitors a group of cells using topic modeling. The diagnosis component, in turn,

uses Markov Logic Networks (MLNs) to generate probabilistic rules that distinguish between different causes. Likewise, [128] uses PCA for dimension reduction and kernel-based semi-supervised fuzzy clustering with an adaptive kernel parameter for proactive network anomaly detection.

Zhang et. al. [129] propose a Multi-Scale Convolutional Recurrent Encoder-Decoder (MSCRED), to perform anomaly detection and diagnosis in multivariate time series data. Specifically, MSCRED first constructs multi-scale (resolution) signature matrices to characterize multiple levels of the system statuses in different time steps. Subsequently, given the signature matrices, a convolutional encoder is employed to encode the inter-sensor (time series) correlations and an attention-based Convolutional Long-Short Term Memory (ConvLSTM) network is developed to capture the temporal patterns. Finally, based upon the feature maps which encode the inter-sensor correlations and temporal information, a convolutional decoder is used to reconstruct the input signature matrices and the residual signature matrices are further utilized to detect and diagnose anomalies.

3.2.1.5 Fault mitigation and automatic reconfiguration

A cognitive loop allows to cover from the sensing of the network state to the self-healing (SH) phase, that is the application of a set of actions (policies) to recover from a malfunction. In its simplest form, action selection for SH is a purely reactive strategy that implies a set of alternative actions corresponding to distinct classes from which the agent can choose based on descriptions of the current system state. This is the case of [130] where repair strategies form a sequence of preconditioned *tactics* that execute an appropriate repair script. Similarly, Valetto et al. [131] propose the utilization of high-level repair action plans coupled with a feedback control loop. Other works instead, like [132], formulate the anomaly compensation problem as a network utility maximization problem. A support vector data description approach (SVDD) is used to detect cell outages in ultra-dense small cell networks; then a reactive distributed compensation algorithm redistributes resources guaranteeing load balancing.

Furthermore, some works advocate that due to the extreme complexity of networking systems, fault mitigation should also encompass the learning from a human system administrator. In this case, the interaction could be either direct like in *adaptive interface* [133] or indirect, that is by means of historical data recorded during the human intervention, like in *behavioral cloning* [134].

Nonetheless, one key idea of cognitive control loops is that the reconfiguration policies would

be learned over time, adapting to problems evolution, tune its operation accordingly and generally increase its reliability and robustness. The learning process should aim at an abstract goal dictated by the business or user technical requirements such as maintaining a certain Quality of Service (QoS) to fulfill a Service Level Agreement (SLA). As highlighted in the chapter introduction, this suggests the incorporation of concepts and methods from machine learning [135]. One of the ML techniques that seem to be particularly suitable is Reinforcement Learning (RL) [136], where an agent carries out actions in the environment and receives some reward signal that indicates the desirability of the resulting states. Because many steps may be necessary before the agent reaches a desirable state, the reward associated to each action leading to a new state is '*cumulated* 'in the sense that it is computed to take into consideration for the future actions and associated rewards. Accordingly, an RL agent is designed to maximize the so-called cumulative reward [137], sacrificing the immediate gains for long-term rewards. In its most commonly studied form, RL aims to learn effective management policies in the absence of explicit system models, with little or no domain-specific initial knowledge. This type of approach attempts to circumvent the unfeasibility of modeling very complex systems, like softwarized networks.

In the context of SDN/NFV systems, reinforcement learning has been used to solve different problems, like VNF placement, optimal chaining of VNFs, VNFs scaling and resource management, routing and anomaly detection, among the others.

For what concern SFC [138] proposes a deep RL approach for VNF placement problem in SDN/NFV-enabled networks according to forecasted SFC Requests (SFCr) in a future time interval. In their proposal, the agent has to put in place an SFC request according to the current state modeled through the available resource ratios of links, nodes and VNF instances, in each part of the network. The reward depends on the VNF placement cost, the penalty of rejecting the SFC requests and the VNFI running cost. Similarly, [139]proposes a Q-learning algorithm for the service placement on SDN switches to minimize the service costs for end-users. The network is modeled with a graph, where the nodes are SDN switches connected via undirected links. A weight is associated with each link to capture service access cost and a penalty cost of service access between the SDN controller and switches is associated to each switch. Q-learning agent state is modeled as the currently available services, the actions correspond to the add/removal of a new service and the reward depends on both the service cost and its penalty.

Concerning resource management, in [140] authors propose a reinforcement learning algorithm for resource management for cloud-based applications. They address the problem of Virtual Machines (VMs) horizontal scaling when an application processes a single type of user request. The proposed approach is using a fuzzy Q-learning method to deal with a continuous workload. The agent has to decide which scaling action to perform from the actual state modeled as fuzzified workload ("low", "medium", "high") and the number of machines. The reward is computed as a fixed revenue minus the cost for using capacity and the penalty for violating the service level agreement on response time. [141] addresses instead the scaling of VNFs in the virtual Evolved Packet Core (vEPC). Authors propose a mechanism to scale virtualized MME to respond to user service request variation. They combine Q-Learning with Gaussian Processes-based system models aiming at maintaining the Mean Response Time (MRT) of the EPC control plane less than a threshold. The agent state is a vector containing the current number of instances and a performance indicator. The action corresponds to horizontal scaling and the reward depends on the performance target (e.g., MRT less than 1 ms) and the utilization factor that is the proportion of time that a server is busy on average.

DQN has been instead used to o target Quality-of-Service (QoS)-aware routing. In [142] a DQN agent is trained to optimize the flow routing in a network where the state is represented through the Traffic Matrix (TM) of the current network load, the actions consist in choosing the path to route the new incoming flow and the reward is related to the QoS parameters that are latency, rate and packet Loss.

Concerning anomaly detection, [143] tackles the stealthy DoS attacks in SDN-based networks. This type of attack is characterized by periodic requests with a low rate with respect to classical attacks. In their proposal, a Q-learning is used to select the optimal features for an ML classification algorithm. The state is a vector composed by the precision, the recall, the F-score, the accuracy, and the false alarm rate of the classifier. The actions are all the possible combinations of feasible features (e.g., average packets per flow, average packet size per flow, packet change ratio and flow change ratio) and possible AI/ML algorithms for traffic classification (e.g., Support Vector Machine, Random forest). The reward depends on the value of the components of the state vector. Similarly, [144] widens the analysis of the problem to various types of anomalies, having as objective to promote resilience in SDN. In particular, the authors consider two profiles: (i) load balancing and server replication. For each profile, that deals with a specific anomaly there is a set of possible actions that can be chosen to

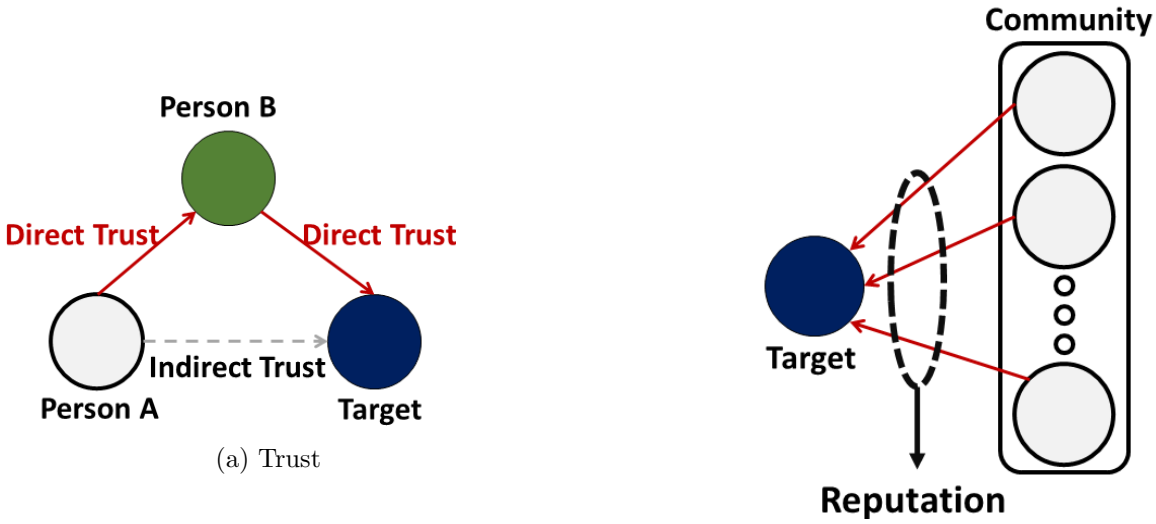
mitigate the anomaly. At each time stepm only one profile is chosen and a specific SARSA algorithm is considered. SARSA is a modified version of the Q-learning where the main function for updating the Q-value depends on the current state of the agent S_1, the action the agent chooses a_1, the reward r the agent gets for choosing this action, the state s_2 that the agent enters after taking that action, and finally the next action a_2 the agent chooses in its new state. In the load balancing profile the possible overflow of the switch input queue is tackled. Here, the state is the load of the route. The actions are to change the route, use the shortest path or modify the route. The reward is negative value if there are links with high traffic, positive otherwise. In the server application profile, the SARSA algorithm instead deals with the vertical scaling problem due to overloading. Here the state is a combination of (i) the traffic in the access link between server and switch, (ii) the set of servers or replicated VNFs, and (iii) the paths to the servers or replicas. The reward associated with scaling actions is negative if there are links with high traffic, positive otherwise.

3.3 Reputation assessment modeling

Trust and Reputation Management (TRM) systems are generally used to reduce the impact of misbehaving or faulty entities in multi-agent systems. The notions of trust and reputation originate from the social sciences which study the dynamics of human societies

As depicted in figure 3.13b reputation is an abstract property defined as public knowledge and represents the collective opinion of members of a community. It is a function of the *trust*, which is instead the perception and entity (trustor) has on another entity (trustee). It is based on past collaboration and it is specific to a particular task and is is not reciprocal. Trust is useful only in an environment characterized by uncertainty and where the participants need to depend on each other to achieve their goals [145]. Trust maybe also indirect, when an trustor base its trust assessment on a third entity witness.

In the state of the art, TRM systems are mainly used to assess the risk of a given interaction between two elements (a trustor and a trustee) within a system and manage security aspects of computer systems. For instance, in [146] WSN (Wireless Sensor Networks) network nodes use reputation to decide where to send their packets to optimize routing. When applying TRM in this context, the nodes are assumed to be noncooperative, i.e., a node is not willing to relay a message/packet of



(a) Trust

(b) Reputation

Figure 3.13: Trust and reputation model

another node unless it can derive some potential benefit. Similarly, [147] presents a framework of web services where a reputation system is incorporated for tracking and predicting users satisfaction. Presented reputation systems operate in an environment of composite services that integrate client and server-side. The approach aims at maximizing user experience for specific customer profiles when the service and network resources are shared.

TRM systems are also adopted in wireless communication systems which include a framework containing both a trust and reputation evaluation model and the protocol for nodes to interact. The protocol includes means for rewarding good behavior and penalizing bad (selfish or malicious) behavior. Trustworthiness is viewed as social capital which can be earned or lost. For instance, the SORI scheme [148] seeks to use the reputation information of nodes in a MANET as a punitive measure to deter selfish behavior during the collaborative forwarding of transmitted packets. The reputation of each node is computed as $r = RF_H(X)/HF_N(X)$, where $RF_H(X)$ is the total number of packets node N has transmitted to node X for forwarding and $HF_N(X)$ is the total number of packets that have been forwarded by node X (including packets from other nodes) and noticed by node N. The nodes probabilistically refuse to relay packets from other nodes with probability $1 - r$. In [149] authors propose a model for calculating trust and reputation for blockchain-based online payment systems which have a characteristic of immutability by preventing data manipulation. The model normalizes user evaluations based on each user's personal evaluation criteria that change over time. In addition,

the model derives reputation of, and trust between, users by applying psychological factors.

Integration of such TRM in softwarized networks is a new topic, but there are some works as the one of Isong et al in [150] where reputation is used to certify how trustable are SDN applications controlling the controller to establish real-time and on demand connexions. TRM models can be also used for automating the decisions regarding the choice made by a user of a virtual network over multiple virtual networks claiming to cater for a given level of service to that user [151]. Betge-Brezetz et al. in [152] propose a trust-oriented controller proxy that intermediates between the controllers and the data plane by making sure the flows sent by different controllers are correct.

3.4 Machine learning techniques

In this section we outline the basic properties of the machine learning algorithms used along this thesis work. We will not fully detail the mathematical background, giving just some elements to understand the intuition about the strength of each technique.

3.4.1 Machine learning and Deep Learning

Figure 3.14 depicts the architecture of the basic component of most used machine learning techniques , the artificial neuron [153].

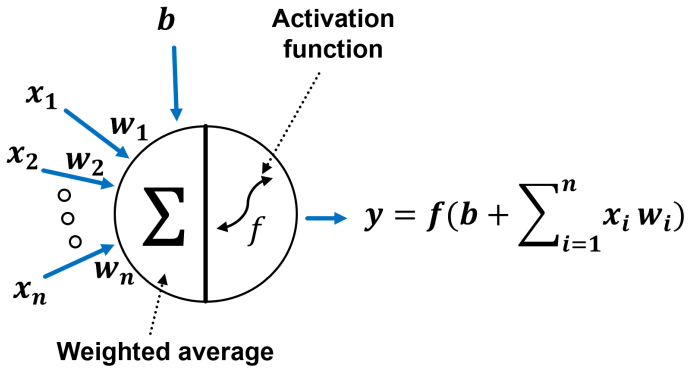


Figure 3.14: Artificial Neuron

A neuron is a mathematical function conceived as a model of biological neurons. It takes n inputs $(x_1, x_2, ..., x_n)$, multiplies each with a specific weight $(w_1, w_2, ..., w_n)$, adds a bias (b) and then passes the result to a nonlinear function called the *activation function* to produce an output. A neural network combines multiple neurons by stacking them vertically/horizontally to create a

network of neurons. Theoretically, there is no restriction on the selected activation function; however when the activation function is non-linear, then a two-layer neural network can be proven to be a universal function approximator [154]. Furthermore, in order to enable the learning process through the Gradient Descent (GD) algorithm, the activation function should be continuously differentiable. Indeed, typically a neural network model is trained using the stochastic gradient descent optimization algorithm and weights are updated using the backpropagation of the error algorithm. Here the gradient refers to an error gradient: the NN with a given set of weights is used to make predictions and the error for those predictions is calculated. The gradient descent algorithm seeks to change the weights of the NN so that the next evaluation reduces the error; to do that the optimization algorithm is navigating down the gradient (or slope) of error.

Horizontally stacking several layers of NN it is possible to obtain an arbitrarily complex NN called Deep NN.

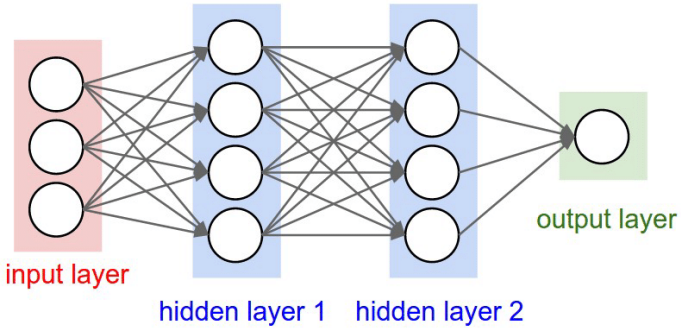


Figure 3.15: Fully connected Deep Neural Network

As shown in figure 3.15, the first layer of DNN is called the input layer, and the number of nodes depends on the number of features composing the analyzed dataset. The final layer of the neural network is called the output layer, and the number of used neurons depends on the problem: for regression and binary classification tasks, a single node is used while for multi-class problems, an output node is used for each class. All the other layers are called hidden.

3.4.2 AutoEncoders

A standard AE is a multi-layer Neural Network (NN), composed of two blocks, an encoder and a decoder. The typical architecture of an AE is shown in Fig. 3.16.

The encoder reduces the F dimensions of the input X to a 'latent-space' composed of $s < F$

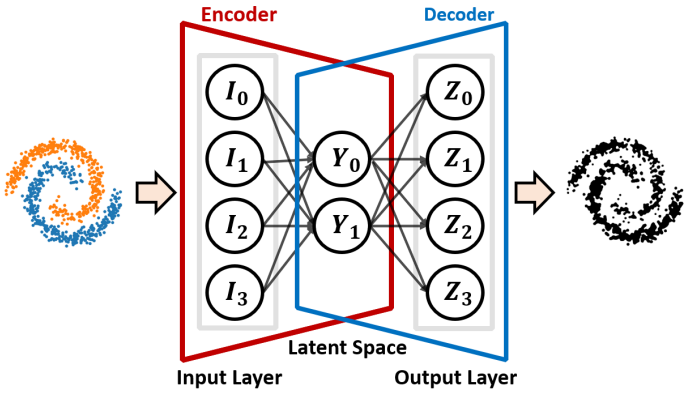


Figure 3.16: Autoencoder architecture

dimensions, while the decoder takes those s dimensions back to reconstruct the input. The autoencoder is trained to learn to reproduce the input vector X of F features $\in \mathbb{R}^F$ by optimization of:

$$\underset{f,g}{\text{minimize}} |I - g \circ f(I)| \tag{3.1}$$

where $f\colon I \in \mathbb{R}^F \mapsto Y \in \mathbb{R}^s$ with $s < F$ is the function representing the encoder, and $g\colon Y \in \mathbb{R}^s \mapsto Z \in \mathbb{R}^F$ is the function representing the decoder. During the learning phase, weights and biases are tuned to minimize the reconstruction error on I.

3.4.3 Long-Short-Term Memory

Unlike Feed Forward (FF) NN, where each element is processed independently from the others, RNNs apply a recurrent relation at every time step to process a sequence in order to take into account past inputs, like a sort of memory. Figure 3.17 outlines both FFNN and RNN architecture: at each time-step t the RNN outputs a value $\hat{y}_t$ which depends on both the actual input x_t and the so-called cell state h_t. The latter is obtained through a non linear function f_W, generally known as *transfer function*, which is parameterized by a a weight Matrix W and a set of biases and is applied to both the current input and the previous step cell state h_{t-1} Consequently, $h_t = f_W(h_{t-1}, x_t)$ where, for instance, f_W could be a $tanh$ function giving $h_t = tanh(W_{hh}h_{t-1}+, W_{xh}x_t + b_{hh} + b_{xh})$. Note that as we have two inputs, we also have one couple of weights and biases for the input and another for the old hidden cell state; furthermore, it is worth noticing that for each time-step t the set of weights and biases is the same, which is crucial for sequence modeling.

The output vector $\hat{y}_t$ is then obtained as a transformed version of the internal state: $\hat{y}_t = W_{hy}h_t$.

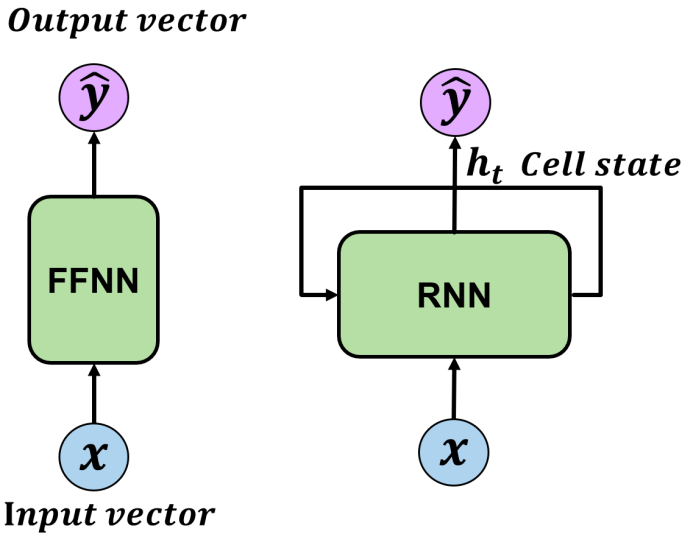


Figure 3.17: Feed Forward and Recurrent Neural Network architectures

Despite being able to deal with sequential data, RNNs suffer from the vanish gradient problem [155], which prevents long-term relations to be learned. The problem arises from a chain of subsequent small valued matrix multiplications that takes place during the back-propagation of the error which makes the gradient of long-term past samples smaller and smaller, which in turn makes the network learning mostly on fresh samples. As a solution to that, authors in [156] propose the use of the so-called Long-Short-Term-Memory (LSTM) RNN that enforces constant error flow through the internal states of special 'memory cells' units by employing the *gates*. Unlike standard RNN, LSTMs feature two cell state: an internal cell state (c_t) and the output cell state (h_t). It is thanks to this separation of internal cell state and outputted cell state that the gradient can backpropagate without any risk of vanishment unlike in RNN. Another key takeaway of LSTM is the filtering information capability obtained through gates: they are composed of a Sigmoid NN layer which caps the input information between 0 and 1, which respectively means forgot everything and remember everything.

Figure 3.18 depicts the four steps through which an LSTM cell process the information:

1 **Forget**: this steps computes what to forget about the irrelevant history of previous layer state h_{t-1} and the current input x_t. This decision is made through the sigmoid layer which is characterized by a weights matrix W_f and a set of biases b_f. The produced output is then $f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$ which is a value between 0 and 1 for each dimension of the cell state. A 1 represents '*completely keep*' while a 0 '*completely forgot*';

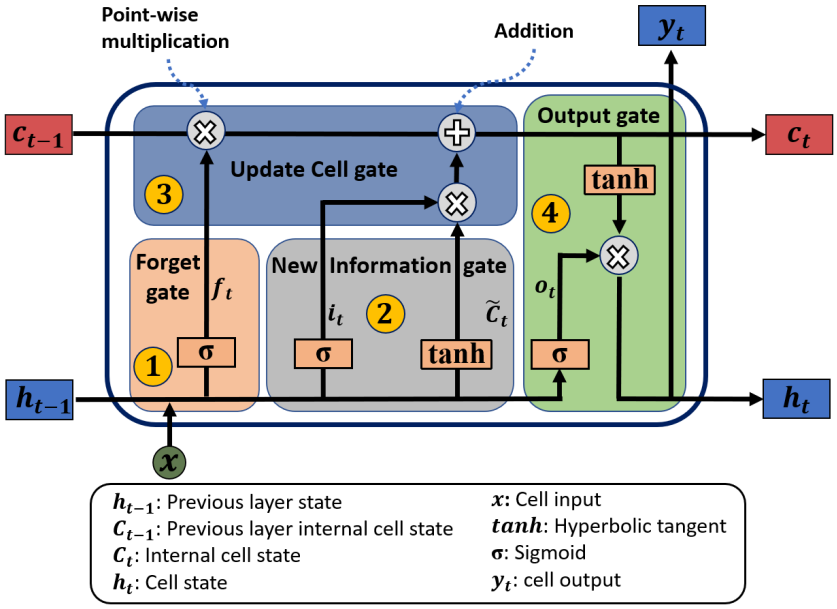


Figure 3.18: Long Short Term Memory Recurrent Neural Network architecture

2 **New Information**: here a computation is performed to identify relevant parts of new information. First the input (actual input and previous layer cell state) is gated to identify what values we should update $i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$. Then, the *tanh* layer generates a new vector of candidates values ($\tilde{C}_t$) that could be added to the state $\tilde{C}_t = tanh(W_C \cdot [h_{t-1}, x_t]) + b_C$;

3 **Update**: this step takes 'forget' and 'new information' output to selectively update cell state. It multiplies the old cell state c_{t-1} with the forget gate output f_t to forget what we decided to forget. Then, it adds the new set of candidates values scaled by how much we decided to update each state value (i_t). Then new internal cell state becomes then $C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$;

4 **Output**: here we finally generate the output deciding what information encoded in the internal cell state is sent as output, which is also the input to the next time-step cell. First o_t is computed through a sigmoid layer applied to the old cell state and the actual input $o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$. Then it is used to decided which part of the updated cell internal state we are going to output $h_t = y_t = o_t * tanh(C_t)$. The tanh layer pushes the values of the internal cell state to be between -1 and 1.

The key implementation design choice which avoid the vanishing gradient problem is the separation of cell internal state from the outputted state: the back-propagation through time algorithm is com-

puted on the cell internal state and it dose not involve repeated small valued matrix multiplications. Doing that, the gradient of samples far away in the past is still appreciable and it is consequently relevant in deciding the optimization direction in the gradient descent learning algorithm.

3.4.4 Reinforcement Learning

Traditional machine learning approaches are either characterized as *supervised* or *unsupervised*, whether the input data are labeled or not. While supervised learning models tend to be more accurate than unsupervised ones, they require upfront human intervention to label the data appropriately. Unsupervised learning models, in contrast, work on their own to discover the inherent structure of unlabeled data. Even if widely used to solve plenty of problems, both categories fail when the problem requires learning from the interaction between an agent and the environment. Indeed, in interactive problems it is often impractical to obtain examples of desired behavior which are both correct and representative of all the situations in which the agent has to act. Furthermore, decisions may happen rapidly and time constraints may be involved. The system may evolve so quickly that often fixed learning rules designed to payoff-optimization on the interactive system might be quickly out of date [137]. On the contrary, the agent has to experience himself with the environment in order to learn from its own experience so that, at some point, it can overcome unseen situations as well. The learning process is guided by the *reward signal*, which quantifies the actions *goodness* to approach the agent to its *goal* state and which the agents always try to maximize. Thereby, one of the challenges that arise in reinforcement learning, and not in other kinds of learning, is the trade-off between exploration and exploitation: in order to gather enough feedback from the environment, the agent should explore, with a certain probability, new actions for winch the reward is unknown.

As depicted in figure 3.19, basic reinforcement is generally modeled as a Markov decision process (MDP), where at each time step t, the agent receives some representation of the environment's state, $S_t \in \mathcal{S}$ where $\mathcal{S}$ is the set of possible states, and on that basis selects an action, $A_t \in A(S_t)$, where $A(S_t)$ is the set of actions available in state S_t. One time step later, as a consequence of its action, the agent receives a numerical reward, $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$, and finds itself in a new state, S_{t+1} with a probability P . The agent goal is to maximize the sum of the rewards $\sum_{i=0}^{N} R_i$ it will obtain so that to reach the goal state through the optimal path.

At each time step, the agent implements a mapping from states to probabilities of selecting each

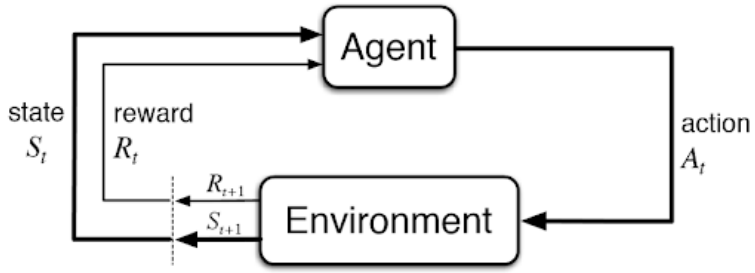


Figure 3.19: Reinforcement Learning framework

possible action. This mapping is called the agent's policy and is denoted π_t, where $\pi_t(a|s)$ is the probability that $A_t = a$ if $S_t = s$. The way in which the agent changes its policy as a result of its experience depends on the specific RL algorithm used. In general, algorithms are classified as model-based or model-free.

Model-free directly learn both the policy and the value function from interactions with the real world scenario. One of the most known model-free algorithm is the Q-learning [157] which aims at maximizing the expected value of the total reward over any and all successive steps, starting from the current state. The "Q" refers to the function that the algorithm computes, which is the expected rewards for an action taken in a given state (Bellman Equation):

$$Q(S_t, a_t) = E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ...|S_t, a_t] \tag{3.2}$$

In particular, the algorithm stores in the so-called Q-Table the value for the Q-function for each action at each state. At the beginning the table is initialized with tentative values (for example all 0). Then, according to a specific policy, an action a_t is selected, executed and the result is observed. According to the received reward, the Q-value $Q(S_t, a_t)$ is updated following the formula:

$$Q_{t+1}(S_t, a_t) = Q(S_t, a_t) + \alpha[R_t + \gamma \max_{a_t}\{Q(S_{t+1}, a_t)\} - Q(S_t, a_t)] \tag{3.3}$$

where α is a learning rate which regulates at what extent newly acquired information overrides old information, γ is the discount factor which determines the importance of future rewards and $\max_{a_t}\{Q(S_{t+1}, a_t)\}$ is an estimate of the optimal future value. Concerning the action selection policy, there exists several types but the most used in the Q-learning is the ϵ-greedy strategy. It consists into exploit the knowledge, selecting, with a high probability of $1 - \epsilon$, the "best" action, i.e. the one with the highest Q-value and with a small probability ϵ to uniformly select one of the other action.

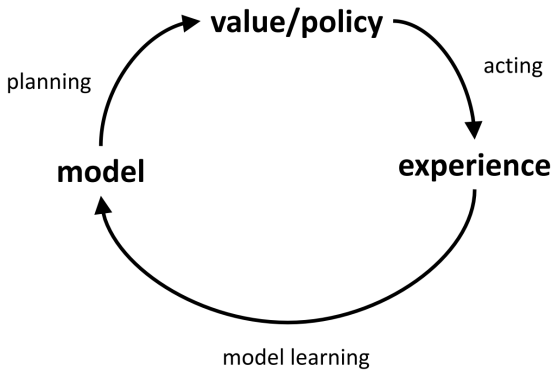


Figure 3.20: Model-based Reinforcement Learning

A popular variant of the classic Q-learning algorithm is the Deep-Q-Learning (DQL) which replaces the Q-table with a DNN. The only difference from the architecture point of view is that Q-learning algorithm takes as input the couple state-action while DQN takes as input only the state; thus the output is the Q-value for each action. However, both DQN and Q-learning suffer of overestimation as they use the same values both to select and to evaluate an action. To prevent this [158] propose the Double Deep q-Networks (DDQN) which levereage two set of weights θ and θ', one to determine the greedy policy, and the other to determine its value. This reduces the correlations between the target and estimated Q-values, stabilizing the algorithm.

Unlike model-free algorithms, model-based learn a model from experience and plan the policy and/or the *goodness* of a state (value function) from the model. The agent can use a single prediction from the model of next reward and next state (a sample), or it can ask the model for the expected next reward, or the full distribution of next states and next rewards. These predictions can be provided entirely outside of the learning agent or they can be learned by the agent, in which case they will be approximate. In particular, given a MDP $M = [S, A, P, R]$, it is generally assumed that the states space S and the set of actions $A(S_t)$ for each state S_t are known; a model M' of the MDP M is then $[S, A, P', R']$. Learning P' is modeled as a density estimation problem while learning R' is a regression problem that in this case can be handled as supervised-learning problems. Once the model is built, it is used to generate samples, that is used as the real world scenario for a model-free RL algorithm which implies an update of the value functions and policies from samples. Figure 3.20 summarizes those steps.

Chapter 4

Virtualized System Anomaly Detection and Characterization

Content

In this chapter we propose an unsupervised machine-learning data-driven approach based on Long-Short-Term-Memory (LSTM) autoencoders to detect and characterize anomalies in virtualized networking services. With a radiography visualization, this approach can spot and describe deviations from nominal parameter values of any virtualized network service by means of a lightweight and iterative mean-squared reconstruction error analysis of LSTM-based autoencoders. We implement and validate the proposed methodology through experimental tests on a vIMS proof-of-concept deployed using Kubernetes. This chapters reports contents of publications [159, 160].

4.1 Introduction

Legacy and novel network services are expected to be migrated and designed to be deployed in fully virtualized environments. Starting with 5G, NFV becomes a formally required brick in the specifications, for services integrated within the infrastructure provider networks. This evolution leads to deployment of virtual resources Virtual-Machine (VM)-based, container-based and/or serverless platforms, all calling for a deep virtualization of infrastructure components. Such a network softwarization also unleashes further logical network virtualization, easing multi-layered, multi-actor and multi-access services, so as to be able to fulfill high availability, security, privacy and resilience requirements. However, the derived increased components heterogeneity makes the detection and the characterization of anomalies difficult, hence the relationship between anomaly detection and corresponding reconfiguration of the NFV stack to mitigate anomalies.

Network automation is a vibrant research area targeting the deployment of novel solutions in operational networks in the coming few years. Even though initial network automation research actually dates back up to a few decades ago, true network automation fueled by artificial intelligence (AI) and machine learning (ML) has only recently become a tangible possibility for operational services, thanks in particular to novel technologies related to Software Defined Networking (SDN) - with the specification of open configuration interfaces - and Network Functions Virtualization (NFV) - breaking the coupling between network functions and the hosting hardware.

In the past few decades, the community has addressed challenges related to how to let distributed sets of agents self-organize, automatically discover themselves the network states, and operate necessary reconfiguration of the network. This was for the focus of many research projects in the area of autonomic networks [161]. We can also cite standardization activities related to network automation, as for instance the ones related to the autonomic signaling protocols among distributed decision-making agents [162]. Nonetheless, these pioneering research activities did lack a stable reference technical architecture on top of which a decision-making framework could be developed and deployed at large scale, for instance to solve routing or resource allocation optimization problems. With the advent of network virtualization technologies, the reference building blocks for 5G infrastructures, and beyond, are today clearly specified and adopted. On the one hand, the relative maturity of NFV-SDN systems has focused the industry specification efforts on the interfaces required for network automation,

somehow meeting the expectation of former autonomic networking research, but now with an operational environment ready for their integration. The Zero-Touch Network and Service Management and Experiential Networked Intelligence groups at ETSI are addressing this need and recently produced a set of specifications [163, 164]. On the other hand, network automation platforms recently emerged, notably the Open Network Automation Platform, chosen by many operators as a reference platform for network automation [165, 166, 167]. More recently than for the core segment, the radio one is undergoing an increasing softwarization, with new platforms as the Open Radio Access Network one [168, 169]. These activities are opening the way to orchestration decisions for which there is a critical need for automation algorithms and methods to (i) determining how the state of a fully virtualized and programmable infrastructure, composed of a variety of software modules, should be modeled, (ii) inferred in runtime, and (iii) to support automated network orchestration.

In this chapter, we present the first brick in this direction and proposes a methodology to detect anomalies in the rather unidentified network state space composed of a very large number of software components. These components can be characterized by a large number of metrics, changing in number and behavior in time, that can be correlated or not to each other, depending on network conditions. This undefined and varying environment motivates us to propose an unsupervised machine learning framework for anomaly detection of NFV infrastructures. We run tests in a virtualized IP Multimedia Subsystem (IMS) architecture, the legacy framework used for voice-over-IP traffic routing and processing. Simulated call distributions and used datasets are available at [170].

4.2 Virtual IP Multimedia Subsystem (vIMS) testbed

vIMS is the virtual solution of the classical Ip Multimedia Subsystem (IMS), which is an IETF and 3GPP standard for Voice over IP (VoIP) for 4G and 5G, and an architectural framework for delivering IP multimedia services such as voice, video calling, and messaging applications. IMS multimedia services are delivered through SIP. SIP is the standard protocol for the telecommunication multimedia services signaling [171]. IMS provides efficient use of spectrum, eliminating the need to separate voice and data in two different networks while allowing the interoperability of multimedia services across operators. The virtualization of the IMS brings scalability, programmability and flexibility of services as well. Figure 4.1 depicts IMS base architecture

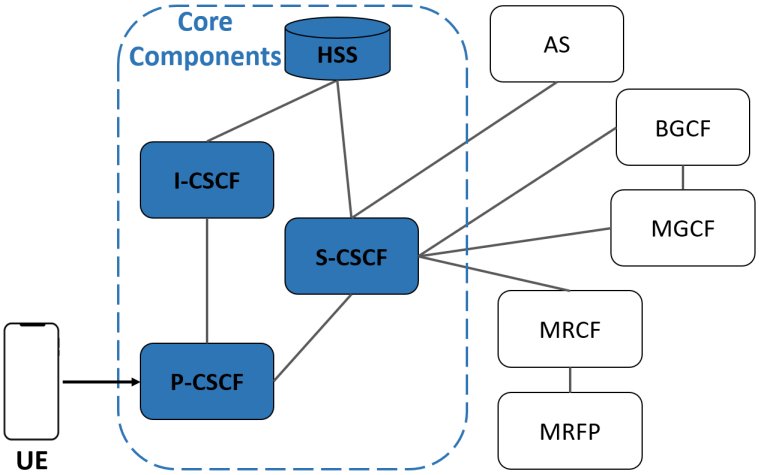


Figure 4.1: IP Multimedia Subsystem architecture

The architecture of an IMS is depicted in figure 4.1 and is composed by:

- HSS (Home Subscriber Server): database containing subscriber's profiles performing authentication and authorization;
- P-CSCF (Proxy Call Session Control Function): the SIP proxy server that is the first point of contact for the users;
- S-CSCF (Serving-CSCF): SIP server and session controller, it is the central node of the signaling plane;
- I-CSCF (Interrogating-CSCF): the SIP function located at the edge of an administrative domain; it assigns an S-CSCF to a user performing SIP registration;
- MRFs (Multimedia Resource Function): enhances multimedia application provided by the AS providing advanced video conferencing features and supporting new audio and videos CODECs for conferencing and streaming. It is decomposed in the Multimedia Resource Function Controller (MRFC) and in the Multimedia Resource Function Processor (MRFP);
- Application Server: host and execute services
- Breakout Gateway Control Function (BGCF) interrconnects the IMS with the Public Switched Telephone Network (PSTN);

- Media Gateway Control Function (MGCF): is responsible for the interworking with the PSTN converting SIP messages to ISDN User Part (ISUP) PSTN signaling.

Several open-source implementations exist in the community, most of them based on the opensource OpenIMSCore IMS [172] project. In our tests we used the opensource OpenIMSCore IMS [172] functions, deployed as separated containers managed by Kubernetes [173].

4.2.1 Platform architecture and traffic simulation

Figure 4.2 depicts vIMS pods and containers location across both physical servers composing our deployment.

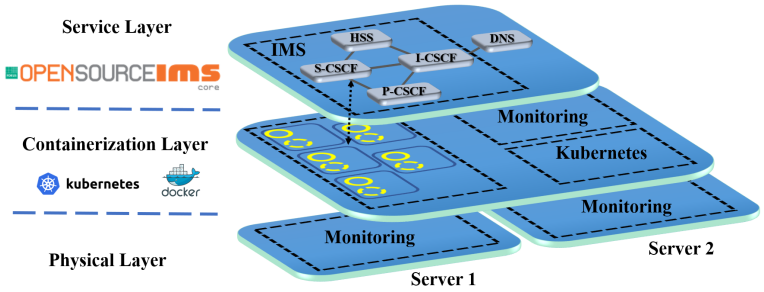


Figure 4.2: Testbed

Each physical server is equipped with an Intel (R) Xeon (R) CPU E5-2620 v4 @2.10GHz with 384 GB of RAM, connected to the same network through a 1-Gbps port physical switch. All the vIMS functions are deployed in dedicated Pods located in the server 1 (srv1), while Kubernetes core components are deployed in the server 2 (srv2). Server 2 hosts the SIPp [174], a traffic simulator used to inject SIP and RTP traffic into the platform as two pods representing the caller and the callee. In particular, we used SIPp to simulate a nominal scenario where several SIP clients get first registered to the vIMS core and then start a call according to a custom call-flow specified to SIPp through an XML scenario which details the messages and their order across a call.

To simulate realistic traffic, we used real call traffic profiles extracted from a given LAC (Location Area Code) from Orange 3G network. We injected three weeks (March 16-29, 2020) of this traffic distribution onto the vIMS containerized platform under test. As shown in figure4.3, we set the average call duration to 3 min according to [175]. Both RTP data traffic and SIP signaling traffic are transported over UDP. Moreover, the vIMS containerized platform is tailored to correctly process this

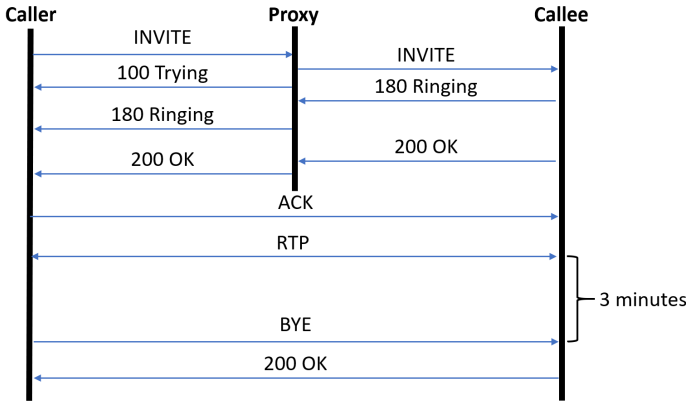


Figure 4.3: Call CallFlow

traffic load. Figure 4.4 reports mean call distribution for the first and the second week as well as an LAC distribution used for testing purpose. Call distributions and obtained datasets are available at [170]. It is worth highlighting that virtualized network services, differently than legacy transport network services, are tailored to a particular traffic, which can be isolated first and then chained through dedicated functions thanks to the possibility to program virtual switches along the network path. This is the reason why we focus on a particular traffic and related virtualized network service architecture, as we believe this is a more realistic application than an application applied to an aggregate with undifferentiated traffic. In particular, we focus on VoIP traffic because it still today represents an important portion of ISPs revenues and as it requires QoS guarantees also in terms of availability.

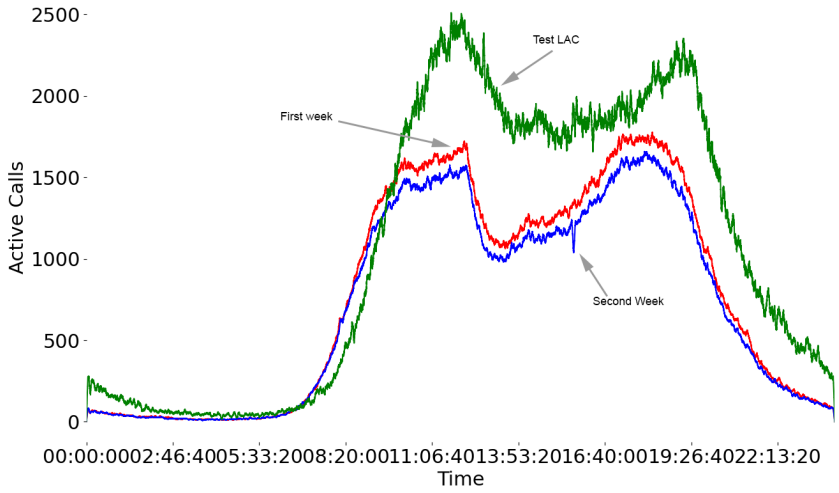


Figure 4.4: VoIP call distributions emulated in the experiments

We evaluate the SYRROCA ability to detect and characterize anomalies, we disrupt the normal

working conditions injecting anomalies following three different scenarios. In the *first scenario* we tested how stressing the physical CPU in the PCSCF container is perceived by the autoencoders and how this stress propagates from the physical to the container layer. We injected a persistent physical CPU stress which increases over time in evenly time distributed increments of 10% during one hour across 32 CPUs, starting from 10% up to 80% of single CPU capacity. Each stress cycle is repeated ten times. The *second scenario* consists in injecting packet loss to generate calls failures.SIPp allows simulating packet loss by simply blocking outgoing messages or discarding received messages. In particular, we alter the call distribution of March 16, 2020, blocking 50% of INVITE (SIP message) acknowledgments, causing at least 50% of calls to fail. In the *third scenario* instead we stress the vIMS core with a call profile exceeding the resources available to the vIMS network functions. To do that, we chose to inject the call distribution of March 22, 2020 from a different LAC than the one used for training, serving more users (Figure 4.4). Actually, even though in our deployment each pod can theoretically use as much memory as the physical server has (best-effort mode), the scripts used to launch IMS services impose a hard-coded memory limit. Nevertheless, we observed that although this script-level limit is not reached, it is possible to overload the vIMS core with a higher amount of traffic as in the selected test LAC. Unfortunately, the SIPp traffic simulation tool showed a limitation on the total number of the simultaneous emulated calls, therefore we could simulate only the first peak of the LAC depicted in Figure 4.4, but in any case not invalidating the correctness of the test.

4.2.2 Dataset

The whole platform is monitored through Prometheus node-exporter [176] for the physical level, while Pods and containers are monitored through Kubernetes embedded CAdvisor [177] agent. Both exporters are compliant with Prometheus data model and architecture so that metrics can be exported through GET requests at a specific polling frequency. Furthermore, collected metrics are explicitly typed as counters or gauges, so that pre-processing becomes easier. In our deployment metrics are directly collected from both CAdvisor and NodeExporter with a Java script that stores metrics in a column-like format every 5 s, forming time-series of 17280 values per feature and per day during 21 days. We downsampled the series to a 30 s frequency (2880 samples/day) to keep a good trade-off between time complexity and training error.

Tables A.1 and A.2 list the names of considered metrics for the physical and the virtual layer

	CPU	Network	Memory	Disk	Total
Physical	370	290	40	260	960
Virtual	60	80	160	230	530

Table 4.1: Number of features per layer and resource type

respectively. Note that each metric name is repeated several times in the dataset, as it can refer to different, pod, container, network interfaces etc... Table 4.1 instead details the number of features per layer and resource group, showing the important magnitude in the number of features that justifies our choice of using deep autoencoders to effectively compress high dimensional inputs vectors.

4.3 The SYRROCA framework

In this section, we detail the architecture of the framework we proposed with a particular attention to the anomaly detection and characterization phases. The framework is called SYRROCA SYstem Radiography and ROot Cause Analysis. Figure 4.5 shows a simplified diagram of the proposed framework. The architecture represented here is split by layer, duplicated to analyze separately both physical and virtual layer metrics to comprehensively analyze the whole stack.

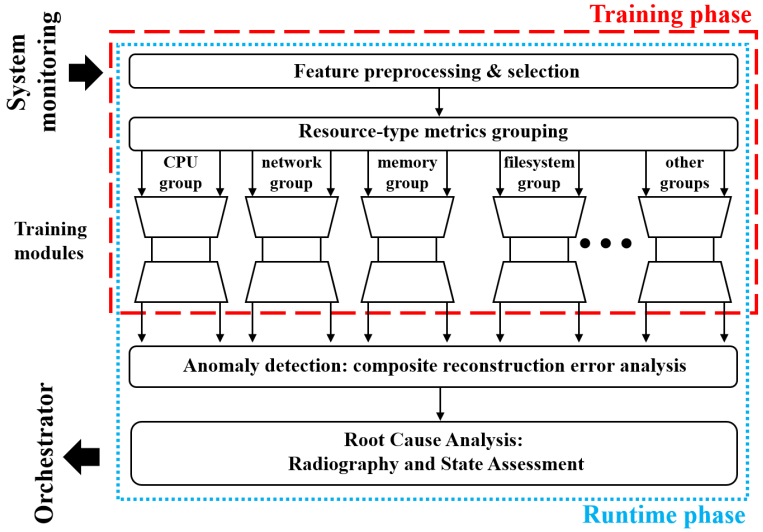


Figure 4.5: SYRROCA framework functional diagram

4.3.1 Metrics collection and pre-processing

In order to monitor a softwarized network platform to infer its state, we can collect heterogeneous data related to the various subsystems in stake. Metrics include categorical information in a text shape (alarms, logs) or in a numerical shape, encompassing metrics and KPIs (Key Performance Indicators). In this work we focus our analysis on numerical metrics, leaving out other textual data that requires other type of pre-processing such as semantic text processing capabilities like in log analysis [178, 179]. The continuous collection of metrics results in a set of time series at different time intervals given by a scraping frequency. Depending on the metric source, each time series may exhibit different properties which requirs pre-processing to make the analysis more effective. For instance, the number of sent/received packets inherently encompasses a monotonically increasing trend, which makes the associated time series a cumulative *counter*. On the contrary, other time series, generally referred to as *gauge*, may arbitrarily describe increasing/decreasing metrics without an inherent trend; e.g., the frequency values taken by a computing processing unit or its temperature. Thereby, to analyze a dataset containing several heterogeneous time series, we pre-process data as follows:

- ***De-trending***. Counters-based time series, if not properly pre-processed, are non-stationary, which tends to produce unreliable and spurious results leading to poor understanding and forecasting capabilities [180]. Furthermore, counter metrics evolution is rather characterized by its increments rather than the absolute cumulative value. On the other hand, gauges-like metrics do not exhibit an a-priori trend and are characterized by each instantaneous value. Therefore we keep the raw values for gauges-like metrics while we only maintain their increments for counters like metrics;

- ***Re-sampling***. Depending on the monitoring scraping frequency, the temporal resolution of each metric may be different and in some cases so fine-grained that spurious outliers spikes could worsen data quality. Furthermore, high time-series resolution would make the training set huge, which proportionally increases training duration without any quality increase guarantee. In our conditions, we found 30 s to be a good frequency value considering how fast anomalies should be detected;

- ***Re-scaling***. Since some metrics values may have a relatively big magnitude while others may have a small one, it is important to re-scale the input data into a uniform range. Indeed, when

CPU	Network	Memory	Disk
Levy (15%)	PowerLaw (25%)	Alpha (14 %)	Betaprime (34 %)
Net (15 %)	Alpha (20 %)	Net (12 %)	Jhonsonsu (26 %)

Table 4.2: Best-fit distributions per resource group

training a NN with a gradient descend algorithm, the learning rate is proportional to the magnitude of the inputs, which means that if the inputs are of different scales, the weights connected to some inputs will be updated much frequently than other ones, biasing the learning [181]. This is especially important for LSTMs, which are sensitive to the scale of the input data when the (default) *sigmoid* or *tanh* activation functions are used.

In the state of the art two techniques are proposed to re-scale data: standardization and normalization; the former assumes that observations fit a Gaussian distribution (with a well behaved mean and standard deviation) and consists in shifting the distribution of each metric to have a mean of zero and a standard deviation of one (unit variance), while the latter consists in transforming the original metrics range so that all values fall within the $[0, 1]$ range. To decide which feature re-scaling approach to use we analyzed metrics distributions. We use the chi-squared Pearson's cumulative test [182, 183] to characterize the goodness of fit of different statistical distributions. The chi-squared statistic, χ^2, is a normalized sum of squared deviations between observed and theoretical frequencies. The χ^2 bins data into n bins based on percentiles so that each bin contains approximately an equal number of values; for each fitted distribution the expected count of values in each bin is predicted from the distribution. The chi-squared value is the sum of the relative squared error for each bin.

Table 4.2 represents the distributions obtained for the four resource groups after the application of the χ^2 test. We can observe that none of the analyzed metrics fit a Gaussian distribution; we therefore select normalization as the re-scaling technique.

During the training phase, SYRROCA learns the nominal conditions from the metrics characterizing a given layer and a resources group involved in a virtualized network service. Autoencoders are trained with a dataset composed of metrics collected during the normal working conditions, so that they can learn a compact representation of the nominal state. It is worth noting that both the quality and the extent of the data used for the training phase greatly affect the representation. Indeed, during the training phase, autoencoders are fed with anomaly-free samples during a sufficient period of time

to learn the dynamics for each metrics group and layer to be characterized. Our framework uses dedicated deep autoencoders for each group of resources (CPU-related, memory-related, network-related, and file-disk-related) and layer (physical and virtual) in order to characterize anomalies occurring in a softwarized service in a fine-grained manner. Consequently, we used a total of 8 deep autoencoders, 4 to analyze virtual layer metrics and 4 more to handle physical layer metrics (i.e. a single autoencoder per resource group). It is worth noting that, for future possible applications of SYRROCA to other applications, additional metric sources and groups can be added with no restriction. As mentioned in the previous section, neural networks struggle to learn with high-dimensional inputs. Hence splitting the dataset per resource group streamlines learning. Additionally, it reduces training time. In fact, as described in [156] the LSTM cell epoch update complexity is $\mathcal{O}(W_i)$, where W_i is the number of cell's weights. For a standard implementation of an LSTM cell, $W_i = 4h_i \times (h_i + h_{i-1})$ where h_i is the number of hidden units and h_{i-1} is the dimension of the layer input, which is the previous layer's output. When using LSTMs in deep autoencoders, the number of hidden units depends on compression level $c_i \in \mathbb{R}$. Thus $W_i = 4Nc_i \times (Nc_i + Nc_{i-1})$ where N is the number of considered metrics. Subsequently, each cell update complexity is $\mathcal{O}(N^2)$. Assuming a deep autoencoder composed of M encoding and M decoding levels, the training complexity of the entire deep autoencoder is $\mathcal{O}(M \cdot N^2)$ for each training epoch. In contrast, if input metrics are split into n^g different groups (resources groups in the remainder of the manuscript), the deep autoencoder training complexity is reduced to $\mathcal{O}(M \cdot N^2/n^g)$ for each epoch.

4.3.2 Training

Whatever library is used to implement the LSTM model, the input to every LSTM layer must be a three-dimensional tensor, which can be seen as three-dimensional array. As shown in Figure 4.6, LSTM tensor dimensions are the number of samples, the number of time steps we want to observe each sample and the number of features/metrics the LSTM layer has to analyze.

Because of that, before training the AEs, we rework the dataset structure to match figure 4.6 structure. We use two time-steps per sample, which ensure fast learning, and overlapping sequences, which preserve the temporal correlation among the various samples. Even though an LSTM cell can theoretically store into its internal memory an infinite sequence of samples, actual implementation requires imposing a limit on the growth of the memory [184]. This is generally achieved with internal

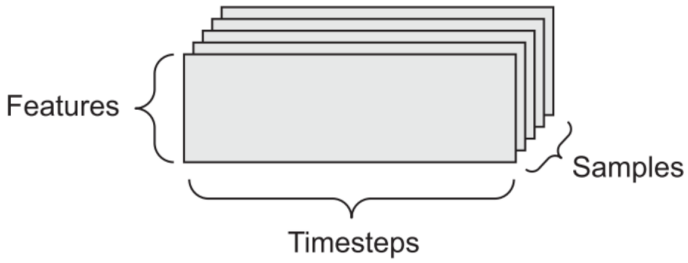


Figure 4.6: LSTM tensor

Hyper-parameter	Value
Batch Size	64
Epoch	600
LSTM activation function	elu
LSTM recurrent activation function	sigmoid
Dropout rate	0.2

Table 4.3: LSTM cell hyper-parameter

memory reset after each *batch* of samples. According to the mini-batch training technique [185, 186], accumulating NNs weight changes over some number u (the batch size) of instances before actually updating the weights, helps to achieve a fast training convergence. Thereby, the batch size value has to be tuned to limit the memory usage and speed the learning while preserving the ability of AEs to learn on a meaningful temporal dynamic. As the training dataset spans three weeks of simulated calls, it is inherently characterized by a one-day pattern. We thereby set the batch size equal to the size of a daily dataset so that the LSTM internal memory cell is only erased after working on an entire day, learning daily metrics interdependencies.

To run an effective AI/ML model, AEs architecture and hyper-parameters must be tuned; even if there exists some rules of thumb or some widely recognized best practices, the brute force approach, which consist of testing several combinations of hyper-parameters and architectures, is the most commonly used and accepted method. Table 4.3 reports some insights about chosen values for some most meaningful hyper-parameters and figure 4.7, instead, depicts the architecture of each deep autoencoder.

Encoder and decoder are composed of two LSTM layers which reduces the input vector dimension by 20% each. One dropout regularization layer is used to prevent over-fitting, which particularly affects Deep neural networks [187]. According to the state of the art, over-fitting can be reduced by

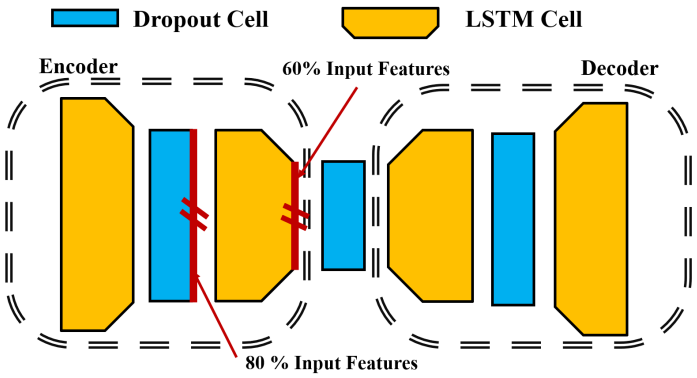


Figure 4.7: SYROCCA deep Autoencoder architecture.

fitting all possible different neural networks architectures on the same dataset and then averaging the predictions from each model [188]. However, this is not feasible in practice. With dropout, during training, some layer outputs are randomly "dropped out"; therefore some layers look like one with a different number of nodes and links to the prior layer, mimicking different architectures.

4.3.3 Anomaly detection and characterization

Table 4.4 summarizes the notation used throughout the chapter.

N	Number of total analyzed metrics
$G = \{g_1, ..., g_s\}$	Set of considered s resources groups
$L = \{l_r, ..., l_r\}$	Set of considered r layers
$N^{l,g}$	Number of metrics referring to resources group g at layer l
τ	Dataset time length
$X_j^{l,g}(t) = [x_j^{l,g}(1), ..., x_j^{l,g}(\tau)]$	time-series of the j^{th} metric in input to the AE operating on resources group g and layer l metrics
$\tilde{X}_j^{l,g}(t) = [\tilde{x}_j^{l,g}(1), ..., \tilde{x}_j^{l,g}(\tau)]$	time-series of the j^{th} metric produced by the AE operating on resources group g and layer l metrics
$\mathcal{D}$	Set of detected deviations
d_t	deviation at time-step t
$T^{l,g}$	MSE threshold for resources group g at layer l
$SE_j^{l,g}(t) = [x_j^{l,g}(t) - \tilde{x}_j^{l,g}(t)]^2$	Feature-wise reconstruction squared error
$p_j^{l,g}(t)$	j^{th} feature MSE contribution rate

$F_t^{l,g}$	most deviated feature index set for deviation d_t of resources group g, layer l

Table 4.4: Table of notations

Reconstruction error design

In SYRROCA, an anomaly is meant as a meaningful deviation from nominal conditions. The framework is based on LSTM AEs wich allow to detecting anomalies when their reconstruction error exceed e fixed threshold. Indeed, when nominal conditions significantly deviate, the autoencoder fails in reconstructing those conditions and the reconstruction error increases.

In general, depending on the problem, it is possible to choose among several types of reconstruction error. In the state of the art, the most used ones are the Mean Squared Error(MSE), the Root MSE (RMSE), the Mean Absolute Percentage Error (MAPE), the Mean Absolute Error (MAE).

The MAE it is defined as the mean of the absolute error $MAE = \frac{1}{n}\sum||x_t - \hat{x_t}||$, where x_t is the actual value and $\hat{x_t}$ is the forecasted one at time t. It is, in general, used when outliers are not expected to be frequent and it is therefore not suitable for anomaly detection [189]. In our case, as we do not have negative values, the MAE boils down to the Mean Bias Error (MBE).

The MAPE is defined as $MAPE = \frac{1}{n}\sum||\frac{x_t - \hat{x_t}}{x_t}||$, and it is in general used to compare different models. However it cannot be used if actual values might contain zeros [190], which is the case for our dataset.

The Mean Squared Error(MSE) and the Root MSE (RMSE) are generally the most used loss functions for anomaly detection: the former is defined as $MSE = \frac{1}{n}\sum(x_t - \hat{x_t})^2$ while the latter is the root of the MSE, $RMSE = \sqrt{MSE}$. Since the RMSE gives a relatively higher weight to large errors compared to smaller ones, it is generally preferred when larger errors are considered much worse than smaller ones [191].

In SYRROCA, AE inputs are re-scaled to the $[0, 1]$ range, thus the training reconstruction error is always lower than 1. Thereby, as $\sqrt{x} > x$ for $x \in \mathbb{R} \wedge x < 1$, the RMSE always produces greater error values than MSE. As the threshold is computed to be the 99.9% quantile of the reconstruction error distribution, the threshold also is always greater in the RMSE case. Accordingly, using the RMSE only produces a re-scaled version of the same information we can compute with the MSE, leading to

identical anomaly detection outcome; we could also verify this aspect experimentally, with the same accuracy, recall, precision and F1 score for two sets of RMSE and MSE-based AEs.

As a conclusion on the choice of the type of reconstruction error we chose the MSE because computationally less heavy than the RMSE, which requires in addition to the MSE computation, also the computation of the squared root (of the MSE) with identical results (with only the MSE).

Anomaly characterization

An autoencoder is trained to reconstruct the nominal conditions with low error. However, when nominal conditions significantly deviate, the autoencoder fails in reconstructing those conditions and the error increases.

For an AE trained with $N^{l,g}$ inputs and outputs the MSE is defined as:

$$MSE^{i,g}(t) = \frac{1}{N^{l,g}} \sum_{j=1}^{N^{l,g}} [\tilde{X}_j^{l,g}(t) - X_j^{l,g}(t)]^2 \tag{4.1}$$

where $\tilde{X}_j^{l,g}(t) = [\tilde{x}_j^{l,g}(1), ..., \tilde{x}_j^{l,g}(T)] \in \mathbb{R}^T$ and $X_j^{l,g}(t) = [x_j^{l,g}(1), ..., x_j^{l,g}(T)] \in \mathbb{R}^\tau$ are respectively the output and the input vectors of the autoencoder working on group g, where τ is the size of the considered time-window.

Even though the MSE provides an efficient way to detect anomalies on a big set of metrics, it is a global metric not rich enough to characterize the anomaly. We therefore design an approach to start from the MSE to determine the causes that contribute the most to the anomaly, as follows.

Let D and $d_t \in \mathcal{D}$ be the set of 'deviations' and a given deviation at time t, such that the MSE of a resources group exceeds a threshold value $T^{l,g}$. We use the 99.9% quantile as the threshold for each metrics group, i.e., 0.1% of the training samples are marked as raw anomalies by each autoencoder; note that, depending on the scales involved, this statistical threshold may be increased at will.

To characterize anomalies using autoencoders we propose to compute the contribution of each feature to the MSE, computed as the feature-wise reconstruction squared error $SE_j^{l,g}(t) = [x_j^{l,g}(t) - \tilde{x}_j^{l,g}(t)]^2$ over the sum of the squared errors across all the features in a given resources group:

$$p_j^{l,g}(t) = \frac{SE_j^{l,g}(t)}{\sum_{j=1}^{N^{l,g}} SE_j^{l,g}(t)} \tag{4.2}$$

The closer $p_j^{l,g}(t)$ is to 1, the stronger is the contribution of feature l in group g to the MSE of that group

($MSE^{l,g}$). Let us define $B^{l,g}(t) = \{b_1^{l,g}(t), b_2^{l,g}(t), ..., b_n^{l,g}(t)\}$ as the set of decreasingly ordered $p_j^{l,g}(t)$. Thus, taking $b_1^{l,g}(t), ..., b_k^{l,g}(t) \in B$ with $k \leq N^{l,g}$ so that $\sum_{j=1}^{k} b_j^{l,g} \geq 0.9$ the features corresponding to these first k values of $p_j^{l,g}(t) \in B$ are those that are reconstructed with the highest error and jointly contribute to at least the 90% of the MSE. Consequently the set:

$$F_t^{l,g} = \{j : b_j^{l,g}(t) \in B^{l,g}(t), 1 \leq j \leq k\} \tag{4.3}$$

contains the indices of the features that deviated the most from their nominal dynamics on layer l, group g; therefore, it gives an insight of symptoms to analyze the root cause.

4.3.4 Radiographies

Depending on the type of anomaly impacting the system, it may be contained in a single layer or may propagate to other layers. For instance, a process inside a container intensively using assigned CPU may be seen as an anomaly at the virtual/container level, while not affecting the quality of the delivered service. To understand how anomalies propagate across layers and impact the service, we propose to combine the reconstruction error $MSE^{l,g}$ and $MSE^{l+1,g}$ of two consecutive layers to obtain a 2D density plot, referred in the remainder of the paper as *radiography*, given its visual similarity with common radiographies.

We can have two types of radiographies:

- *Service cross-layer view:* this view correlates the $MSE^{virt,g}$ of the virtual layer with the $MSE^{service,g}$ metrics characterizing the delivered service. For simplicity, in this chapter we treat the case of a single service metric. Hence, let $f(MSE^{virt,g}, < service\_metric >)$ be the bi-variate function joining the $MSE^{virt,g}$ of virtual layer and group g to values of the selected service metric (e.g., number of failed calls in the vIMS use-case).

- *Infrastructure Cross-layer view:* this view correlates the MSE of the physical and virtual layers, for a given features group. Let $g(MSE^{phy,g}, MSE^{virt,g})$ be a bi-variate function describing how the MSE from the two layers of a given group are related to each other.

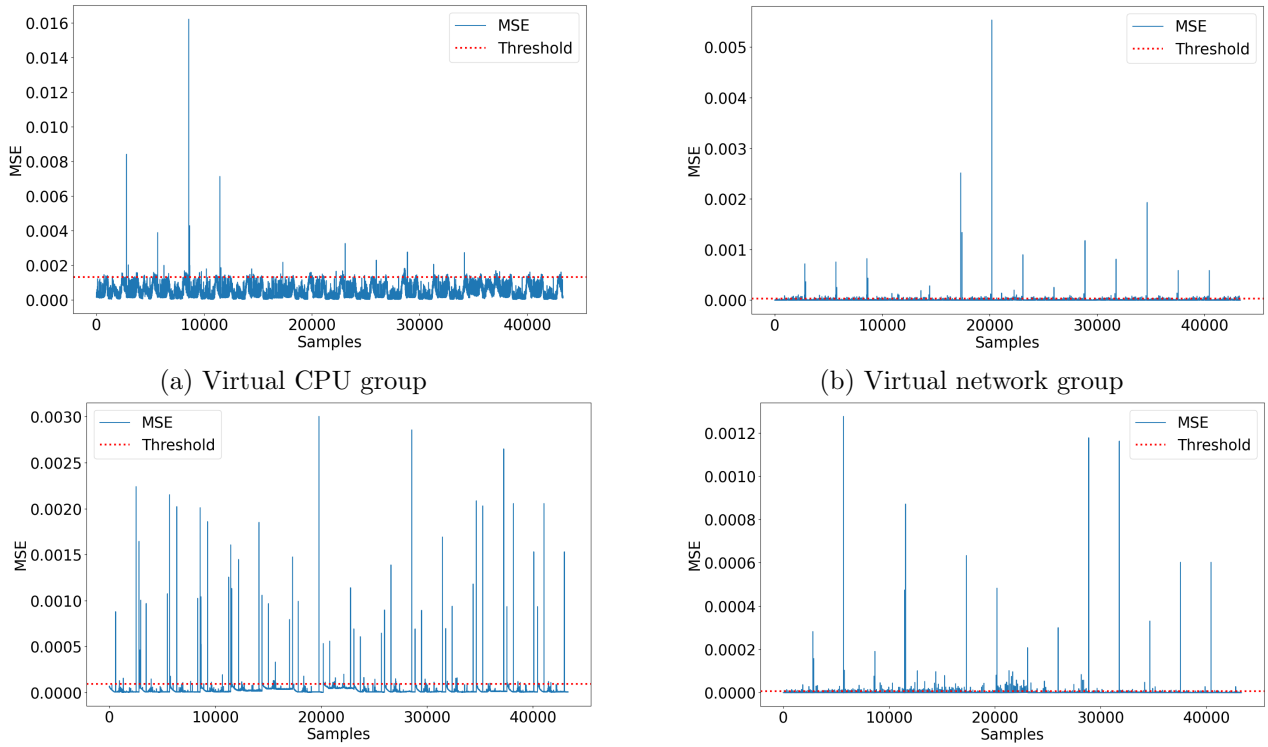
In practice, a radiography is a 2D density plot, computed through the Kernel Density Estimation (KDE), which is used to estimate the density of the f and g bi-variate functions. In particular, the density estimator is defined as $\hat{f}(x, H) = \frac{1}{n} \sum_{i=1}^{n} K_H(x - X_i)$, where $(X_1, ..., X_n)$ are the bi-variate

function samples, K is the used Gaussian kernel and H is a bandwidth parameter, chosen with a well known rule of thumb [192]. A color/grey scale mapping density from high to low with colors from black to white, is used to visualize the computed KDE, obtaining the so-called radiography. Accordingly, the KDE-based radiography represents the density for each function for a chosen time window: the higher the density of a given point area is, the darker the color; darker regions represent most observed conditions. Density zones along the space bisector denote a direct propagation across variables/layers, while density zones near any of the two axes denote an impact affecting only one variable/layer hence with no propagation among the two. We later showcase radiographies for the vIMS use-case showing how to spot and characterize anomalies.

4.4 Experimental results

4.4.1 Training on a nominal scenario

Figure 4.10 depicts $MSE^{virt,g}(t)$ for the four AEs fed with the virtual CPU, network, memory and file system metrics groups respectively. Note that it is not required to have an MSE equal to zero for time-stamps representing nominal conditions. Nonetheless, we are interested in MSE differences between points to get an insight on the deviations. Through the methodology proposed in section 4.3.3, we found out that in the memory metrics group MSE (4.10 (c)), 57.4% of samples crossing the threshold are characterized by the set of features F =[MEMORY FAILURES TOTAL{POD=HSS, TYPE=PGFAULT, SCOPE=HIERARCHY}, MEMORY FAILURES TOTAL {POD=HSS, TYPE=PGFAULT, SCOPE=CONTAINER}]. However, for CPU, network and file system groups there is no predominant type of features F characterizing those deviations. Therefore, we can conclude that the dynamics of the memory usage of the HSS during normal activity is characterized by a repeated fixed pattern. This pattern has to be taken into consideration when analyzing test datasets as something somehow belonging to nominal operating conditions. It is worth noting that the MSE trend for the virtual CPU group clearly follows the call distribution (Figure 4.4), which confirms our AEs being able to carefully characterize the learning dataset.



(a) Virtual CPU group

(b) Virtual network group

(c) Virtual memory group

(d) Virtual file system group

Figure 4.8: Training MSE for each metrics group.

4.4.2 Test phase on degraded conditions

We evaluate the SYROCCA ability to detect and characterize anomalies under three different degraded scenarios. Please refer to Section 4.2.1 for a detailed explanation of the three different test scenarios.

First test scenario

Fig. 4.9 shows that: (i) the AE for the virtual CPU metrics group detects at all times those CPU stress at the physical level as deviations above the threshold at the virtual level, (ii) the $MSE^{g=vCPU}(t)$ increases according to the injected increasing physical CPU load with the same trend. This confirms that the AE can detect deviations at physical and virtual layers and characterize relative intensities of those deviations. This experiment is extensible to virtual network and virtual memory, or other data sources groups. Furthermore, when the 1 hour physical CPU stress ends, that MSE behavior falls back to the nominal region under the threshold (diamond points).

Applying the methodology described in Section 4.3.3 to get an insight on the features describing the anomaly, it turns out that features characterizing anomalies correspond to CPU group F =[CPU USER SECONDS TOTAL{POD=PCSCF}, CPU USAGE SECONDS TOTAL{POD=PCSCF}]. This confirms SYROCCA's ability to recognize sets of resources that most deviate for any type of anomaly.

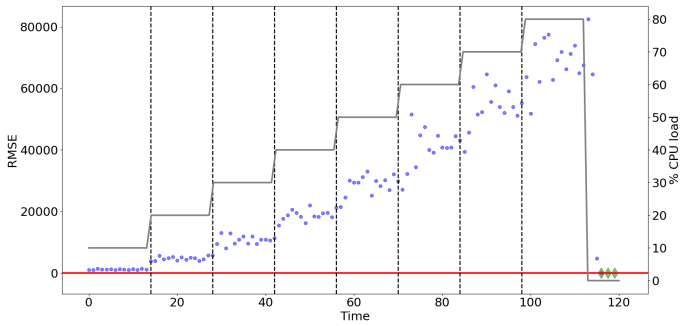
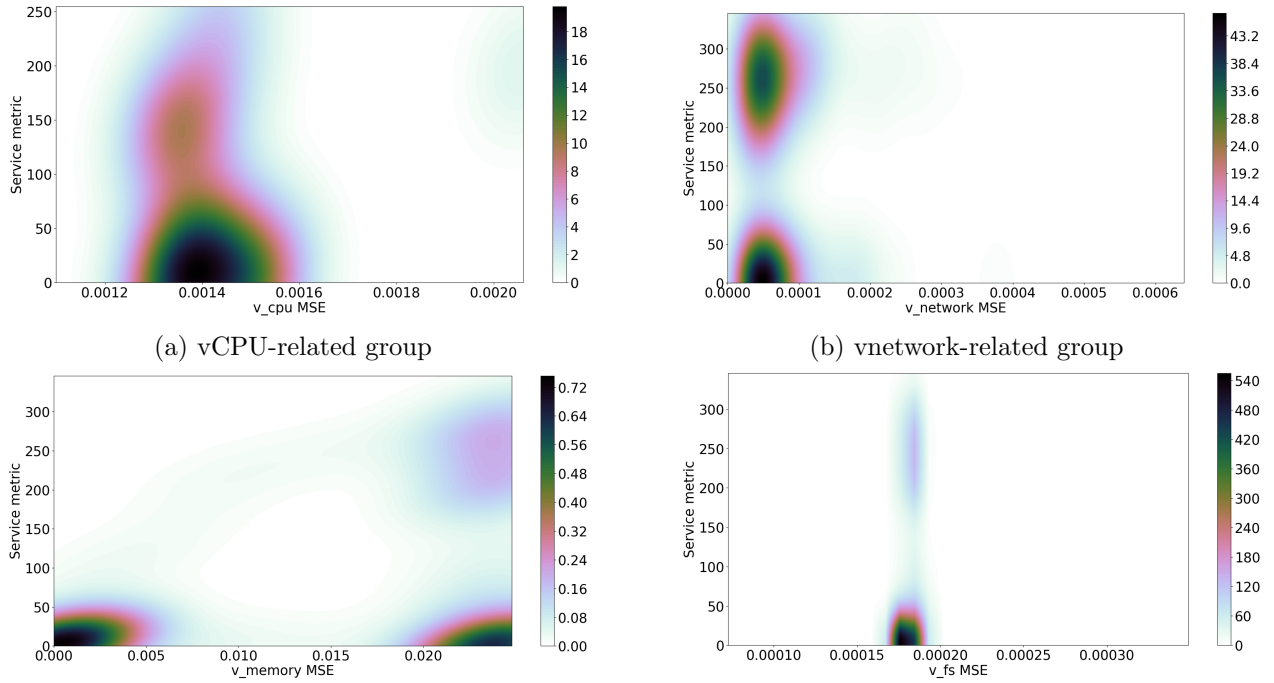


Figure 4.9: Time evolution of the MSE for virtual CPU-related metrics group, under an increasing CPU stress.

Second test scenario

Figures from 4.10a to 4.10d depict the obtained radiography for the test case plotting only anomalous data points. The horizontal axis is the MSE for the analyzed group of metrics while the vertical



(a) vCPU-related group

(b) vnetwork-related group

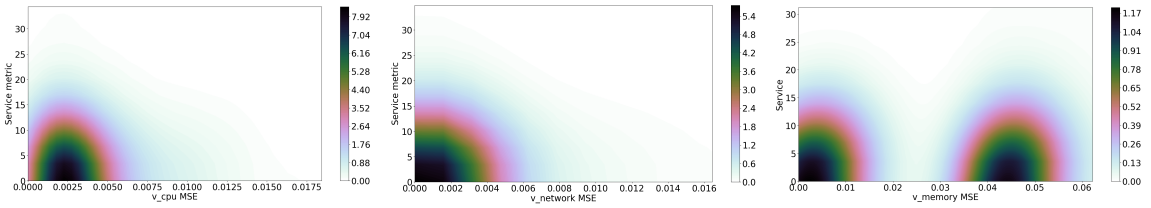
(c) vmemory-related group

(d) vfilesystem-related group

Figure 4.10: vIMS system radiographies under packet loss injection.

one is the number of failed calls. Darker zones denote high-density regions $(MSE^g, failed\_calls)$ while colors from green to white indicate less dense regions. It is worth recalling that the values associated with the color scale correspond to an estimate of the probability density function via KDE, and therefore do not represent a physical density value but only a measure proportional to density. Since Figures 4.10c, 4.10b and 4.10d present high-density zones only for small values of failed calls, one can conclude that CPU, memory and file system are not behind the service degradation. On the contrary, Figure 4.10b clearly depicts two high-density zones, one of them corresponding to more than 250 failed calls, which clearly indicates that anomalies detected from the network metrics group directly impact vIMS service. Indeed, 90% of detected network anomalies are only characterized by metrics related to sent/received packets from/by SCSCF and PCSF. In fact, when a call fails, the SCSCF generates a *Failed_call* message that is redirected to the PCSCF, and then to the user. Moreover, Figure 4.10a depicts a moderate density zone (light orange) for more than 250 failed calls, highlighting that the anomaly slightly impacts the CPU. Similarly, in Figures 4.10c and 4.10d two slighter low density zones (light violet) corresponding to more than 250 failed calls points out an even slighter impact on memory and file system related metrics.

Third test scenario



(a) vCPU-related group (b) vnetwork-related group (c) vmemory-related group

Figure 4.11: vIMS system radiographies under call overload injection.

As expected, radiographies from Figures 4.11a to 4.11c show that the introduced anomaly evenly impacts CPU, network and memory metric groups, mainly seen as high-density zones corresponding to at most 15 failed calls. Furthermore, file system-related metrics are nearly not impacted as only a few samples are recognized as anomaly making it impossible to produce a radiography.

4.4.3 Time-windowed radiography

SYRROCA can characterize the state of a physical or virtual layer and its time evolution. We can analyze this evolution through time-windowed radiographies. In practical usage, SYRROCA is meant to receive test datasets in batches of arbitrary time duration. Instead, to accumulate the features collected within each batch and analyze the whole sequence of batches at once, it is possible to separately analyze each batch. So doing, produced radiographies and state graph describe only the latest evolution of the system and not the whole evolution since the beginning of metrics collection.

Figure 4.12 shows an example of time-windowed radiographies about the system evolution for the CPU stress test case. We show three consecutive radiographies computed with a time window of 40 samples to represent the last 20 minutes. The dark high-density region in the leftmost radiography completely exceeds the threshold on the virtual layer, but is partially exceeded in the physical layer: the stress is immediately perceived as an anomaly at the virtual layer, while at the physical layer the stress starts to be recognized as an anomaly. As the simulation continues, the anomaly starts getting more and more deviated for both virtual and physical layers. This behavior can be seen in the middle radiography, where the high-density zone spreads around the space bisector, thus denoting simultaneous stress and direct propagation across physical and virtual layers. During the last 20-minute radiography the system reaches the greatest level of deviation but it remains focused on a very

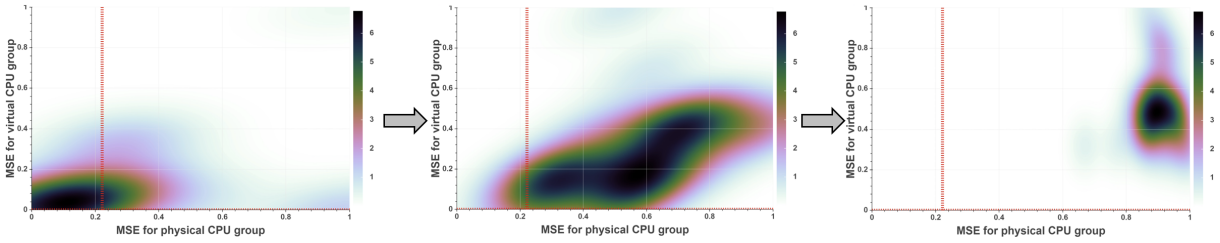


Figure 4.12: Radiography time evolution for the CPU stress test case

concrete region located to the rightmost side of the radiography.

4.5 Conclusion

In this chapter we proposed a LSTM autoencoder-based methodology to detect and characterize network deviations in softwarized environments. LSTM AEs are fueled with time series composed of hundreds of metrics collected from the physical and the virtual layer composing a virtualized infrastructure. A set of 8 AEs is used to better profile the CPU, network, disk and memory metrics for both the physical and virtual layers. A threshold on AEs mean squared error is exploited to detect the deviations from the learned nominal working conditions.

The Radiography representation is proposed to detect and visualize anomaly propagation through layers. The proposal is validated with a proof-of-concept based on a vIMS deployed using Kubernetes.

In the following chapter, we demonstrate how to leverage AEs reconstruction error to comprehensively assess the virtualized system state.

Chapter 5

Virtualized system state assessment

Content

To cope with the high heterogeneity and number of software components, in the previous chapter we proposed an AI approach to detect anomalies. Here we extend the framework to assess the running state and the state deviations of a softwarized infrastructure making use of containerized services. Our framework learns the nominal working conditions of the infrastructure, based on which deviations from the learned reference are detected and analyzed. We detail how characterizing state deviations can explain the root causes of service failures. We implement and validate the proposed framework through experimental tests on a containerized voice-over-IP IMS platform managed by Kubernetes. This chapter reports the content of publication [14].

5.1 Introduction

Network softwarization eases the adoption of so-called 'cognitive network' approaches, e.g. referring to a closed-loop process consisting of sense, learn, decide, policy and act phases [12, 13]. The observations captured by the sensors (sense) help to build a model from the useful observations (learn), which is in turn used by a decision-making module to choose (decide) the actions to be taken based on possible moves and learned experience. Potential actions, i.e. strategies stored in the policy module (policy), are shortlisted by the planning module, so that, finally, the actuators execute (act) selected re-configurations [11]. The policies developed by the cognitive loop aim to achieve a final end-to-end goal dictated by the business and/or user requirements such as maintaining a certain Quality of Service (QoS) to fulfill a Service Level Agreement (SLA). Build a precise enough model of the network state from the sensed data is a paramount step in the cognitive/automation loop.

In a softwarized environment, recent advanced monitoring tools (e.g., Prometheus) allow retrieving thousands of metrics at different levels to sense a connect-compute platform composed of both computing and networking components. However, automatically extracting relevant features from such a massive amount of data to assess the system state is a challenge that we firstly addressed in the previous chapter, where we pose the bases of the so-called SYRROCA (System Radiography and Root Cause Analysis) framework. In this chapter, we further develop the SYROCCA framework, proposing a cross-layer state characterization of the system. We show how this novel analysis allows characterizing anomalies at any layer and their propagation across layers. We run tests in the same containerized vIMS architecture we introduced in the previous chapter. We also propose a formalization for the autonomic anomaly remediation problem as a Reinforcement Learning algorithm. Finally, we propose a trust and reputation model to quantify MANO entities contribution to softwarized system resilience.

Table 5.1 summarizes the notation used across the first part of the chapter.

N	Number of total analyzed metrics
G	Set of resources groups
L	Set of layers
U	Set of considered v computational units
$N^{l,g}$	Number of metrics referring to resources group g at layer l
$\mathcal{D}$	Set of detected deviations

d_t	deviation at time-step t
$T^{l,g}$	MSE threshold for resources group g at layer l
$F_t^{l,g}$	most deviated feature index set for deviation d_t of resources group g
$\Theta : F_t^{l,g} \mapsto \{(g_1, u_1), ..., (g_k, u_k)\}$	Function associating to $F_t^{l,g}$ the couples (g, u) corresponding to features in $F_t^{l,g}$

Table 5.1: Table of notations

5.2 System State Inference

In cognitive network approaches [67, 11, 193] the sensing of the environment and learning the state is the preliminary step to inference. Optimal decision-making processes in orchestration concern the capability to identify with low uncertainty the state of network resources to know where to apply those remediation actions, so that they are effective enough to revert the deviated conditions towards normalcy.

The notion of system 'state' can be declined in different ways according to the addressed problem. In SYRROCA, we distinguish among three types of states:

- *Nominal State*: the system is in normal working condition when the MSE does not deviate for each group of resource metrics.
- *Training Degraded State*: the system is in an anomalous but known-in-advance working condition. In these cases, the MSE falls above the threshold for at least one group of resource metrics in a given layer.
- *Test Degraded State*: the system is in an anomalous and unknown working condition during the run-time test when the MSE falls above the threshold for at least one group of resource metrics in a given layer.

While the anomalous unknown states are meant to be detected in the testing phase, the nominal state and the known degraded states can be ex-ante characterized for a given use-case.

Let a layer, whether physical or virtual, be fully characterized by two element sets:

- the set of resources groups, denoted as $G = \{g_1, ..., g_s\}$, where each g_k can for instance denote CPU, memory, network and file-system related metrics;
- the set of computing units, denoted as $U = \{u_1, ..., u_v\}$, where each unit may represent a virtual machine, a container, or a server in the case of a physical layer.

In order to provide an insight on the deviation that leads the system to a degraded and anomalous state, we would like to compute the partition of the set containing all the known degraded states $\mathcal{D} = \{d_{t_1}, ..., d_{t_k}\}$ in a specific layer. Indeed, according to the partition set definition, $part(\mathcal{D})$ groups all elements in D into non-empty subsets, i.e. equivalence classes, in such a way that every element is included in exactly one subset. The simplest way to obtain such a partition is to define an equivalence relation for which the set of its equivalence classes is a partition of D. The simplest equivalence relation $\mathcal{R}$ one could imagine would group together deviated states which are characterized by the same sets $F_t^{l,g}$ of most deviated feature indexes. Nonetheless, depending on the number of features composing the multivariate time series collected from the monitored softwarized infrastructure, the number of possible classes, defined by $|part(\mathcal{D}|\mathcal{R})|$, could be huge. For instance, it is common to have at least n=200 metrics for a physical layer, which gives: $|part(\mathcal{D}|\mathcal{R})| = \sum_{k=1}^{n} \binom{200}{k} \cong 1.67e^60$ possible deviated states. Moreover, remediation actions may not have sufficient granularity to act on a single metric, rather they may act on a specific resource and computational unit. Hence, we propose to further group anomalies with respect to the impacted resource (e.g. CPU, memory, disk) and the computational unit (e.g. container, VM, server), so that deviations referring to the same sets of resources and computational units belong to the same equivalence class. This helps in narrowing down the choice of possible mitigation actions to those that act on the identified resources and computational units.

Let's $\Theta : F_t^{l,g} \mapsto \{(g_1, u_1), ..., (g_k, u_k)\}$ be a function mapping the features indexes set $F_t^{l,g}$ to the corresponding network resources and computation units. We define then the relation $\mathcal{R}'$:

$$\mathcal{R}' := \{d_{t_a} \sim d_{t_b} \; if \; \Theta(F_{t_a}^{l,g}) \equiv \Theta(F_{t_b}^{l,g}) \forall g \in G, l \in L\} \tag{5.1}$$

which is simply verified to be an equivalence relation as it is:

- Reflexive, as $\Theta(F_{t_a}^{l,g}) \equiv \Theta(F_{t_a}^{l,g})$;
- Symmetric, because if $\Theta(F_{t_a}^{l,g}) \equiv \Theta(F_{t_b}^{l,g})$ then $\Theta(F_{t_b}^{l,g}) \equiv \Theta(F_{t_a}^{l,g})$;

- Transitive, since if $\Theta(F^{l,g}_{t_a}) \equiv \Theta(F^{l,g}_{t_b})$ and $\Theta(F^{l,g}_{t_b}) \equiv \Theta(F^{l,g}_{t_c})$ then $\Theta(F^{l,g}_{t_a}) \equiv \Theta(F^{l,g}_{t_c})$.

Thereby, $\mathcal{R}'$ correctly induces the desired partition on D. Given a deviation d_{t_a} the class $C(d_{t_a})$ of similar deviations through $\mathcal{R}'$ is defined as :

$$C(d_{t_a}) = \{d_{t_k}, \forall k | \Theta(F^{g}_{t_k}) \equiv \Theta(F^{g}_{t_a}) \forall g \in G, l \in L\} \tag{5.2}$$

For example, given the deviation d_{t_a} characterized by $F^{virt,CPU}_{t_a} = \{21 : container\_cpu\_load\_average\_10s\{dns\}\}$ and $F^{phy,mem}_{t_A} = \{33 : node\_memory\_MemFree\_bytes\{server1\}\}$, $\Theta(F^{virt,CPU}_{t_a}) \cup \Theta(F^{phy,mem}_{t_a}) = \{(CPU, DNS), (memory, server1)\}$ thus $C(d_{t_a})$ contains d_{t_a} along with every other degraded state characterized by a deviation on both the CPU of the DNS and the memory of the server1. Therefore, we define the system states as:

$$S_t = \begin{cases} Nominal & \text{if } MSE^{l,g}(t) < T^{l,g} \ \forall g \in G \wedge \forall \ l \in L \\ \{(g_1, u_1), ..., (g_k, u_k)\} & \text{if } \exists \ g \in G \wedge \exists \ l \in L \mid MSE^{l,g}(t) \geq T^{l,g} \end{cases} \tag{5.3}$$

It is worth noticing that through $\mathcal{R}^{'}$ the cardinality of $part(\mathcal{D}|\mathcal{R}^{'})$ is greatly reduced with respect to the cardinality of $part(\mathcal{D}|\mathcal{R})$. Indeed, for the container level we have $|part(\mathcal{D}|\mathcal{R}^{'})| = \sum_{k=1}^{s} \binom{s}{k} \times \sum_{k=1}^{v} \binom{v}{k}$, where s is the number of resources and v is the number of the computational units the features refer to. Considering the set of resources $\{CPU, memory, network, filesystem\}$ and a set of 5 containers $|part(\mathcal{D}|\mathcal{R}^{'})| = 465$, which is then the total number of possible deviated states. For a physical layer composed by 3 physical servers, using the same set of resources we have $|part(\mathcal{D}_T|\mathcal{R}^{'})| = 217$ possible deviated states.

5.3 Experimental results

In this section we report the results of the experimental analysis conducted on the same testbed presented in the previous chapter. We will refer to the same three test scenarios defined at 4.2.1.

5.3.1 Training - known state characterization

The aim of the learning phase is to acquire a characterization of the nominal states as well as known degraded states with respect to the nominal conditions, so as to be able to detect future deviations from both nominal and known degraded states during the run-time usage for testing.

The nominal scenario is simulated through several SIP clients that first register to the vIMS core and then start a call. The vIMS containerized platform is tailored to correctly process the previously mentioned emulated VoIP traffic load.

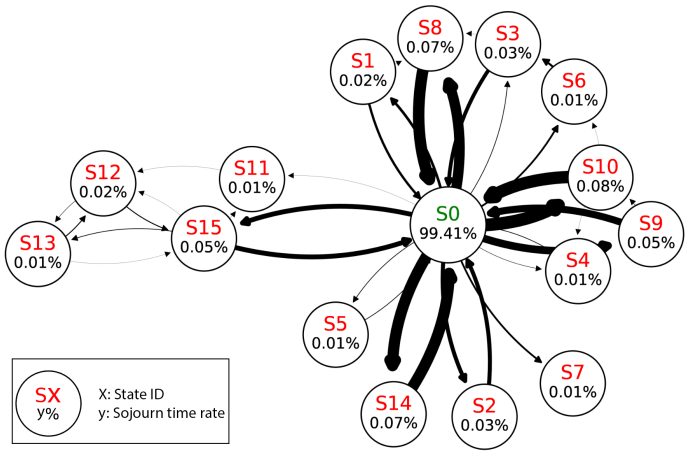


Figure 5.1: State graph obtained during the training phase

Figure 5.1 represents the states learned during the training phase; they are obtained as explained in Section 5.2. States are connected within a directed graph, where an edge indicates any state transition that occurred during the learning phase. The nominal state (or reference) is tagged as $S0$, while the degraded states are tagged as SX, where X is a unique identifier to unequivocally characterize each degraded state. Table 5.2 summarizes the taxonomy of all degraded states detected across our tests. The thickness of the edge transition between states is proportional to the number of times it occurs. Under each state label, it is quoted the percentage of time-steps the infrastructure sojourned in the given state during the simulation.

In order to report only on relevant transitions, we omit the states in which the system stays less than three time-steps (1 minute and 30 seconds). Some state transitions may not appear, such as outgoing edges from the state $S7$ to $S0$ in Figure 5.1. Full states graphs are available at [170]. As each AE threshold is set to the 99.9 quantile of the training MSE distribution, each AE inherently detects 0.1% of the training samples as anomalies. Consequently, when each of the eighth AEs detects anomalies at different time-steps the maximum amount of detected anomalies is 8% over the training dataset.

As shown in Figure 5.1, the most frequent transitions in the virtual layer are from/to the nominal state $S0$ to/from $S8$, that is a degraded file-system state related to anomalous DNS, HSS, ISCSCF and

SCSCF features behavior. Similarly, the most frequent transitions in the physical layer are from/to the nominal state towards/from $S9$, $S10$, $S14$ and $S15$ which respectively represent CPU, network, memory and disk deviated feature states. Additionally, it is worth mentioning that in the physical layer states describing more than one type of degraded resource at the same time, i.e. $S3$, $S4$ and $S5$, only occur few times, while in the virtual layer none of the degraded states describe a simultaneous degradation for more than one resource type. This means that degraded states perceived in the training phase only represent occasional events that temporarily affect only one type of resource and that do not simultaneously impact the entire infrastructure (probably outliers). Nonetheless, a considerable amount of degraded states concern DNS and HSS features, meaning that AEs struggle in modeling associated metrics likely due to barely predictable behavior. Finally, note that some states appear to describe an amplified deviation of other states, as it happens for $S13$ which adds a further deviation on (net,srv1) with respect to $S12$.

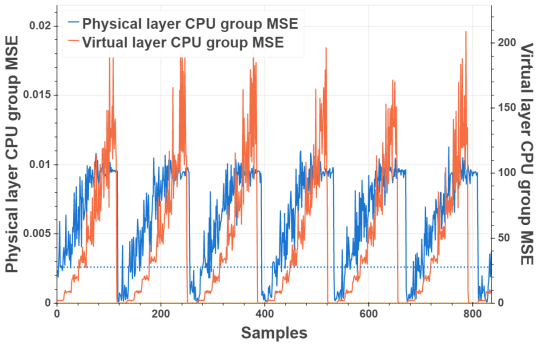
5.3.2 Test on degraded conditions

We evaluate the SYRROCA ability to detect and characterize anomalies under three different anomaly scenarios. For each test case, SYRROCA provides a state graph and a set of different radiographies allowing in-depth root cause analysis. Due to space constraints, for each test case we only show the most relevant representations (omitted ones can be found in [170]).

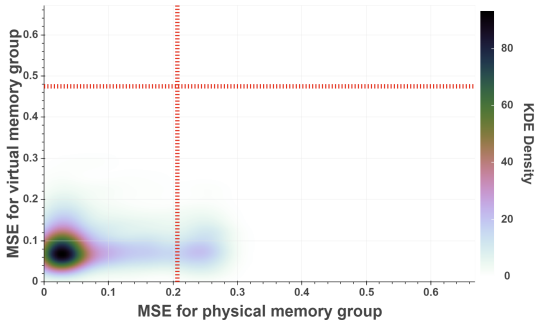
5.3.2.1 CPU stress test

In the *first scenario* we tested how stressing the physical CPU in the PCSCF container is perceived by the autoencoders and how this stress propagates from the physical to the container layer. We injected a persistent physical CPU stress which increases over time in evenly time distributed increments of 10% during one hour across 32 CPUs, starting from 10% up to 80% of single CPU capacity. Each stress cycle is repeated ten times.

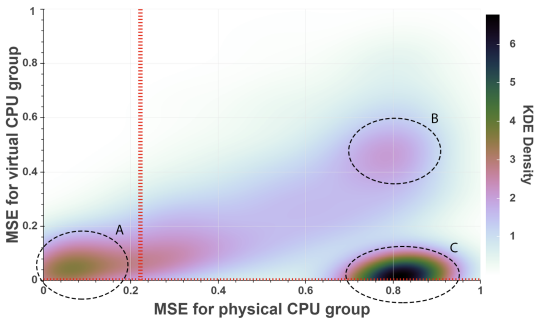
Figure 5.2a depicts the stress pattern perceived by the autoencoders working respectively on the CPU group metrics for the physical layer (in blue) and the virtual layer (in orange). As expected, the most frequently visited state is $S26$ characterized by a deviation on the PCSCF and physical CPU, as seen in Figure 5.2d. Interestingly, SYRROCA detects other less frequent deviations on the CPU group of other containers as a consequence of that deviation, such as the states $S27$, $S28$ and $S29$.



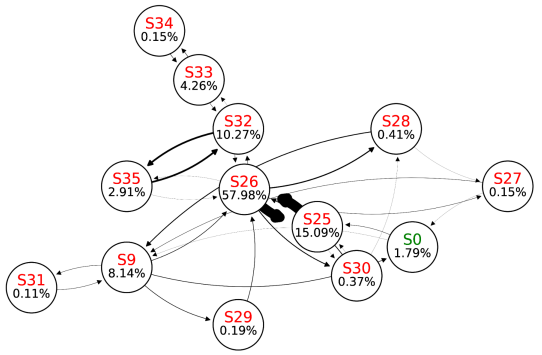
(a) autoencoder MSE for the CPU metrics groups in physical and virtual layers



(b) PCSCF memory stress radiography



(c) PCSCF CPU stress radiography



(d) State graph obtained during the CPU stress test case

Figure 5.2: SYRROCA output visualizations for the CPU stress test case

Indeed, those deviated states were not observed in the training phase, thus they do not refer to known occasional deviations. We can suppose then non-perfect isolation of containers CPU lets the stress inside PCSCF occasionally get through other containers.

The third most frequent state is $S32$, which is the result of a stronger deviation from the state $S26$. This state is characterized by a deviation in the physical memory in addition to that of those deviated resources of state $S26$. This state is likely due to the high amount of memory used by the stressing script. Figure 5.2b depicts the radiography showing how the deviations evolve through time across the 10 stress episodes. Red dotted horizontal and vertical lines represent the autoencoders thresholds for the memory group of virtual and physical layers, respectively. Accordingly, the bottom left rectangle represents the nominal region, where a dark high-density region. For instance, 5.2b shows that most of the time the system is in nominal conditions for both physical and virtual layers metrics. It can be observed that the memory deviation remains contained in the physical layer. This can be evidenced by the horizontal medium density region exceeding only the physical layer threshold but not exceeding

the vertical threshold.

Figure 5.2c instead depicts how the injected anomaly behaves for the CPU metrics group. While region A represents a deviation affecting only the virtual layer (vertical axis threshold is not exceeded), regions B and C represent deviations perceived at both layers. In particular, as depicted in Figure 5.2a, at the beginning of each stress episode, the anomaly is first perceived only at the virtual layer (A). Then as the stress increases, the deviation increases for both physical and virtual layer (B). Finally, when the stress episode terminates, the virtual layer deviation immediately decreases, while at the physical layer a residual deviation is still perceived (C). Note that across the 10 stress episodes time window, region C is the most frequent one: this is probably due to the effect of cumulative metrics which represent the physical CPU load across the last 5 and 10 minutes. Indeed, metrics representing the CPU load for the virtual layer are only instantaneous.

5.3.2.2 Packet-loss injection test

The *second scenario* consists in injecting packet loss to generate calls failures.SIPp allows simulating packet loss by simply blocking outgoing messages or discarding received messages. In particular, we alter the call distribution of March 16, 2020, blocking 50% of INVITE (SIP message) acknowledgments, causing at least 50% of calls to fail.

Looking at Figure 5.3 system state graph, it is interesting to point out that almost all the degraded states only refer to network-related metrics, except $S16$ and $S14$, rarely visited. This means that the injected anomaly does not propagate across different resource types. In particular, the most recurrent deviated state is $S17$ which represents a degradation on the network metrics of the ICSCF and PCSCF containers, thus a virtual-layer only anomaly. Similarly, state $S10$ is only characterized by a deviation on the physical network metrics. As collected virtual layer metrics only belong from vIMS containers, a deviation like the one of state $S10$, which is only detected at the physical layer, describes something related to Kubernetes pods or to any underlying Linux system-level process. Consequently, we can infer that deviated state $S10$ does not depend on the packet loss we imposed. Similarly, as no degraded state is characterized by metrics from both the physical and virtual layer, we can conclude that the detected anomalies never propagate through physical and virtual layers for any kind of resource.

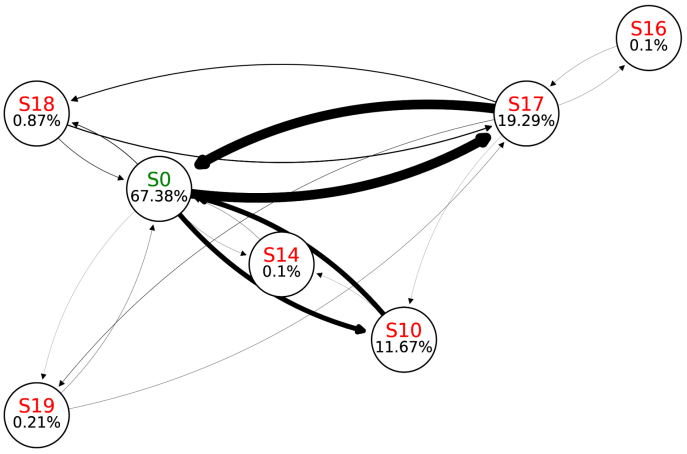


Figure 5.3: State graph obtained during the packet loss test case

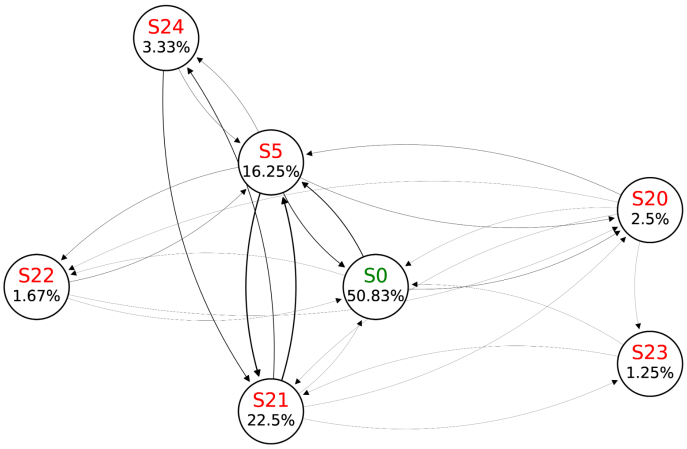


Figure 5.4: State graph obtained during the overload test case

5.3.2.3 Call overload test

The *third scenario* consists in stressing the vIMS core with a call profile exceeding the resources available to the vIMS network functions. To do that, we chose to inject the call distribution of March 22, 2020 from a different LAC than the one used for training, serving more users (Figure 4.4). Actually, even though in our deployment each pod can theoretically use as much memory as the physical server has (best-effort mode), the scripts used to launch IMS services impose a hard-coded memory limit. Nevertheless, we observed that although this script-level limit is not reached, it is possible to overload the vIMS core with a higher amount of traffic as in the selected test LAC. Unfortunately, the SIPp traffic simulation tool showed a limitation on the total number of the simultaneous emulated calls, therefore we could simulate only the first peak of the LAC depicted in Figure 4.4, but in any

case not invalidating the correctness of the test.

The state graph in Figure 5.4 shows that the most frequent degraded state, $S21$, is characterized by deviations on SCSCF network and CPU-related metrics, and only network metrics of the PCSCF IMS virtual function. Similarly, the state $S5$, which is the second most frequent state, points out a deviation on the same resource/container, but on the SCSCF CPU. Furthermore, all the other most frequent states only point out deviations on CPU and/or networks metrics of different vIMS containers. Consequently, we can conclude that the simulated call overload generally produces a deviation on network-related metrics and occasionally generates an additional load on the SCSCF CPU. Contrarily, no deviation is detected in any physical level resource, meaning that the anomaly does not propagate from the virtual to the physical layer.

$S1$	{ (cpu,dns), (cpu,icscf)}
$S2$	{ (cpu,hss)}
$S3$	{ (net,dns), (net,hss), (net,icscf)}
$S4$	{ (net,dns), (net,hss), (net,icscf), (net,scscf)}
$S5$	{ (net,pcscf), (net,scscf)}
$S6$	{ (mem,dns), (mem,hss)}
$S7$	{ (mem,hss), (mem,pcscf), (mem,pcscf), (mem,scscf)}
$S8$	{ (disk,dns), (disk,hss), (disk,pcscf), (disk,scscf)}
$S9$	{ (cpu,srv1)}
$S10$	{ (net,srv1)}
$S11$	{ (cpu,srv1), (net,srv1), (mem,srv1), (disk,srv1)}
$S12$	{ (cpu,srv1), (disk,srv1)}
$S13$	{ (cpu,srv1), (net,srv1), (disk,srv1)}
$S14$	{ (mem,srv1)}
$S15$	{ (disk,srv1)}
$S16$	{ (cpu,scscf), (cpu,pcscf), (cpu,pcscf)}
$S17$	{ (net,pcscf), (net)}
$S18$	{ (net,pcscf), (net,pcscf), (net,scscf)}
$S19$	{ (net,srv1), (net,pcscf), (net,pcscf)}
$S20$	{ (net,pcscf)}
$S21$	{ (cpu,scscf), (net,pcscf), (net,scscf)}

$S22$	{ (net,scscf)}
$S23$	{ (cpu,pcscf), (cpu,scscf), (net,pcscf), (net,scscf)}
$S24$	{ (cpu,dns), (cpu,scscf), (net,pcscf), (net,pcscf)}
$S25$	{ (cpu,pcscf)}
$S26$	{ (cpu,pcscf), (cpu,srv1)}
$S27$	{ (cpu,srv1), (cpu,dns), (cpu,hss), (cpu,icscf), (cpu,pcscf), (cpu,scscf)}
$S28$	{ (cpu,srv1), (cpu,hss), (cpu,pcscf)}
$S29$	{ (cpu,srv1), (cpu,dns), (cpu,pcscf)}
$S30$	{ (cpu,srv1), (cpu,pcscf), (net,srv1)}
$S31$	{ (cpu,srv1), (net,srv1)}
$S32$	{ (cpu,srv1), (cpu,pcscf), (mem,srv1)}
$S33$	{ (cpu,srv1), (mem,srv1)}
$S34$	{ (cpu,srv1), (net,srv1), (mem,srv1)}
$S35$	{ (cpu,pcscf), (mem,srv1)}

Table 5.2: Couples (g,u) of resource group (g) and computing units (u) impacted in each degraded state

Recurrent system states

Degraded states from the training phase, marked with labels from $S1$ to $S15$, refer to anomalous but known-in-advance working conditions that may be found at the test phase as well. In our tests we observed that training degraded state $S5$ occurred also during call overload injection, state $S14$ during packet loss injection and state $S9$ in the CPU stress test case. As $S14$ and $S9$ are characterized by a deviation on server-wide metrics that are influenced by all the running processes, those states are quite likely to occur in a generic test case as autoencoders are not completely robust to outliers or state fluctuations. Furthermore, there could be several reasons for the CPU to deviate from the nominal learned state, which can make a degraded state characterized by only one deviated physical resource quite likely to appear. On the other hand, the state $S5$ characterizes a fine-grain deviation on PCSCF and SCSCF IMS components. The intuition could suggest that this state may be due to a sort of background noise, but this is not actually the case as $S5$ covers 16.25% of the total call overload test case time-stamps.

	CPU		Network		Memory		Disk	
Model	**Virt**	**Phys**	**Virt**	**Phys**	**Virt**	**Phys**	**Virt**	**Phys**
RNN	140	249	160	233	492	359	103	359
LSTM	379	191	269	184	382	275	166	270

Table 5.3: Comparison between RNN and LSTM autoencoders training epochs

5.3.3 Performance comparison

As seen in 3.4.2, autoencoders proved to be effective in detecting and characterizing anomalies thanks to the inherent capability to encode and compress inputs. Likewise, our LSTM NN design demonstrated to be effective in learning long-term sequence correlations and to model complex multivariate sequences. In this section we compare our SYRROCA LSTM-based autoencoder with both Isolation Forest (ISF) - a well-known unsupervised anomaly detection baseline - and RNN-based autoencoders - RNNs are also a good baseline, not storing long-term dependencies.

First, we compared a set of eight LSTM-based autoencoders with its equivalent RNN-based version: both autoencoder neuron architectures are identical, and the difference remains at unit-level (RNN unit and LSTM-unit). We trained both sets of autoencoders with the same dataset setting the maximum number of training epochs at 2000. Looking to use the framework in real environments, we stop the learning process if the MSE remains steady at least 10 epochs.

We can observe that LSTM autoencoders need fewer epochs than an RNN to reach the early stopping condition when analyzing physical layer metrics, as shown in Table 5.3. On the contrary, results show that when working with fewer metrics such as in the virtual layer, RNNs reach the early stopping condition faster.

We compare the performance of LSTM-based autoencoders (AE) with RNN-based autoencoders and a conventional unsupervised Isolation Forest (ISF) to detect deviations on the CPU group at the physical layer when analyzing the *CPU stress test* dataset. We labeled as 1 (positive samples) the anomalous samples while nominal ones are labeled with 0 (negative samples). The obtained results are summarized in Table 5.4. On one hand, ISF could not correctly detect most anomalous samples as it classifies most of the samples as nominal giving a perfect precision score but very low accuracy and recall (0.007% of anomalies are detected). A well-performing anomaly detection algorithm should

	Accuracy	Recall	Precision	F1-Score	F2-Score
ISF	0.145	0.007	1	0.014	0.089
RNN AE	0.166	0.053	0.714	0.098	0.065
LSTM AE	**0.826**	**0.866**	**0.927**	**0.895**	**0.877**

Table 5.4: Anomaly detection evaluation metrics

be able to catch all the anomalies; in our experiments we labeled anomalies with 1, thus a properly working anomaly detection algorithm should have a Recall value close to 1. In order to account also for its ability to not label as an anomaly (positive) a sample which is nominal (negative), we computed the F-2 score [194]. In fact, based on the F-beta score that is the weighted harmonic average of precision and recall, the F-2 score lowers the importance of precision and doubles the importance of recall. As depicted in Table 5.4, RNN-based autoencoders slightly improve F-2 score with respect to IF, mainly due to a slightly increased recall score. However, LSTM-based autoencoders clearly outperform both ISF and RNN-based autoencoders as they maintain a high precision score as long as greatly increasing the recall score.

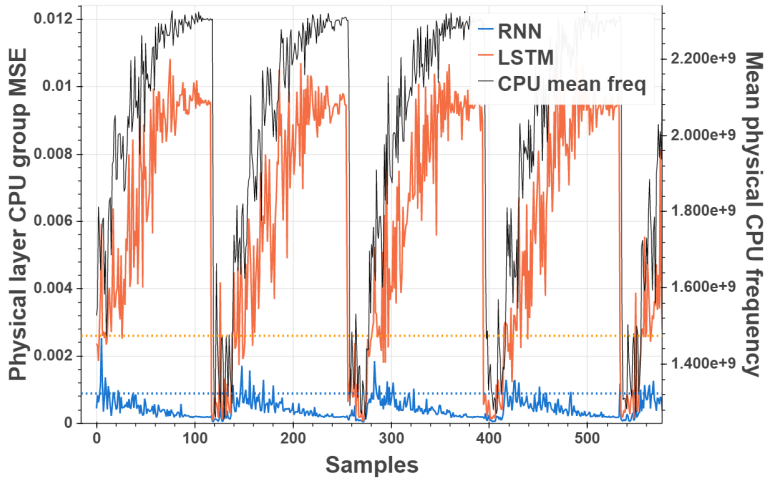


Figure 5.5: RNN and LSTM based autoencoders reconstruction error compared to mean physical CPU frequency

Finally, Figure 5.5 depicts RNN and LSTM autoencoders output reconstruction errors for the first 4 cycles of the CPU stress test case. While the LSTM-based autoencoders produce an MSE that closely mimics the average CPU frequency, the RNN-based autoencoders can only detect the deviation at the very first start, but it fails in tracking the CPU stress during the rest of the cycle as its reconstruction

error systematically tends to plummet below the threshold in contrast to LSTM-based approach. In conclusion, the RNN approach would be good enough to detect the anomaly at its very start, which is useful to immediately react; however, in our case, to both detect and characterize the anomaly along its entire life, we need also to track it down through time, therefore LSTM-based is substantially more appropriate as it allows meeting this requirement.

5.4 Conclusion

In this chapter we proposed and experimentally evaluated a methodology to assess the system state of a virtualized network service through the detection of deviations from a nominal state, learned from known history, for each type of resource and layer. The described state characterization methodology is visualized through directed state graphs which summarize the evolution of the system across the nominal and the degraded states. The state assessment pinpoints the most deviated resource group(s) and computing unit(s) for a given anomaly. We showed how radiographies and system state assessment allow a comprehensive view of the resiliency state of the virtualized platform.

We tested SYRROCA for a vIMS environment with the goal to detect and characterize deviations and degradation on compute units and network resources in order to suggest the appropriate orchestration actions. Experimental results show that LSTM autoencoders are able to model the nominal state of a virtualized platform by analyzing a multivariate time series composed of metrics collected by a monitoring system. We showed how to exploit the MSE of AEs to characterize system states that deviate from the nominal working conditions. Furthermore, we compared SYRROCA LSTM based autoencoders with a baseline RNN and an isolation forest (ISF) approach, proving that RNN autoencoders improve both the recall and the F2-scores compared to the ISFs, which fail to extract any information on unlabeled data. We also showed that LSTM further increased the autoencoder performance compared to RNN thanks to their ability to store long-term dependencies.

The state graph analysis methodology we proposed is meant to fuel a decision engine within a closed-loop automation system: as each degraded state is characterized by the set of most deviated metrics, remediation policies can be tailored to these deviations. In the following chapter, we explore a possible formalization of such a closed-loop automation engine.

Chapter 6

Automated reconfiguration method

Content

In the previous chapters we presented the SYRROCA framework which covers the sense phase of the network automation loop by detecting and characterizing anomalies across physical and virtual layers. This phase involves learning the nominal state conditions to later identify and characterize which network resources (resource group, computing unit) and to what extent are deviated. In the following section, we present the architecture of an automated recovery methodology able to compensate for the deviation leveraging SYRROCA system state assessment. The proposed approach is based on a Reinforcement Learning (RL) algorithm which learns how to select the most appropriated remediation actions targeting at those resources the state characterization points as most deviated ones from the nominal working conditions. We will show how SYRROCA enriched with a reputation model and an RL-based agent complete the cognitive closed-loop automation framework. This chapter mainly describes the patent [15] and its possible system design.

6.1 Anomaly remediation

A cognitive loop allows to cover from the sensing of the network state to the act phase, that is the application of a set of actions (policies) to recover from a malfunction. Those policies are dynamically learned aiming to achieve an abstract goal dictated by the business or user technical requirements such as maintaining a certain Quality of Service (QoS) to fulfill a Service Level Agreement (SLA). Furthermore, the cognition should be implemented through an algorithm that improves its performance through experience gained over a period of time [67]. Regardless of the implementation, not only the process needs to observe the network state transition resulting from the remediation actions, but it also has to learn from that result. In the event of any degradation due to an incorrectly chosen remediation action, the process should receive negative feedback discouraging it to take the same action given the same conditions that led to that new state. Conversely, if the remediation action correctly mitigates the detected anomaly, positive feedback should reward the process.

When implementing a closed-loop for cognitive resiliency management of softwarized networks, softwarization provides natural hooks for the cognition plan. In fact, as we already pointed out in past chapters, virtualized networks rely on several active entities which provide the configuration flexibility and scalability network operator need. For example, the Kubernetes container Orchestrator 3.1.2 continuously tries to enforce the desired state; to do so it can execute actions like horizontal/vertical scaling or pod migration. Since each entity is only able to operate on a subset of the virtualized resources, entities are intrinsically organized in a hierarchy that reflects the layered structure of softwarized networks. Thereby, remediation actions may also target different layers of the infrastructure and their realization could entail lower layer sub-actions. For instance, when a Kubernetes orchestrator performs a container migration to a new compute node, other lower-level sub-actions may be required, like for example the instantiation or the Docker container and the computation of a new path to it performed by an SDN controller. Furthermore, each action could potentially be performed by different entities spanning different levels of the virtualized platform stack (e.g. SDN, NFVO, orchestrator, etc...). Due to the network complexity, it is not possible to anticipate the side effects of remediation actions across all layers.

Reputation models have as main goal to drive system interactions based on previous experience [145]. This is very relevant to the multi-partner and multi-layer softwarized systems. Indeed,

several new business models arise in 5G systems, as pointed by NGMN (Next Generation Mobile Networks) consortium in [195] such as "operator offer enriched by partner" where an operator can offer an enriched service with additional functionalities developed by a third party. In this context, a reputation model can foster trustful collaboration among different management agents in the network, where each agent has the freedom to choose which to cooperate with based on a historical record of past collaborations. Although the NFV framework analyses resilience aspects of network services implemented with VNFs from the scope of reliability, it does not answer two paramount questions:

- how MANO entities coordinate in case of mismanagement of one of those entities;
- how the decisions taken by MANO entities may impact the infrastructure as well as the services.

Indeed, within the NFV architecture, alarm correlation is proven to be challenging, as there are several resources to be managed by different MANO entities such as VIM (Virtual Infrastructure Manager) at the infrastructure level, the VNFM (VNF Manager) at the level of a VNF (Virtual Network Function), or at network service level by the NFVO (NFV Orchestrator). Furthermore, when SDN controllers are used to establishing interconnections among VNFs, they may also react to faults by rerouting traffic among VNFs. This scenario seems to make difficult action coordination among agents because each different MANO entity sees different expressions of the same anomaly and may try to act upon the problem, which may worsen the original fault consequences itself. Each entity may decide to act unilaterally (in case local information is enough to make an effective decision) or to delegate the decision by triggering alarms to upper layers entities.

6.2 RL-based remediation policy learning

Learning from past experience is what drives the cognition process every human being rely on to learn across their life. The reinforcement learning paradigm is inspired by this process and it thereby seems a good fit for whatever cognition loop. Nonetheless, the adoption of RL algorithms as the implementation of a resiliency management cognition cycle, poses several challenges, such as:

- Given the complexity of virtualized networks we already pointed out in this dissertation, the resiliency system state assessment is a very brought concept that can be modeled in different ways;

- The resilience notion is a system-wide property, which should be translated into a goal state towards which the RL algorithm tries to make the agent converge;
- The reward function design should produce feedback on the quality of the actions. It should assess whether the system transitioned towards an *improved* state or a *worsened* one. In both cases, it should also quantify the improvement, such for example the *distance* from the goal state.

As introduced in section 3.4.4, regardless of the RL algorithm used to produce and update the agent policy, the RL agent needs to retrieve from the environment its actual state S_t, the list of possible actions in that state $A(S_t)$, the state towards which it moves performing the selected action $a \in A(S_t)$ and the associate reward r_{t+1}. In the previous chapter, we showed how leveraging the SYRROCA framework it is possible to obtain the current system state and its characterization as deviated state from the nominal learned working conditions. We also showed how each deviation can be associated with a particular class of deviations, containing anomalous states of the same *type*, i.e which are characterized by a relevant deviation on the same set of couples (g, u). In this chapter, we propose to leverage these features to formalize the RL problem as follows.

Table 6.1 summarizes the notation we use throughout the following formalization:

G	Set of considered resource groups
L	Set of considered layers
$\Omega(S_t)$	Set of possible remediation intents ω on state S_t
$E = \{e_1, ..., e_i, ..., e_z\}$	Set of considered z active entities
$\mathcal{A}_t^{e_i} = \{\omega_j\}$	Set of all intents $\omega \in \pi_t$ the entity e_i took part in
$\Psi_t^{l,g}$	Radiography between layers l and $l+1$ of resource group g
$C_t^{l,g}$	Center of the high-density region of $\Psi_t^{l,g}$
w	Radiography sliding window width
$\hat{x}_{l,g}$ and $\hat{y}_{l,g}$	base vectors of $\Psi_t^{l,g}$ Euclidean space
S_t	SYRROCA system state at time-step t
ϵ	number of time-step required for intent execution
$\mathbf{d}_t^{l,g}$	Vector from $\Psi_t^{l,g}$ axes origin to $C_t^{l,g}$
$\mathbf{n}^{l,g}$	Difference vector of $\mathbf{d}_t^{l,g}$ and $\mathbf{d}_{t+\epsilon}^{l,g}$
$\hat{x}'_{l,g}$ and $\hat{y}'_{l,g}$	Base vectors of the Euclidean sub-space centered at $C_t^{l,g}$

$\mathbf{n}^{l,g}_{\hat{x}'}$	$\mathbf{n}^{l,g}$ projection on $\hat{x}'_{l,g}$
$R_{t+\epsilon}(S_t, \omega)$	RL agent reward for intent ι from state S_t
$\pi_t = \{\omega_1, ..., \omega_t\}$	Policy the RL agent performed at time t

Table 6.1: Table of notations

6.2.1 Agent

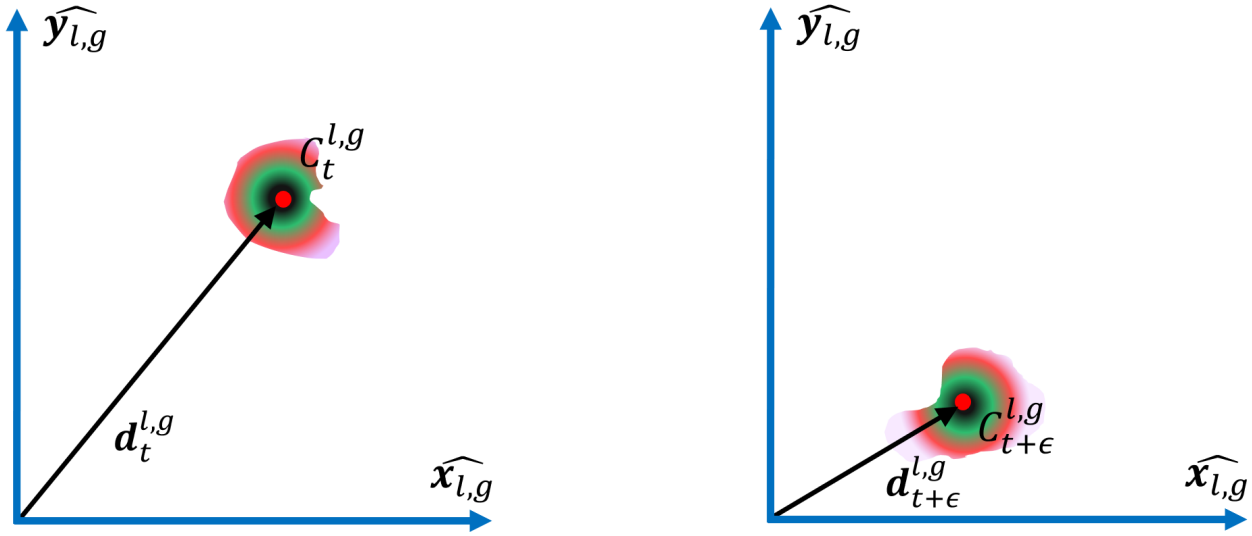
SDN/NFV virtualized infrastructures encompass several resource controllers, i.e. where each controller is able to act on specific resources (i.e. link, computing, radio) composing the end-to-end systems. Those resource controllers provide a perfect hook for an RL agent. Even though they are in general scoped to act on a sub-set of the system resources and not all the entities are always involved in the anomaly recovery process, we propose to model all resource controller as a unique RL agent. Consequently, each RL-level action is rendered by one or more resource controller-level actions, eventually executed by distinct resource controllers. To avoid confusion, from now on we denote RL agent actions as "*intent*"ω, while *action* a refers to resource controllers-level actions. In the state of the art, an intent is a declaration of high-level operational goals that are to be achieved by the network, without specifying how to achieve them and holds for the network as a whole, not individual devices [39].

System state S_t

As stated in 5.3, the SYRROCA framework is able to assess whether the system is in its Nominal state or in a degraded one. In the latter case, the framework characterizes the state with reference to the set of most deviated (g, u) couples. Thereby system states are defined as:

$$S_t = \begin{cases} Nominal & \text{if } MSE^{l,g}(t) < T^{l,g} \ \forall g \in G \wedge \forall\ l \in L \\ \{(g_1, u_1), ..., (g_k, u_k)\} & \text{if } \exists\ g \in G \wedge \exists\ l \in L \mid MSE^{l,g}(t) \geq T^{l,g} \end{cases} \tag{6.1}$$

Note that states as defined in 6.1 encompass all the groups of resources and all the layers to produce an overall view of the system resiliency state. As we will show later, this is particularly useful to narrow down candidate remediation actions.



(a) Radiography $\Psi_t^{l,g}$ computed on a sliding-window $[t-w;t]$

(b) Radiography $\Psi_{t+\epsilon}^{l,g}$ computed on a sliding-window $[t+\epsilon-w;t+\epsilon]$

Figure 6.1: Reward computation on radiography

6.2.2 Reward

While SYRROCA state graph is useful to characterize the system state with reference to the most deviated resources and computation units, it does not provide any insight about the deviation extent. On the contrary, in section 5.3.2.1 we showed that the MSE proportionally increases with the anomaly intensity, which in turn makes the radiographies high-density region move across the resiliency state space. Thereby, we propose to leverage the high-density region position within the state to estimate the deviation intensity as the distance of its center from the axis origin, which represents the optimal working condition. Indeed, even if AEs MSE is not likely to go down to exactly 0, the smaller the reconstruction error the closer the actual state is to the nominal operating conditions.

Let us take a virtualized system made up of L layers and G resource groups and let us suppose that $C_t^{l,g}$ is the center of the higher density region of the radiography $\Psi_t^{l,g}$ between layers l and $l+1$, for the group of resource g for the deviation d_t occurring at time-step t. The radiography is computed on a sliding-window $[t-w;t]$. Figure 6.1a shows a simplified version of a sample radiography where we omitted low density regions and colors are chosen just to recall real radiography colors. $\hat{\mathbf{x}}_{l,g}$ and $\hat{\mathbf{y}}_{l,g}$ are the base vectors of the Euclidean space associated with the radiography $\Psi_t^{l,g}$. For example, given the radiography between the physical and the virtual layers for the CPU group, $\hat{\mathbf{x}}_{l,g}$ is the

base vector representing the physical layer reconstruction error $MSE^{phy,CPU}$ dimension and $\hat{\mathbf{y}}_{l,g}$ the base vector representing the virtual layer reconstruction error $MSE^{virt,CPU}$ dimension. Called S_t the system state retrieved from the SYRROCA system state graph, let us then suppose the RL agent selects a remediation intent $\omega \in \Omega(S_t)$ which makes the system transitioning to $S_{t+\epsilon}$, where ϵ accounts for the execution delay of the intent. Thereby, $C^{l,g}_{t+\epsilon}$ is the new center of the higher density region in the radiography $\Psi^{l,g}_{t+\epsilon}$ between layers l and $l+1$ for the time-window $[t+\epsilon-w; t+\epsilon]$.

Let us then call $\mathbf{d}^{l,g}_{t+\epsilon}$ and $\mathbf{d}^{l,g}_{t}$ the vectors from the origin of the radiography Euclidean plan to, respectively, $C^{l,g}_{t}$ and $C^{l,g}_{t+\epsilon}$. Note that as each $MSE^{l,g}$ could have widely different magnitudes, each $\mathbf{d}^{l,g}_{t+\epsilon}$ may have widely different length. Thus, in order to fairly compare each radiography contribution to the reward computation, we need to re-scale on the same scale all the $MSE^{l,g}$. To do that, we can for example take the wider range $MSE^{\bar{l},\bar{g}}$ and normalize all the other $MSE^{l,g}$ its range.

We propose then to compute the reward $R_{t+\epsilon}$ of the intent $\omega \in \Omega(S_t)$ which makes the system transitioning from S_t to $S_{t+\epsilon}$ as:

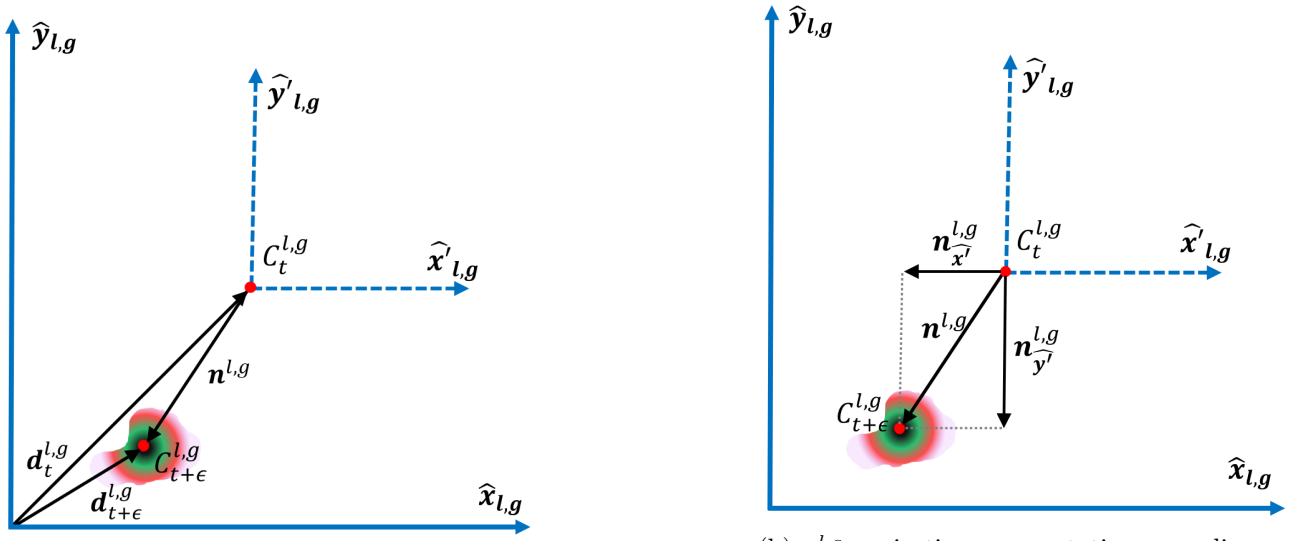
$$R_{t+\epsilon}(S_t, \omega) = \sum_{l \in L} \sum_{g \in G} \frac{1}{||\mathbf{d}^{l,g}_{t+\epsilon}||} \tag{6.2}$$

According to 6.2, the reward is computed as the sum of all contributions across each radiography, where each contribution is inversely proportional to the module of the vector $\mathbf{d}^{l,g}_{t+\epsilon}$, which is the distance from the axis origin (i.e. the ideal nominal working conditions) of the new center of high-density. Therefore, the reward associated with an intent ω is always positive and increases the closer the centers of maximum density of each radiography are to their respective axes origins.

Nonetheless, this reward definition does not account for state S_t positions of radiographies high-density region centers, which makes the reward not completely well defined.

As depicted in figure 6.2a, let us then call $\mathbf{n}^{l,g}$ the difference vector between $\mathbf{d}^{l,g}_{t+\epsilon}$ and $\mathbf{d}^{l,g}_{t}$ obtained with the parallelogram low. According to this low $\mathbf{n}^{l,g}$ is centered on $\mathbf{d}^{l,g}_{t}$ tip and ends on $\mathbf{d}^{l,g}_{t+\epsilon}$ tip. Let us call $\hat{\mathbf{x}}'_{l,g}$ and $\hat{\mathbf{y}}'_{l,g}$ the base unit vectors of the Euclidean state space centered in $C^{l,g}_{t}$ and with both axis parallel to the ones of $\hat{\mathbf{x}}_{l,g}$ and $\hat{\mathbf{y}}_{l,g}$. Let us also call $\mathbf{n}^{l,g}_{\hat{x}'}$ and $\mathbf{n}^{l,g}_{\hat{y}'}$ the $\mathbf{n}^{l,g}$ projection on $\hat{x}'_{l,g}$ and $\hat{y}'_{l,g}$ (figure 6.2b). We then define two coefficient $m^{l,g}_{\hat{x}'}$ and $m^{l,g}_{\hat{y}'}$ as:

$$m^{l,g}_{\hat{x}'} = \begin{cases} -\alpha & \text{if } \mathbf{n}^{l,g}_{\hat{x}'} > 0 \\ +\alpha & \text{if } \mathbf{n}^{l,g}_{\hat{x}'} < 0 \\ 0 & \text{if } \mathbf{n}^{l,g}_{\hat{x}'} = 0 \end{cases} \tag{6.3}$$



(a) $n^{l,g}$ computation on radiography $\Psi_{t+\epsilon}^{l,g}$

(b) $n^{l,g}$ projections computation on radiography $\Psi_{t+\epsilon}^{l,g}$

Figure 6.2: Reward computation on radiography

$$m_{\hat{y'}}^{l,g} = \begin{cases} -\beta & \text{if } \mathbf{n}_{\hat{y'}}^{l,g} > 0 \\ +\beta & \text{if } \mathbf{n}_{\hat{y'}}^{l,g} < 0 \\ 0 & \text{if } n_{\hat{y'}}^{l,g} = 0 \end{cases} \tag{6.4}$$

where $\alpha < \beta$ and $\alpha \in \mathbb{R}^+$ and $\beta \in \mathbb{R}^+$. When summed up, those coefficients account for the the sense and the directions of the movement associated with ω. Indeed, consider $\alpha = 1$ and $\beta = 4$, figure 6.3 depicts the possible values of $m_{\hat{x'}}^{l,g} + m_{\hat{y'}}^{l,g}$. Note that the condition $\alpha < \beta$ makes the agent prefer improving both layer conditions when possible but giving a greater reward to a movement towards an improvement over the y axis. Indeed, whatever layers we consider, layer l provides a service to layer $l+1$ and in a service-centric logic it is preferable to keep upper layers service in the best state possible.

Accordingly, the reward is properly defined as:

$$R_{t+\epsilon}(S_t, \omega) = \sum_{l \in L} \sum_{g \in G} \frac{1}{||\mathbf{d}_{t+\epsilon}^{l,g}||} \cdot (m_{\hat{x'}}^{l,g} + m_{\hat{y'}}^{l,g}) \tag{6.5}$$

where each radiography contribution will both take into account the improvement or the worsening of state $S_{t+\epsilon}$ with respect to state S_t and the distance from the goal state. Therefore, a movement over the negative sense of both direction results in a positive reward, while a movement along the positive sense of both directions results in a negative reward (i.e. punishment.). Likewise, a movement towards

the first and second sectors of the Euclidean plan gives a negative reward, while the third and fourth sectors results in a positive reward.

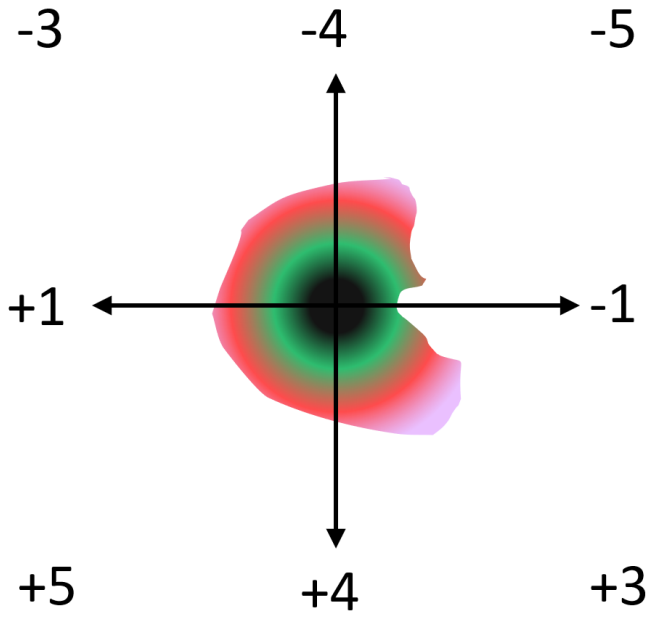


Figure 6.3: Possible values for coefficient $m^{l,g}_{\hat{x}'} + m^{l,g}_{\hat{y}'}$

Finally, it is worth noticing that as each radiography shares one axis with another radiography, radiography contributions for a given g are not completely unrelated. For example, a leftward movement on radiography corresponds to a downward movement on the upper layer one.

6.2.3 Actions

As already pointed out, the origin of radiographies axis represents an ideal situation where the current state is perfectly aligned with the one learned by the AEs. However, in real settings, this case never occurs. Indeed, AEs rely on thresholds on reconstruction MSE to tag a sample as anomaly or nominal. The nominal learned state is in fact characterized by the compact representation the AEs extract from the training set, not by single and specific values for each metric. Thereby, thresholds can be used to identify the maximum value of the MSE of each group of resources and for each level for which the state of the system can be considered nominal with respect to the metrics composing the MSE.

As depicted in figure 6.4 thresholds define a sort of tolerance region, where the system state can

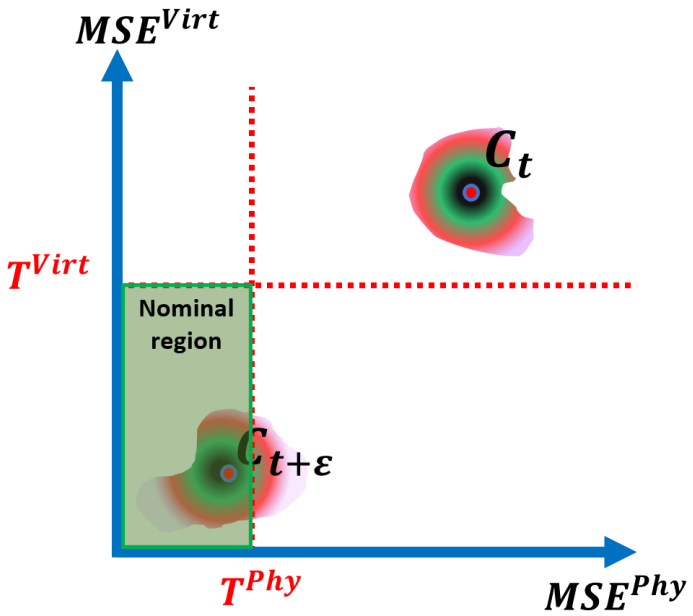
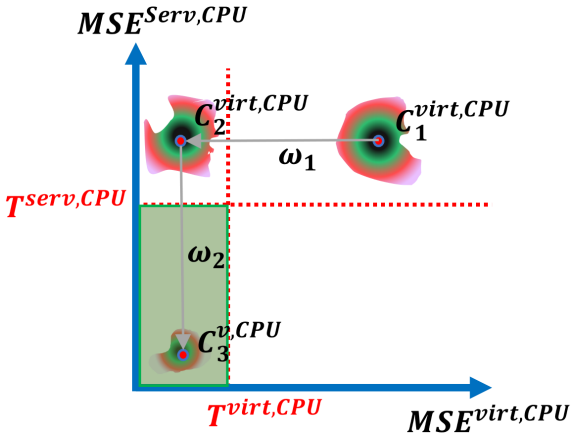


Figure 6.4: Nominal region within a radiography of physical and virtual layers.

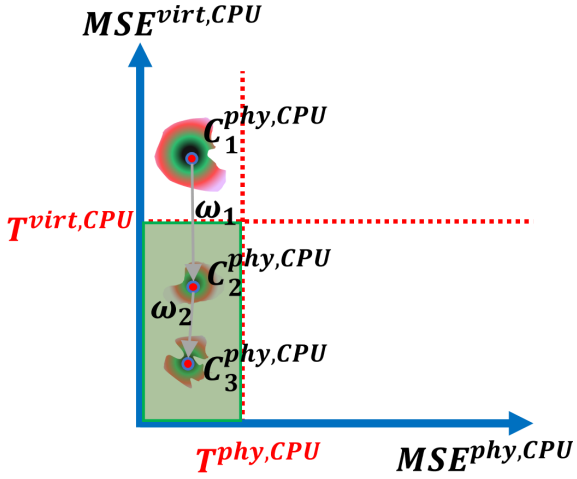
still be considered nominal. In the example of figure 6.4 the radiography between physical and virtual layers highlights a deviation on both layers as the high-density region C_t exceeds both the physical layer threshold T^{phy} and the virtual layer one T^{virt}. Then at time-step $t+1$ a remediation intent pushes the high-density region towards the bottom left corner: even if the high-density region is not completely within both thresholds, its center falls into both. In that case, other remediation intents could further approach the high-density region to the axis origin, but those improvements would be pointless as the MSEs for both groups and layers are already within the nominal range. Hence, when deciding which remediation intent to put in place, those performed by entities acting on physical and virtual layers should be discarded.

Accordingly, we propose to narrow down the set $\mathcal{I}(S_t)$ of possible orchestration intents from where the RL agent has to choose the remediation intent according to the system state characterization proposed in 6.2. This characterization is indeed particularly suited as it clearly points out the resources and the computation units a remediation action should act on, rather than the specific metric on which an action could not directly act on. Accordingly, given the deviated state $S_t = \{(g_1, u_1), .., (g_k, u_k)\}$, $\Omega(S_t)$ is populated by the network administrator with intents that only directly affects one, a subset, or all the computation units and resources listed in S_t. When populating $\Omega(S_t)$ with recovery intents, the network administrator expertise is crucial to avoid listing pointless intents so that to maximize

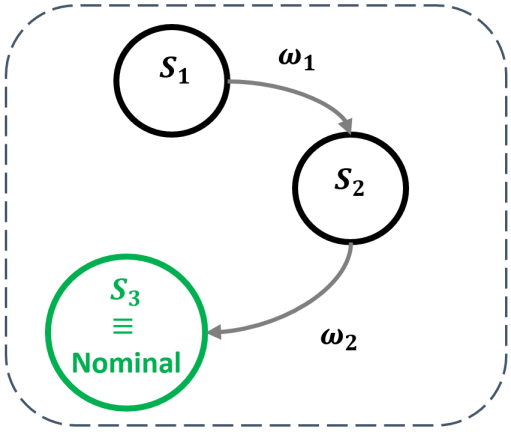
the RL agent performance. Nonetheless, the network administrator should provide the RL agent with a sufficient degree of freedom in order to explore non-obvious intents.



(a) Virtual-Service layers radiography



(b) Physical-Virtual layers radiography



S_1: {[call, rtt], [cpu, srv1][cpu, cont5]}
S_2: {[call, rtt] }
S_3: Nominal

(c) SYRROCA system state graph

Figure 6.5: Example radiographies and system state evolution for a given group of resource.

Consider for instance the example of figure 6.5 where the system starts in the deviated state S_1. As it can be deduced from high-density region centers $C_1^{virt,CPU}$ and $C_1^{phy,CPU}$ of radiographies 6.5a and 6.5b, S_1 is characterized by a deviation on the service, the virtual and the physical layers, which is clearly summarized by the S_1 characterization obtained through the SYRROCA framework. In that case, $\Omega(S_t)$ could contains several remediation intents like for example $\{scale\_container\_5, move\_container\_5,\ increase\_container\_5\_cpu,\ move\_others\_containers\}$. Sup-

pose that *move_container_5* intent is selected and that the container is moved to *server_2*. This could then trigger, for instance, a path re-computation for the SDN controller which would also re-configure some vSwitches to correctly route the traffic towards $container_5$ now located at *server_2*. Imagine that this intent is called I_1 and that it brings the system to the new state S_2, where $C_2^{phy,CPU}$ high-density region center fall into the nominal region and $C_2^{virt,CPU}$ highlight a persisting deviation on the service layer. Looking at S_2 characterization, is it now clear that the following remediation intent, should try to ameliorate the service quality.

6.3 Formalizing resiliency management through the notion of reputation

Even though we model all the active entities in a softwarized platform as one unique RL agent, each remediation intent is in general rendered as at least one specific entity action. As showed in figure 6.5, for a given deviated state there cloud be a set of RL agent remediation intents related to the state characterization, and each RL action may be performed by one/more entity level actions, eventually executed by distinct entities. Thereby, each entity during the execution of a remediation policy, i.e. multiple recovery intents, may contribute to all positive, all negative or mixed rewarded intents. We show hereafter how we propose to leverage all of this to compute a reputation value for each entity. The reputation value is intended as a quantification of the contribution of each entity to the resilience of the softwarized system. Therefore, in order to compare each entity contribution, entity reputation at time t is computed as a real number $Rep_t^{e_i} \in [-1, 1]$, where values close to -1 suggest a poor contribution to resilience while values close to 1 highlight a good contribution.

Considering an entity e_i among the set of all entities involved in a softwarized network system $E = \{e_1, ..., e_i, ..., e_z\}$. For each policy $\pi_t = \{\omega_1, ..., \omega_t\}$, we can build the set $\mathcal{A}_t^{e_i} = \{\omega_j\}$ of all intents $\omega_j \in \pi_t$ the entity e_i takes part in. Note that a intent ω_j could be rendered as more than one entity action, thus ω_j could belong to more than one $\mathcal{A}_t^{e_i}$. We propose to define the entity reputation $Rep_t^{e_i}$ as:

$$Rep_t^{e_i} = \chi\Big(\sum_{\omega_j \in \mathcal{A}_t^{e_i}} \zeta(R_{t+\epsilon}(\cdot, \omega_j))\Big) \tag{6.6}$$

where $\zeta : \mathbb{R} \mapsto \{+\mu, -\lambda\}$ function is defined as:

$$\zeta(x) = \begin{cases} +\mu & \text{if } R_{t+\epsilon}(\cdot, \omega_j) > 0 \\ -\lambda & \text{if } R_{t+\epsilon}(\cdot, \omega_j) < 0 \end{cases} \tag{6.7}$$

and $\chi(x) : \mathbb{R} \mapsto [-1, 1]$ as:

$$\chi(x) = \begin{cases} -1 & \text{if } x < 0 \\ 0 & \text{if } x = 0 \\ +1 & \text{if } x > 0 \\ x & otherwise \end{cases} \tag{6.8}$$

Accordingly, after each intent execution, resource controllers reputation can either increment by $+\mu$ or decrement by λ. Choosing a value of $\lambda >> \mu | \lambda, \mu \in (0, 1)$, it is possible to make the reputation suddenly drop when a wrong remediation intent is performed (negative reward) while slightly increasing if a positive rewarded intent is executed. χ, instead, is a step function we use to clip the reputation between -1 and 1. Indeed, without the clipping, the reputation could grow, negatively or positively, so much that at a given point the updates would turn out useless preventing also a comparison among entities reputation. According to section 3.3, $Rep_t^{e_i}$ is a properly defined reputation values as it is computed as an aggregation of trust values: trust depends in fact on punctual interactions, that in our case are each action the entity performs for each intent ω_j and the reputation is thus a global view of the entity *goodness* across different actions. Consequently, the reputation value can be used by the RL agent as a discriminator value to further guide the decision-making process. Indeed, a low

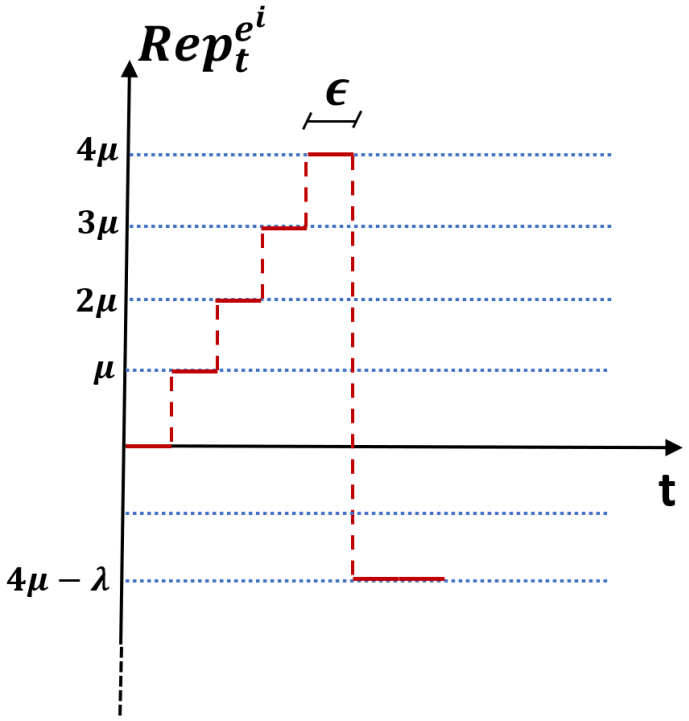


Figure 6.6: Reputation computation example

reputation score indicates a recent history of sub-optimal actions which suggests the entity should be prevented to take any other action on the system, waiting for a check on its functioning. Figure 6.6 visualizes an example of $\mathcal{R}_t^{e_i}$ computation where the reputation increases for five consecutive intents ω but suddenly drops due to a wrong remediation intent.

6.4 Conclusion

In this chapter we outlined an analytical development of the patent [15]. We showed how Reinforcement Learning naturally fulfills the requirements for a cognitive closed-loop automation mechanism for resiliency management of virtualized networks. We modeled all NFV resource controllers as one unique RL agent, which enforces high-level operational goals, intents, to recover from anomalous state. The RL agent current state is obtained through the SYRROCA systems state assessment outlined in the previous chapter. Intents are rendered with one or more resource controller-level actions; a list of possible recovery intents are statically assigned to each deviated state according to its characterization. The agent reward is computed on the SYRROCA radiographies and accounts for the distance of the current state from the Goal state, identified as the origin of radiography euclidean space. We finally proposed to quantify each resource controller contribution to the system resilience through a reputation value.

As the vIMS platform we previously introduced does not allow active recovery intents realization, we are currently developing an ad-hoc 5g-core-based platform. As future work, we will test and validate both the proposed RL formalization and the reputation system.

Chapter 7

Concluding remarks and perspectives

In this dissertation we presented and experimentally evaluated a completed autonomic control loop for virtualized networks resilience management.

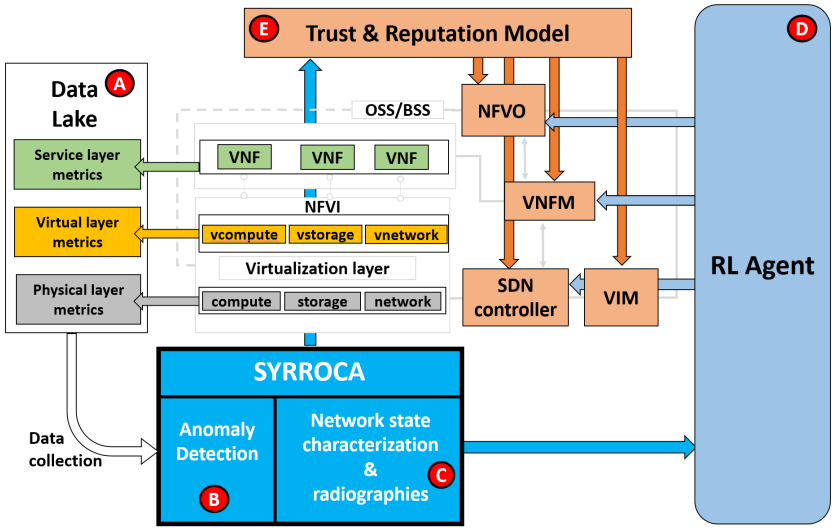


Figure 7.1: SYRROCA closed loop automation on ETSI NFV architecture

Figure 7.1 summarizes our proposal as different blocks applied to the NFV functional architecture. The core of our proposal is the SYRROCA framework, which is an LSTM-autoencoder based approach to detect (B) and characterize (C) anomalies from a multivariate time series composed of hundreds of metrics collected at physical and virtualization levels (A) of a softwarized platform supporting a virtualized network service. We showed how raw metrics can be aggregated and pre-processed to constitute a training dataset on which LSTM Deep autoencoders can learn a compact representation of the nominal working conditions of the softwarized system. Indeed, thanks to the compression proper-

ties of autoencoders the high dimensional input time series is compressed into a lower dimensionality sub-space (the latent space) where anomalous samples look significantly different from nominal ones. To improve the autoencoder performance, we further split the input dataset into per-resource-type groups of metrics and we feed each group to a dedicated autoencoder. Considering CPU, network, memory and disk resource types, we have a set of four AEs for the physical layer and four more for the virtualization layer. To properly analyze both monotonically increasing and arbitrarily shaped metrics, we only keep increments of the former while normalizing to a uniform range both categories of metrics. We select LSTM RNN as base autoencoder cells since they allow handling of arbitrarily long time series not losing memory of samples far in the past as it happens in standard RNNs. We compared SYRROCA LSTM based autoencoders with a baseline RNN and an isolation forest (ISF) approach, proving that RNN autoencoders improve both the recall and the F2-scores compared to the ISFs, which fail to extract any information on unlabeled data. We also showed that LSTM further increased the autoencoder performance compared to RNN thanks to their ability to store long-term dependencies.

Then, we showed how to exploit the MSE of AEs to characterize detected anomalies both through the set of most deviated metrics and with the radiography novel visualization. In particular, we demonstrated how autoencoders MSE is correlated with the anomaly intensity which enables our framework to assess anomaly severity. We then explained how radiographies can visualize network anomalies on a bi-dimensional euclidean space where the density of the bi-variate functions $f(MSE^l, g, MSE^{l+1,g})$ of two consecutive layers AE mean square errors, spots the most recurrent system state under the considered time-window. We presented two versions of a set of $(L-1) \times G$ radiographies for G groups of resources of a system composed of L layers. The sliding-window radiography version highlights anomaly evolution through time while the growing-window version shows anomaly propagation across consecutive layers. In both versions, AEs thresholds identify a rectangular area that corresponds to the normal functioning conditions.

Both the Autoencoder-based anomaly detection and system state characterization were experimentally evaluated on a vIMS platform managed by Kubernetes. We injected realistic traffic based on real call profiles extracted from a given LAC (Location Area Code) from the 3G Orange network. We also injected three anomalies types to evaluate SYRROCA detection and characterization performance. The generated dataset is available at [170].

We then proposed a methodology to assess softwarized system running state (C) basing on the anomaly characterization produced at step B. The methodology produces a state graph that is meant to fuel a decision engine within a closed-loop automation system: as each degraded state is characterized by the set of most deviated metrics, orchestrations policies can be tailored to these deviations. Following autonomic computing guidelines, we proposed a Reinforcement Learning based remediation mechanism (B) which aims at recovering the system state back to the nominal learned working conditions. RL, indeed, perfectly matches the cognitive process that an autonomous network should implement, and the various softwarized entities that make up the SDN/NFV stack constitute an ideal hook for the cognitive process. Accordingly, the RL agent embeds all the system active entities in a higher level agent which manages the softwarized system resilience through high-level intents, implemented with one/several entities actions. RL agent state is obtained through SYRROCA state assessment block C, which also computes intents reward from sliding-window radiographies as a quality rating for the intent. Specifically, the reward is obtained as the sum of contributions of each of the $(L-1) \times G$ radiographies. Each contribution is inversely proportional to the distance of the high-density region from the origin of radiography Euclidean space and directly proportional to a factor $m^{l,g}_{\hat{x}'} + m^{l,g}_{\hat{y}'}$. While the distance accounts for intent resulting state deviation from the nominal working conditions, the $m^{l,g}_{\hat{x}'} + m^{l,g}_{\hat{y}'}$ coefficient model the relative positions of pre and post-intent high-density region location within the radiography.

We finally proposed a Trust and Reputation model (E) to quantify the contribution to resilience for each orchestration entity. We interpreted the intents reward as an estimation of its *goodness*, which is used to quantify the trust the RL agent has about each entity on the execution of a specific remediation action. Thereby, we proposed to aggregate reward values so that each entity reputation is evaluated on a range between -1, i.e low reputation, and 1 which stands for the high level of reputation. In particular, remediation actions associated with good recovery intents slightly increase the entity reputation while remediation actions associated with wrong recovery intents drastically decrease the reputation.

A straightforward extension of this work is to test the proposed RL autonomic recovery approach on a realistic virtualized service. However, the community lacks of virtualized network service projects that fully support automatic recovery through remediation actions like scaling. A promising open source project is open5gs [196] which implements all the network functions of a 5G core easily decom-

posable into micro-services.

On a practical note, the dataset splitting into per-resource sub-datasets implies a small loss of information regarding the interdependence between the metrics of the various resource groups. One possible solution to investigate would be to pick each AEs latent space representation and further encode them in a dedicated AE which will work on all the compact representation.

Another line of research consists in applying SYRROCA to other VoIP platforms, such as enterprise Telephone over IP infrastructures, as well as other NFV use cases such as the 4G and 5G core networks, also using more sophisticated NFV/SDN platforms such as OpenCord, OMEC, OPNFV, OSM and related software component logs.

Nous avons enfin proposé un modèle de confiance et de réputation (E) pour quantifier la contribution à la résilience de chaque entité d'orchestration. Nous avons interprété la récompense des intentions comme une estimation de son *goodness*, qui est utilisée pour quantifier la confiance que l'agent RL a envers chaque entité lors de l'exécution d'une action de remédiation spécifique. Nous avons donc proposé d'agréger les valeurs des récompenses de manière à ce que la réputation de chaque entité soit évaluée sur une plage comprise entre -1, c'est-à-dire une faible réputation, et 1, qui représente le niveau élevé de réputation. En particulier, les actions de remédiation associées à de bonnes intentions de récupération augmentent légèrement la réputation de l'entité, tandis que les actions de remédiation associées à de mauvaises intentions de récupération diminuent radicalement la réputation.

Une extension directe de ce travail consiste à tester l'approche de récupération autonome RL proposée sur un service virtualisé réaliste. Cependant, la communauté manque de projets de services de réseaux virtualisés qui supportent pleinement la récupération automatique par des actions de remédiation comme le redimensionnement. Un projet open source prometteur est open5gs [196] qui implémente toutes les fonctions réseau d'un noyau 5G facilement décomposable en micro-services.

D'un point de vue pratique, la division de l'ensemble de données en sous-ensembles de données par ressource implique une petite perte d'informations concernant l'interdépendance entre les métriques des différents groupes de ressources. Une solution possible serait de choisir la représentation de l'espace latent de chaque AE et de les encoder dans un AE dédié qui travaillera sur toutes les représentations compactes.

Une autre ligne de recherche consiste à appliquer SYRROCA à d'autres plateformes VoIP, telles que

les infrastructures de téléphonie sur IP d'entreprise, ainsi qu'à d'autres cas d'utilisation NFV tels que les réseaux centraux 4G et 5G, en utilisant également des plateformes NFV/SDN plus sophistiquées telles que OpenCord, OMEC, OPNFV, OSM et les registres de composants logiciels connexes.

Appendix A

Collected metrics

Name	Counter/Gauge
cpu_seconds_total_mode_idle	G
cpu_seconds_total_mode_iowait	G
cpu_seconds_total_mode_irq	G
cpu_seconds_total_mode_nice	G
cpu_seconds_total_mode_softirq	G
cpu_seconds_total_mode_steal	G
cpu_seconds_total_mode_system	G
cpu_seconds_total_mode_user	G
cpu_core_throttles_total	G
cpu_package_throttles_total	G
cpu_frequency_max_hertz	C
cpu_frequency_min_hertz	C
cpu_guest_seconds_total_mode_nice	G
cpu_guest_seconds_total_mode_user	G
cpu_scaling_frequency_hertz	C
cpu_scaling_frequency_max_hrts	C
cpu_scaling_frequency_min_hrts	C
context_switches_total	G
disk_io_now	C
disk_io_time_seconds_total	C

disk_io_time_weighted_seconds_tota	C
disk_read_bytes_total	C
disk_read_time_seconds_total	C
disk_reads_completed_total	C

Table A.1: Physical layer metrics

Name	**Counter/Gauge**
cpu_load_average_10s	C
cpu_load_average_10s	C
cpu_system_seconds_total	G
cpu_usage_seconds_total	G
cpu_user_seconds_total	G
fs_inodes_free	C
fs_inodes_total	C
fs_io_current	C
fs_io_time_seconds_total	G
fs_io_time_weighted_seconds_total	G
fs_limit_bytes	C
fs_reads_bytes_total	G
fs_read_seconds_total	G
fs_reads_merged_total	G
fs_reads_total	G
fs_sector_reads_total	G
fs_sector_writes_total	G
fs_usage_bytes	C
fs_writes_bytes_total	G
fs_write_seconds_total	G
fs_writes_merged_total	G
fs_writes_total	G
memory_cache	C
memory_failcnt	G

memory_failures_total	G
memory_mapped_file	C
memory_max_usage_bytes	C
memory_rss	C
memory_swap	C
memory_usage_bytes	C
memory_working_set_bytes	C
network_receive_bytes_total	G
network_receive_errors_total	G
network_receive_packets_dropped_total	G
network_receive_packets_total	G
network_transmit_bytes_total	G
network_transmit_errors_total	G
network_transmit_packets_dropped_total	G
network_transmit_packets_total	G

Table A.2: Virtual layer metrics

Appendix B

Going beyond diffserv in IP traffic classification

Quality of Service (QoS) management in IP networks today relies on static configuration of classes of service definitions and related forwarding priorities. Packets are actually classified according to the DiffServ architecture based on the RFC 4594, typically thanks to static configuration or filters matching packet features, at network access equipment. In this annex, we propose a dynamic classification procedure, referred to as Learning-powered DiffServ (L-DiffServ), able to detect the distinctive characteristics of traffic and to dynamically assign service classes to IP packets. The idea is to apply semi-unsupervised Machine Learning techniques, such as Linear Discriminant Analysis (LDA) and K-Means, with a proper customization to take into account the issues related to packet-level analysis, i.e. unbalanced distribution of traffic among classes and selection of proper IP header related features. The performance evaluation highlights that L-DiffServ is able to change dynamically the classification outcome, providing a higher number of classes than DiffServ. This last result represents the first step toward a more granular differentiation of IP traffic.

B.1 Introduction

Quality of Service (QoS) technologies commonly address the network link bottleneck issue by introducing protocols to support priority packets to pass first. Differentiated Services (DiffServ) is the de-facto QoS protocol used in IP networks, in most of Internet Service Provider (ISP) networks as well as in layer-3 operated data-center and entreprise networks. Once packets are classified into classes of services at a network access point or router, they can get dropped or delayed in the queuing system

of core routers according to class priority. DiffServ was proposed as a stateless alternative to the stateful Integrated Services (IntServ) [197] architecture, suffering from scalability issues by following an end-to-end layer-4 flow virtual circuit resource reservation approach.

Indeed, IntServ revealed to be not well suited for heterogeneous systems as it requires support at the source terminal, destination terminal, all intermediate routers and at the application as well, which made it practically difficult to happen. In addition, IntServ-enabled routers to maintain an internal state for each virtual circuit opened by the resource reservation protocol [198], which makes them vulnerable to failures. Therefore, the community turned to DiffServ as a lighter and stateless approach, not touching at terminals nor applications, and not requiring all routers on the way to implement it. User traffic is mapped by edge routers or access points into the appropriate forwarding class, encoded into the packet header. This information is then used by the intermediate routers to differentiate packet processing, as forwarding classes indicate drop and resource priorities.

In this annex, we propose to go beyond the legacy DiffServ policy of manually setting QoS classes, in order to be able to dynamically learn the appearance of new classes worth being differentiated. Our proposal, called Learning-powered DiffServ (L-DiffServ), is to dynamically refine the set of classes in order to increase the granularity of the macro service classes. We propose a machine learning methodology for determining valuable sub-classes for actual packet classification. Our solution is composed of three main building blocks: i) data pre processing, performing features extraction and oversampling); ii) dimensionality reduction by means of the Linear Discriminant Analysis (LDA) procedure; and iii) clustering and classification, exploiting the K-Means algorithm. We run our experiments against real data from the MAWI dataset [199], containing daily IP traces of a transpacific backbone link.

The annex is structured as follows. In Section B.3 we report the numerical result. Finally, Section B.4 concludes the annex.

B.2 Machine Learning Methodology

This section details our Learning-powered DiffServ (L-DiffServ) solution for dynamic refinement of DiffServ classes of services, including dataset processing, oversampling, dimensionality reduction, clustering and classification steps.

B.2.1 The L-DiffServ architecture

Our proposal consists in generating a new set of service classes through a hybrid semi-unsupervised machine learning technique that automatically assesses the new number of service classes, identifying the sub-classes within existing pre-set classes, based on features extracted from IP packet flows. Figure B.1 depicts the L-DiffServ workflow.

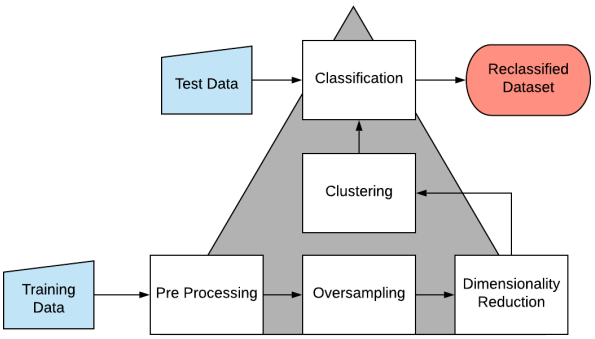


Figure B.1: L-DiffServ Workflow Overview.

Data preparation is accomplished through Pre-processing, Oversampling and Dimensionality Reduction. In the pre-processing phase the numeric values from captured packet features are normalized, while the categorical ones are transformed into binary; at the end of these operations, we delete the features with zero variance because they have not information for the differentiation between classes. In the oversampling phase, we generate artificial samples of the minoritary class (the one with less occurrences) to balance the data set. It is worth noting that we choose to oversample the information of the less numerous classes rather than undersampling the best effort class because undersampling could lead to significant information losses [200]. Finally, we reduce the space dimensionality maximizing the variance between service classes and projecting the starting dataset into the new dimensional space.

The goal of the Clustering step is to produce a new set of classes, so that the proposed classification is based on a grouping done according to the clustering method outcome. First, we evaluate the clusters obtained through the Silhouette Coefficient index of goodness [201]; it measures the magnitude between cohesion (intra-cluster distance) and dispersion (inter-cluster distance) for each group. Once we obtain the maximum Silhouette value we establish the optimal number of clusters. In this way we can assign a new label for each observation and train the model based on our classification. The end of the

Clustering step coincides with the beginning of Classification one. We can reclassify a test trace, with the same structure, applying the transformations used in our methodology, evaluating the minimum euclidean distance between the packet and centroids extracted from the Clustering step.

We validate this methodology on a public dataset provided by MawiLab [199] described hereafter.

B.2.2 Dataset and Features

The WIDE project publishes daily IP traces of a transpacific link, called the MAWI Archive [199]. Each file contains 15 minutes of traffic flows, captured between 14:00:00 and 14:15:00 local time. This represents usually between 10 and 20 GB of traffic for one file. Before being released, traces are anonymized to hide any personal information (removing application data and scramble IP addresses with the Crypto-PAn Algorithm [202] following collision-free and prefix-preserving principles). As of our knowledge, there is no other equivalent public dataset, spanning many months, we could exploit.

In our analysis, we work on traces of multiple days across multiple weeks for training and classification. We consider the trace belonging to the period which goes from 3rd April 2019 to 10th May 2019, considering only Wednesday. The average size of the traces is 19240.36 MB and the average number of packets is 252,046,154. In the traffic analysis, we work only with IPv4 packets along one forwarding direction by filtering packets through the MAC address. We consider for each trace 3,000,000 packets of the total trace because of memory limits; we make a random sample splitting the trace with 200,000 packets maintaining the percentage of packets for each service classes. We focus the attention on a specific day in this section (April 3, 2019).

The following characteristics are the features extracted from every packet header: Internet Header Length (IHL), Differentiated Services Code Point (DSCP), Explicit Congestion Notification (ECN), Total Length, Flags, Fragment Offset, Time To Live (TTL), Protocol, Source address, Destination address, and from the TCP header part we extract Source Port and Destination Port.

In Table B.1 we report the mapping between the DSCP value and the service class name we adopt during the analysis as packet labels. Observing the column *Class Label*, clearly we unify the CS class of service both with the AF and with EF because there are few observation for the backward compatibility classes.

In the following we show each step of our methodology.

DSCP Value	DSCP Class	Class Label
48, 56	CS6, CS7	Network and Internetwork Control
40, 46	CS5, EF	Critical RTP Voice
32, 34, 36, 38	CS4, AF4	Flash Override
24, 26, 28, 30	CS3, AF3	Flash Voice
16, 18, 20, 22	CS2, AF2	Immediate
8, 10, 12, 14	CS1, AF1	Priority
0	CS0	Best Effort

Table B.1: Mapping between DSCP and class labels, from: [1]

B.2.3 Pre Processing

We process the dataset composed of the packet features with the DSCP marking as label. We do not to consider the features with zero variance because not important for packets differentiation. Such features revealed to be the Internet Header Length (IHL), the Flags except for the DF (Do-not-Fragment) flag, and the Fragment Offset.

For the four categorical variables, i.e., Source and Destination Address and Source and Destination Port, [202] uses a different key for each day to anonymize the trace, so we cannot include IP addresses in our model; instead, we can handle the Source and Destination Port - our idea is to determine an identification port for each packet, trying to cover a high-density of information. We cover the ports that allow to keep 90% of packet volume, while the remaining ports with a small occurrence are transformed into an artificial port of number 0. Moreover for the packets whose protocol has no defined port, as ICMP packets, we assign the port number -1; the values -1 and 0 have not a numerical importance but they affect the presence or the absence of such port within the packet. In this way we create the summary variable called *Heavy Port*. Finally, in the data Pre-Processing we transform the categorical variables into binary variables applying *One-Hot Encoding* [203] (it transforms a variable with n observations and d values, to d binary dichotomous variables with n observations, each observation indicating the presence with 1 or absence with 0 of the variable) while the numerical variables are normalized. For data normalization we use the *MinMaxScaler* function,

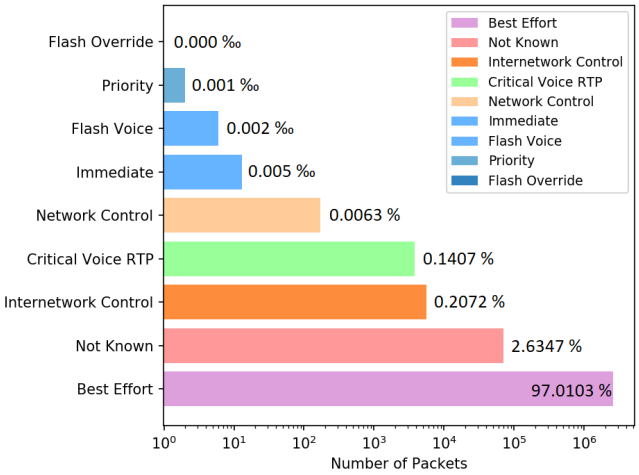


Figure B.2: DSCP Distribution in the considered MAWI dataset.

with the following formula:

$$\frac{x_i - min(x)}{max(x) - min(x)} \tag{B.1}$$

However, until now, our methodology considers all packets in an undifferentiated way regarding the DSCP marking. Nevertheless, observing the service class occurrences there is a great imbalance in favour of the 0 DSCP label (best-effort class). Thus, we apply the analysis for the Source and Destination Port and the normalization part splitting the beginning data frame into two components: the first one with only the best-effort data, while in the second one we consider everything not marked as 0 for DSCP value. In this way we maintain the information related to best-effort and non best-effort classes, otherwise the few observations of the non best-effort classes could disappear under the magnitude of the best-effort packets. This last consideration opens the door to the main challenge for our analysis, the data unbalance.

B.2.4 Oversampling

Let us comment on the the DSCP distribution from our trace, focusing on the percentage of packets for each service class in Figure B.2. More than 97% of the traffic is best-effort traffic, the other services covering the remaining 3%. We report in the histogram the label *Not Known*, not listed in the Table B.1; it corresponds to DSCP values not recommended by RFC 4594 [1], most of them being values

between 0 and 8. This traffic is known as Scavenger, i.e., traffic with lower priority than the best-effort class and to which is allocated the lowest amount of bandwidth.

Furthermore it is interesting to observe the percentage of packets related to the other service classes, to have a complete picture of the available information. The Scavenger class is the second service with the greatest usage 2.63%. The Expedited Forwarding (CS5, EF) class and Network & Internetwork Control (CS6 and CS7) have a good percentage of occurrences: the first with 0.14% and the second with 0.21%. However, the four Assured Forwarding (AF) classes have almost no packets. In fact the packets related to the Priority (CS1, AF1), Immediate (CS2, AF2), Flash Voice (CS3, AF3) and Flash Override (CS4, AF4) are only the 0.0008%.

Such unbalanced data distribution represents one of the most discussed problems in the Machine Learning literature, as it strongly influences the behaviour of classification algorithms in favour of a specific class. This problem is known as the *Paradox of Accuracy* [204]. In fact, if we limit ourselves to identify best-effort packets from non best-effort, any classification algorithm trained on this data will classify everything as best-effort getting always more than 90% of Accuracy.

To countermeasure this issue, we exploit the Smote oversampling technique presented in [205]. The Smote algorithm can oversample the service classes for which we have very few samples, without losing information from the best-effort class. In this way we obtain a balanced dataset, where each service class has the same number of occurrences of the best-effort class, hence we can extract the characteristics which maximize the differentiation between service classes.

B.2.5 Dimensionality Reduction

The dataset matrix after the over-sampling procedure has $10,611,286$ rows, representing the packets, and 42 columns, as number of features (including the DSCP label). Our purpose is to reduce the features only to the ones that allow to identify the service class each packet belongs to. We reduce the dimensionality of the space working with the LDA (Linear Discriminant Analysis) technique [206]: it allows to specify the number of axis for the new space, thus obtaining a 3D graphic distribution of the packets. Moreover it works in a supervised way, being also a classification algorithm; in this way it can maximize the variance between classes of services; in fact, we set it to use a variance of 97.8%. In Table B.2, for each one of the new 3 LDA axis, we report the rate of correlation between the original features and the corresponding axis; for each axis, we report only the top ten components.

Variable	Value
Port 123	30.6%
Protocol 17	20.3%
Port 443	6.5%
Protocol 47	2.6%
Protocol 97	1.9%
Protocol 6	1.8%
Protocol 89	1.7%
Port 22	1.7%
Protocol 80	1.6%
Port 8080	1.6%

Variable	Value
Port 123	27.1%
Protocol 47	10.6%
Protocol 97	4.6%
Protocol 4	3.2%
Protocol 6	3.1%
Protocol 17	3.0%
Port 22	2.8%
Port 53855	2.8%
Port 53469	2.8%
Port 8080	2.7%

Variable	Value
Protocol 47	12.5%
Protocol 97	6.1%
Protocol 4	5.9%
Protocol 17	5.2%
Protocol 6	4.9%
Port 22	4.9%
Port 53855	3.7%
Port 443	3.2%
Port 89	2.9%
Port 8080	2.9%

Table B.2: Linear Discriminant Components

B.2.6 Clustering

At this point, the dataset space is projected into the new dimensional space defined by LDA, and the classification step begins. We have to cluster packets even if they have already an assigned label. Our purpose is to increase the granularity of the current service classes. The starting point is the recommended classification in RFC 4594 [1], which represents the macro-classification currently used for the DSCP marking:

- Best-Effort (BE);
- Not Known;
- Assured Forwarding (AF);
- Expedited Forwarding (EF);
- Network & Internetwork Control.

The problem we are going to face is a clustering problem, where data do not have a label and we cannot observe the correctness of the results obtained. However, it is essential to analyze the results according to an index of goodness about the clustering. Therefore, the fundamental tools are the clustering algorithm and the measure to evaluate the results. These tools lead us to determine the optimal number of centroids (k), which are the new available service classes. For the clustering algorithm we work with the well-known K-Means [207], while for the evaluation index we use the Silhouette Coefficient [201].

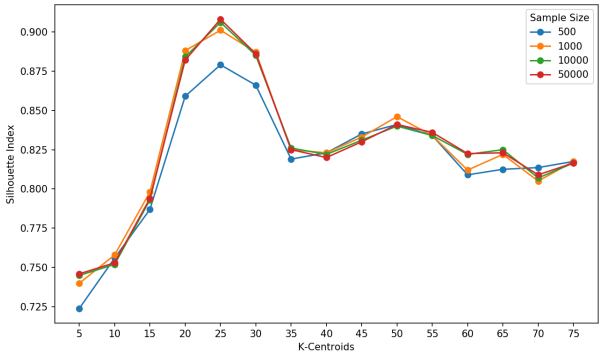


Figure B.3: Silhouette as a function of the number of centroids.

The Silhouette index is a measure of how similar a cluster is to its own cluster (*Cohesion*) compared to other ones (*Separation*). The Silhouette ranges from -1 to $+1$, where a high value indicates that the object is well matched to its own cluster and poorly matched to neighbouring clusters. The formula and notations used to compute the Silhouette index are as follows:

$$S(i) = \frac{b(i) - a(i)}{max\{a(i), b(i)\}} \quad if \quad |C(i)| > 1 \tag{B.2}$$

- $S(i)$: Silhouette index for cluster $C(i)$;
- $a(i)$: Cohesion;
- $b(i)$: Separation;
- $|C(i)|$: Number of elements in $C(i)$.

We evaluate the optimal number of clusters (k) to establish our final proposal for the service classes. The analysis considers a range of possible values for k, from 5 to 75 with a step of 5. Figure B.3 shows the results of the Silhouette Coefficient.

In each k we compute (B.2) 100 times for each possible sample size; in this way we capture the real behaviour of our packets population without considering all the packets. In Figure B.3 we show the trend according to the variation of the sample size. At the beginning we have a dramatic increase for the Silhouette Index passing from 5 to 25 centroids. Then there is a decrease until 35 centroids and then, for the remaining part of the line chart, we have a steady behaviour around 0.825. So we select 25, the peak of our analysis, as the new number of service classes. It is interesting to check the

Figure B.4: Silhouette Coefficient & 3D K-Means.

association among subclasses and macro classes. In the Figure B.4 we show the result of K-Means with 25 centroids; analyzing each cluster according to the Silhouette index. In the upper plot, on the left side, the dotted red line identifies the average value between all the clusters for the Silhouette Coefficient, equal to 0.908. In the right side we have the legend related to the 3D-plot about K-Means clustering. In this legend we can see the sub-classes identified within the main classes. The best-effort class is differentiated into 11 sub-categories, while the Not Known service is divided into 7 different sub-classes. For both Assured Forwarding (AF) and Critical Voice RTP (EF) we see that our clustering algorithm finds only one subclass, without sub-divisions.

B.2.7 Classification

For classification we use the K-means algorithm, exploiting the centroids built during the clustering analysis: in this way we can obtain the reclassification in real time. The low complexity of this step is proven by the following result: the time to reclassify 1 million packets is about 0.51 seconds. In this way, we can obtain any part of the trace used as a test reclassified according to our new distribution in service classes.

B.3 Numerical Analysis

We can summarize the results in terms of generated sub-classes fragmentation, for the whole period considered (Apr. 3 to May 8, 2019, one day per week), through the bar chart in Figure B.5. For each of the given 5 classes, we report the number of subclasses generated - the total number of subclasses is given by the first column for each date. It is relevant to observe that the best effort service class

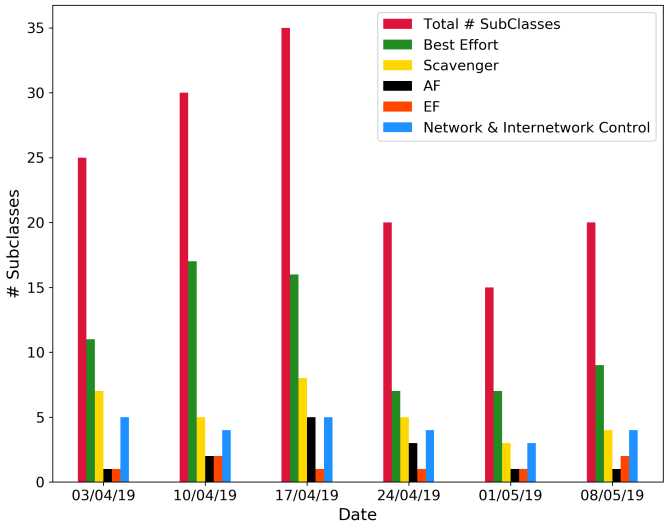


Figure B.5: L-Diffserv Dynamical Behaviour.

is always the most differentiated one in subclasses, followed by the Scavenger class. The other service classes, especially AF and EF, show a low fragmentation; traffic from these classes is therefore not showing specific behaviors within the class, likely because existing features are already used in general to distinguish them (e.g., Port number or Protocol) from the other classes.

An important aspect to highlight is that the number of subclasses generated for the best effort and Scavenger classes have a high variability in time - e.g., the number of best effort subclasses in Apr. 4 is twice the same number in Apr. 24, 2019. This shows that L-DiffServ is able to capture the dynamicity in traffic patterns and to adapt traffic aggregation in subclasses.

The capacity to aggregate traffic flows in subclasses can be beneficial in bottleneck management; large amount of traffic (of best effort and Scavenger classes) that by default would all be assigned to two priority levels, can instead by discriminated by L-DiffServ in up to 23 priority levels, while being reassured that subclasses are ordered with respect to each other based on L-DiffServ ordering. Hence traffic loss can be concentrated to few flows of few subclasses instead of a high number of flows spanning a large number of subclasses.

B.4 Conclusions

This work aims to provide a methodology, named L-DiffServ, for reaching dynamic class of service generation in IP networks. In fact, L-Diffserv differentiates further the traffic based on an existing

high-grain classification, giving us advices to improve the resource allocation (buffer and bandwidth) according to this new service classification. As further work, we want to test the behavior of L-DiffServ in real-time systems, increasing the amount of packets analyzed. Finally we want to evaluate the behaviour of the network under congestion according to the different service classification methods, describing advantages and disadvantages of our proposal and the current DiffServ method. We do also envision extending the methodology for service-level agreement management in network slicing.

Appendix C

Performance Comparison of ONOS and ODL controllers

In the following, we report on control-plane reactivity to topology changes and discovery events, comparing ONOS and ODL behaviors. Topology update reactivity SDN controllers are expected to maintain an updated view of the network in a semi-real time fashion in order to let applications work with a consistent view. Generally speaking, the topology update is implemented following an event driven pattern logic. When some specific packets sent by SDN switches are received or some expected packets are not received by the controller, an event is raised in the controller's core. This event is received and held by subscribed listeners, which will in turn solicit topological representation changes in a database, eventually distributed. We focus in particular on what happens in ONOS and ODL controllers when a OFPT_PORT_STATUS packet is received from a switch to notify a change in a port's status after a link disruption event or after a link is established/re-established.

Supposing that a path computation application is activated in the controller, the controller has to react to this change in the topology, eventually installing new flows to circumvent the disruption and finally ensure communications.

In this section, we evaluate how fast and promptly ONOS and ODL controllers perform these update actions when reacting to a topological change. To perform this comparison, we analyze a very basic test case (as in C.1): two hosts, H1 and H2, connected by a single path composed of 6 links and 5 SDN switches (OVS switches), and exchanging UDP packets through an Iperf [16] session. We use Quali version for ONOS and Oxygen version for ODL.

During tests, the configuration shown in Figure C.1 was deployed through a developed python

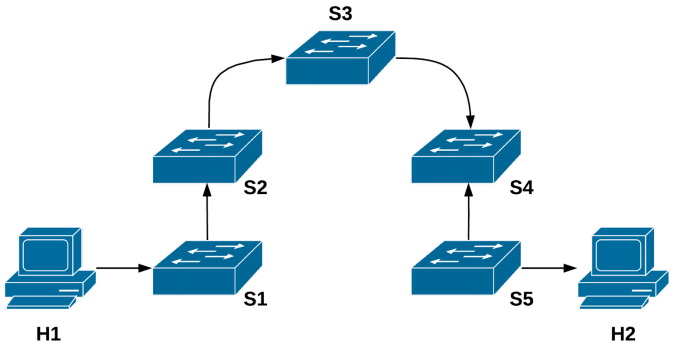


Figure C.1: Test case network topology

module that simulates complex network topologies with redundant links and alternative paths among hosts. The simulator also provides a fault injection module that is capable of injecting faults and degradations on each of the simulated network elements. To ensure connectivity between the hosts, "org.onosproject.fwd" and "org.onosproject.openflow" apps were activated in ONOS and "odl-l2switch-all" was activated in ODL. In order to gather sufficient data, we iterate the test case 1400 times, cleaning both topology and controller state between iterations. Supposing the i-th iteration starts at time t0 when the Iperf session is started, at timet0+=Tstart a failure in the link between switches S2 and S3 is introduced and finally after T seconds the link is restored. To monitor the traffic flowing along the path, at t0 a tshark [17] capture is started on the link between S3 and S4. In particular, from this capture, it is possible to extract the time at which the first UDP packet appears on link S3-S4, defined as Tfirst. Knowing that the link is restored at time t0++T=Tstop, the controller's reaction time can be computed as Treact=Tfirst-Tstop . Let us note that Iperf is configured in UDP mode in order to remove all synchronization overhead specific to TCP that would have biased the reaction time. Furthermore, in the topology, no alternative path from H1 to H2 was created so that the controller will not perform actions other than those described.

Figure C.2 reports the empirical probability distribution function (PDF) of the reaction time (Treact) for the 1400 tests and for both ONOS and ODL. For a number of tests both controllers react in a similar way. However, ODL shows two different modes: in 30.1% of the tests, the reaction times fall into the interval [0, 0.04], while in the remaining 69.9% of cases values are in [3, 10]. Thus, in Figure C.3, we represent the dynamics of the reaction times in two split PDF plots for the two modes. Equivalent results for ONOS are in Figure C.4, in a single plot, for the sake of clarity.

ONOS appears to be significantly more stable than ODL. Note that ONOS also shows two separate

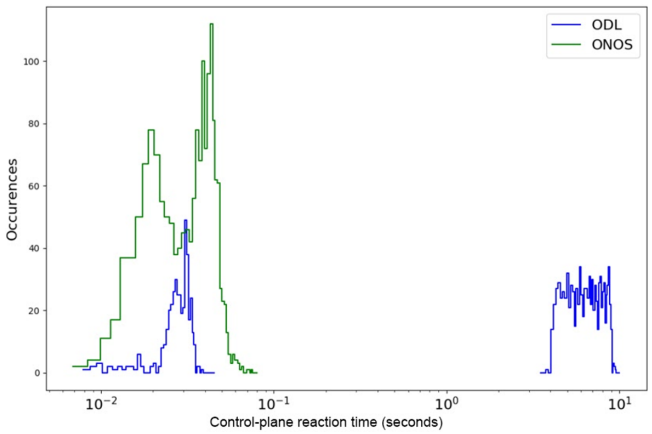
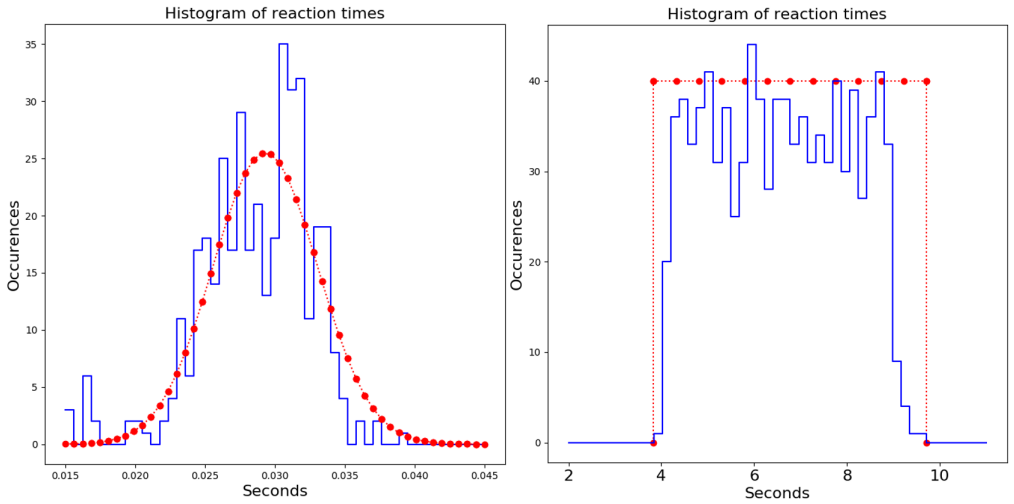


Figure C.2: Distribution of reaction time (Treact) for ONOS and ODL controllers.



(a) [0,0.04] seconds (b) [3,10] seconds

Figure C.3: Distribution of the reaction times for ODL at two distinct intervals

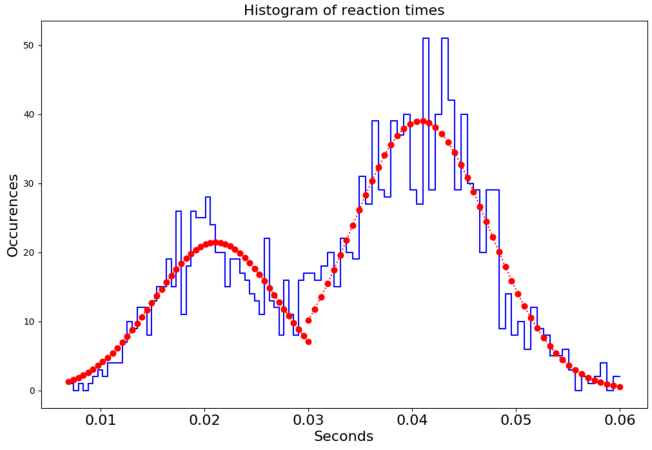


Figure C.4: Distribution of the reaction times for ONOS

modes, similar to ODL, yet they are much closer than with ODL; while the distance between the modes is approximately 6 seconds for ODL, it is approximately 20 ms for ONOS with no occurrences gap.

ONOS is also much faster in reacting to topology event updates, with a median reaction time of 36 ms; that is two orders of magnitude less than ODL that has a median of 5.45 seconds, as shown in the boxplot statistics in Figure C.5.

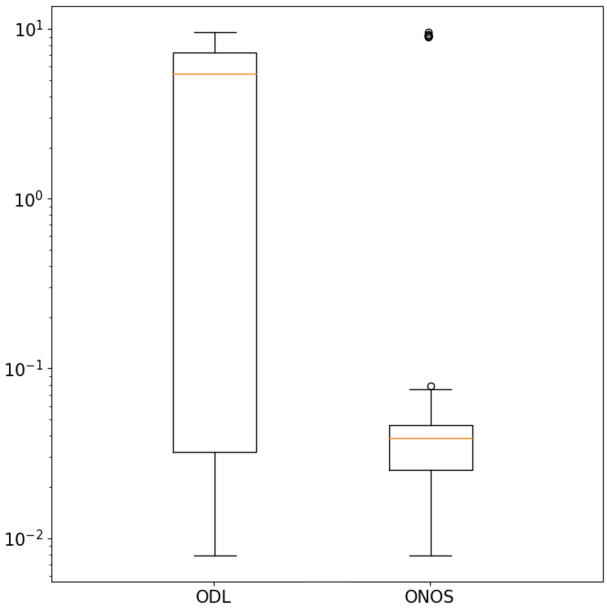


Figure C.5: Boxplot statistics of the reaction times. A boxplot shows the minimum, first quartile, median in red, third quartile and maximum values.

To better understand the reasons for the detected unstable ODL behavior, we tried to capture the variability across tests characterizing how frequently the reaction time switches from the first mode (Figure C.3a) to the second one (Figure C.3b) in subsequent tests. To do so, we use a metric that is incremented by one each time a switch from the first mode to the second one is detected, when considering i-th and i+1-th tests. The result is shown in Figure C.6. A controller whose reaction times in subsequent tests would flip among the modes, i.e., fall in the other mode each time, would have produced the first quadrant bisector line in such a plot. However, it is quite close to the bisector, which means that ODL is very unstable as it is reacting in very different ways across subsequent tests. In order to search for possible correlations, we also computed the empirical probability that at the i-th test the reaction time is in the first [second] mode while in the subsequent i+1-th test the reaction time falls in the second [first] mode. We found no difference, with an empirical probability to switch from

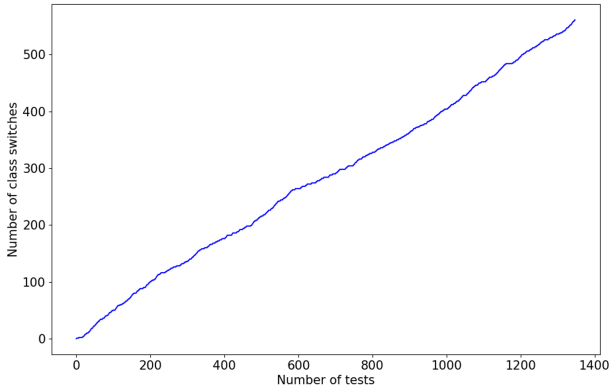


Figure C.6: Number of mode switches as a function of the number of tests – ODL

the first to the second of 0.2075, while the reverse is 0.2083 (very close to the former). Summing these probabilities, we obtain a probability of 0.4158 to switch from one class to another, which confirms the unpredictability of the ODL controller.

An aspect that remains unclear from the test is the origin of the large gap between the two working modes in ODL. Further work might inspect openflow messages exchanged between the controller and the switches to identify the ODL core mechanism triggered by those messages.

C.0.1 Topology discovery

We analyze the amount of control traffic required for ONOS and ODL to discover and update the topology over time. Both controllers use Link Layer Discovery Protocol (LLDP) to infer links connecting switches. Basically, the controller sends a PACKET_OUT message to each switch containing as many LLDP frames as active ports in the corresponding switch. Each switch then sends out LLDP frames in designated ports, forwarding to the controller, through a PACKET_IN message, each LLDP frame it receives. Consequently, the controller can infer links binding the port where the packet was sent (inside LLDP frame) and the port where the packet was received (a field in the PACKET_IN packet). This procedure is repeated periodically in order to maintain an up-to-date topology; every 3 seconds in ONOS and every 5 seconds in ODL, by default.

To test the implementation of LLDP in ONOS and ODL, we simply record for a given period all PACKET_IN and PACKET_OUT messages exchanged since the first topology discovery using a topology such as that in Figure C.1. In order to fairly estimate the amount of traffic, we manually set the LLDP cycle in both controllers to 5 seconds.

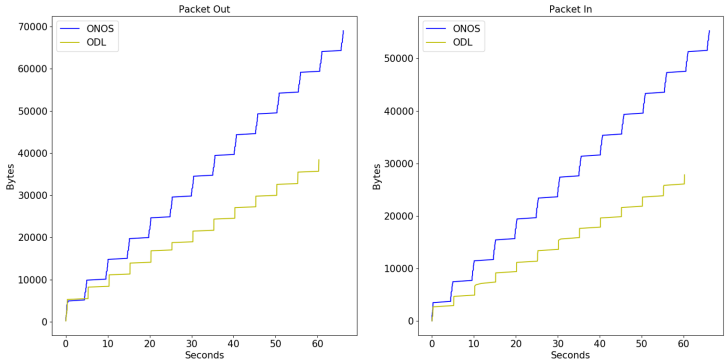


Figure C.7: Topology discovery volume for both PACKET_IN and PACKET_OUT messages

Figure C.7 shows the obtained results. ONOS produces a larger amount of control traffic in terms of PACKET_OUT and thus PACKET_IN with respect to ODL, as each PACKET_OUT will result in a PACKET_IN if there is an active link on the related port. We attribute this difference in the amount of exchanged packets to an optimized implementation of LLDP in ODL, e.g. OFDPV2.

References

[1] J. Babiarz, K. H. Chan, and F. Baker, "Configuration guidelines for diffserv service classes," *RFC 4594*, 2006.

[2] Ericsson, "Ericsson mobility report," https://www.ericsson.com/en/press-releases/2020/11/more-than-1-billion-people-will-have-access-to-5g-coverage-by-the-end-of-2020.

[3] "Global network automation market research report," https://www.marketresearchfuture.com/reports/netwo automation-market-5852, accessed on 15/10/2021.

[4] ETSI GS NFV 002 V1.2.1, "Network functions virtualisation (nfv): Architectural framework," December 2014.

[5] "Docker vs Virtual Machines (VMs) : A Practical Guide to Docker Containers and VMs," https://www.weave.works/blog/a-practical-guide-to-choosing-between-docker-containers-and-vms, accessed on 15/10/2021.

[6] ETSI GS NFV-IFA 011 V3.3.1, "Network functions virtualisation (nfv) release 3; management and orchestration; vnf descriptor and packaging specification," September 2019.

[7] "Kubernetes architecture," https://kubernetes.io/it/docs/concepts/overview/components/, accessed on 15/10/2021.

[8] J. Mitola, "Cognitive radio. an integrated agent architecture for software defined radio." Ph.D. dissertation, Royal Institute of Technology, 2000.

[9] ETSI GS NFV-REL 001 V1.1.1, "Network function virtualisation (nfv)-resiliency requirements," January 2015.

[10] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[11] C. Fortuna and M. Mohorcic, "Trends in the development of communication networks: Cognitive networks," *Computer Networks*, vol. 53, no. 9, pp. 1354–1376, 2009.

[12] D. D. Clark *et al.*, "A knowledge plane for the internet," in *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, 2003, pp. 3–10.

[13] Q. Mahmoud, *Cognitive networks: towards self-aware networks.* John Wiley & Sons, 2007.

[14] A. Diamanti, J. M. Sanchez Vilchez, and S. Stefano, "An ai-empowered framework for cross-layer softwarized infrastructure state assessment," in *Transactions on Network and Service Management, major revision.*

[15] A. Diamanti, J. M. S. Vilchez, and S. Secci, "Procédé de contrôle d' une entité d'orchestration dans un réseau logiciel." Patent FR2 107 239, 2021.

[16] S. E. Madnick, "Time-sharing systems: Virtual machine concept vs. conventional approach," *Modern Data*, vol. 2, no. 3, pp. 34–36, 1969.

[17] M. Chiosi *et al.*, "Network functions virtualisation.introductory white paper," *SDN and OpenFlow World Congress*, 2012.

[18] H. Hawilo *et al.*, "Nfv: state of the art, challenges, and implementation in next generation mobile networks (vepc)," *IEEE Network*, vol. 28, no. 6, pp. 18–26, 2014.

[19] A. Desai *et al.*, "Hypervisor: A survey on concepts and taxonomy," *International Journal of Innovative Technology and Exploring Engineering*, vol. 2, no. 3, pp. 222–225, 2013.

[20] R. Morabito, J. Kjällman, and M. Komu, "Hypervisors vs. lightweight virtualization: a performance comparison," in *2015 IEEE International Conference on Cloud Engineering*, 2015, pp. 386–393.

[21] "Linux cgroups," https://man7.org/linux/man-pages/man7/cgroups.7.html, accessed on 15/10/2021.

[22] "Linux namespaces," https://man7.org/linux/man-pages/man7/namespaces.7.html, accessed on 15/10/2021.

[23] T. Bui, "Analysis of docker security," *arXiv preprint arXiv:1501.02967*, 2015.

[24] ETSI GS NFV-MAN 001 V1.1.1, "Network function virtualisation (nfv); management and orchestration," December 2014.

[25] "Open source mano(osm)," https://osm.etsi.org/, accessed on 15/10/2021.

[26] "Tacker," https://wiki.openstack.org/wiki/Tacker, accessed on 15/10/2021.

[27] "Openstack," https://wiki.openstack.org/, accessed on 15/10/2021.

[28] "Open baton," https://openbaton.github.io/cases.html, accessed on 15/10/2021.

[29] T. Benson, A. Akella, and D. A. Maltz, "Unraveling the complexity of network management." in *Prc. of the 6th USENIX Symposium on Networked Systems Design and Implementation*, 2009, pp. 335–348.

[30] D. Kreutz and otehrs, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2014.

[31] N. McKeown *et al.*, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM computer communication review*, vol. 38, no. 2, pp. 69–74, 2008.

[32] S. Azodolmolky, *Software defined networking with OpenFlow*. Packt Publishing, 2013.

[33] S. Sezer, S. Scott-Hayward, P. K. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, and N. Rao, "Are we ready for sdn? implementation challenges for software-defined networks," *IEEE Communications Magazine*, vol. 51, no. 7, pp. 36–43, 2013.

[34] R. Ozdag, "Intel® ethernet switch fm6000 series-software defined networking," *See goo. gl/AnvOvX*, vol. 5, 2012.

[35] R. Amin, M. Reisslein, and N. Shah, "Hybrid sdn networks: A survey of existing approaches," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 3259–3306, 2018.

[36] S. Secci, A. Diamanti, J. Sanchez, M. Vilchez *et al.*, "Security and performance comparison of onos and odl controllers," *Open Networking Foundation, Informational Report*, September 2019.

[37] S. Scott-Hayward, G. O'Callaghan, and S. Sezer, "Sdn security: A survey," in *Proc. of the IEEE SDN For Future Networks and Services*, 2013, pp. 1–7.

[38] E. Zeydan and Y. Turk, "Recent advances in intent-based networking: A survey," in *Proc. of the IEEE 91st Vehicular Technology Conference*, 2020, pp. 1–5.

[39] A. Clemm, L. Ciavaglia, L. Granville, and J. Tantsura, "Intent-based networking-concepts and overview," *Internet Engineering Task Force, Internet-Draft*, 2019.

[40] F. Callegati, W. Cerroni, C. Contoli, and F. Foresta, "Performance of intent-based virtualized network infrastructure management," in *Proc. of the IEEE International Conference on Communications*, 2017, pp. 1–6.

[41] J. Nïemöller, L. Mokrushin, S. K. Mohalik, M. Vlachou-Konchylaki, and G. Sarmonikas, "Cognitive processes for adaptive intent-based networking," *Ericsson Technology Review*, 2020.

[42] B. Hayes, "Cloud computing," *Communications of the ACM*, vol. 51, no. 7, pp. 9–11, July 2008.

[43] R. Mijumbi *et al.*, "Network function virtualization: State-of-the-art and research challenges," *IEEE Communications surveys & tutorials*, vol. 18, no. 1, pp. 236–262, 2015.

[44] P. Mell and T. Grance, "The NIST definition of cloud computing," *Special Publication*, September 2011.

[45] I. a. Parvez, "A survey on low latency towards 5g: Ran, core network and caching solutions," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 3098–3130, 2018.

[46] F. J. Ros and P. M. Ruiz, "Five nines of southbound reliability in software-defined networks," in *Proc. of the the third workshop on Hot topics in software defined networking*, 2014, pp. 31–36.

[47] R. Bryant *et al.*, "Accelerating nfv delivery with open-stack," *OpenStack Foundation Report*, 2016.

[48] N. Kratzke and P.-C. Quint, "Understanding cloud-native applications after 10 years of cloud computing-a systematic mapping study," *Journal of Systems and Software*, vol. 126, pp. 1–16, 2017.

[49] M. Richards, *Microservices vs. service-oriented architecture*. O'Reilly Media, 2015.

[50] C. Pautasso, O. Zimmermann, M. Amundsen, J. Lewis, and N. Josuttis, "Microservices in practice, part 1: Reality check and service design," *IEEE software*, vol. 34, no. 01, pp. 91–98, 2017.

[51] A. Balalaie, A. Heydarnoori, and P. Jamshidi, "Microservices architecture enables devops: Migration to a cloud-native architecture," *Ieee Software*, vol. 33, no. 3, pp. 42–52, 2016.

[52] J. P. Sterbenz *et al.*, "Resilience and survivability in communication networks: Strategies, principles, and survey of disciplines," *Computer networks*, vol. 54, no. 8, pp. 1245–1265, 2010.

[53] R. J. Ellison *et al.*, "Survivable network systems: An emerging discipline," Carnegie-mellon Univ Pittsburgh PA Software Engineering Inst, Tech. Rep., 1997.

[54] P. E. Heegaard and K. S. Trivedi, "Network survivability modeling," *Computer Networks*, vol. 53, no. 8, pp. 1215–1234, 2009.

[55] "Analysis techniques for dependability - Reliability block diagram method," Standard, 1991.

[56] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE transactions on dependable and secure computing*, vol. 1, no. 1, pp. 11–33, 2004.

[57] N. F. Doherty, L. Anastasakis, and H. Fulford, "The information security policy unpacked: A critical study of the content of university policies," *International journal of information management*, vol. 29, no. 6, pp. 449–457, 2009.

[58] A. A. Lazar and G. Pacifici, "Control of Resources in Broadband Networks with Quality of Sen/ice Guarantees," *IEEE Communications Magazine*, 1991.

[59] M. W. Ibrahim, "Level of resilience measure for communication networks," *Journal of Information and Communication Technology*, vol. 17, no. 1, pp. 115–139, 2017.

[60] W. Najjar and J.-L. Gaudiot, "Network resilience: a measure of network fault tolerance," *IEEE Transactions on Computers*, vol. 39, no. 2, pp. 174–181, 1990.

[61] A. Jabbar, "A framework to quantify network resilience and survivability," Ph.D. dissertation, University of Kansas, 2010.

[62] D. Zhang and J. P. Sterbenz, "Measuring the resilience of mobile ad hoc networks with human walk patterns," in *2015 7th International Workshop on Reliable Networks Design and Modeling (RNDM)*. IEEE, 2015, pp. 161–168.

[63] "Etsi poc 35," http://nfvwiki.etsi.org/index.php?title=Availability_Management_with_Stateful_Fault_Tolerance, accessed on 15/10/2021.

[64] "Project Doctor," https://wiki.opnfv.org/display/doctor, accessed on 15/10/2021.

[65] "Cloudify," https://cloudify.co/, accessed on 15/10/2021.

[66] M. A. Khan and H. Tembine, "Meta-learning for realizing self-x management of future networks," *IEEE Access*, vol. 5, pp. 19 072–19 083, 2017.

[67] R. W. Thomas *et al.*, "Cognitive networks," pp. 17–41, 2007.

[68] V. Srivastava and M. Motani, "Cross-layer design: a survey and the road ahead," *IEEE communications magazine*, vol. 43, no. 12, pp. 112–119, 2005.

[69] V. Kawadia and P. R. Kumar, "A cautionary perspective on cross-layer design," *IEEE Wireless communications*, vol. 12, no. 1, pp. 3–11, 2005.

[70] A. Schaeffer-Filho, P. Smith, A. Mauthe, and D. Hutchison, "Network resilience with reusable management patterns," *IEEE Communications Magazine*, vol. 52, no. 7, pp. 105–115, 2014.

[71] S. Hariri, M. Eltoweissy, and Y. Al-Nashif, "Biorac: biologically inspired resilient autonomic cloud," in *Proc. of the Seventh Annual Workshop on Cyber Security and Information Intelligence Research*, 2011, pp. 1–1.

[72] T. M. Mitchell, S. Mabadevan, and L. I. Steinberg, "Leap: A learning apprentice for vlsi design," in *Machine learning*. Elsevier, 1990, pp. 271–289.

[73] P. Langley, "Relevance and insight in experimental studies," *IEEE Expert*, vol. 11, no. 5, pp. 11–12, 1996.

[74] "Celiometer," https://wiki.openstack.org/wiki/Telemetry, Accessed on 12/08/2021.

[75] G. Gardikis *et al.*, "An integrating framework for efficient nfv monitoring," in *Proc. of the IEEE NetSoft Conference and Workshops (NetSoft)*, 2016, pp. 1–5.

[76] "Promethues," https://prometheus.io/, accessed on 15/10/2021.

[77] C. Liu and otehrs, "Online arima algorithms for time series prediction," in *Proc. of the Thirtieth AAAI conference on artificial intelligence*, 2016.

[78] J. J. Faraway, *Extending the linear model with R: generalized linear, mixed effects and nonparametric regression models*. CRC press, 2016.

[79] C. S. Hood and C. Ji, "Proactive network-fault detection [telecommunications]," *IEEE Transactions on reliability*, vol. 46, no. 3, pp. 333–341, 1997.

[80] M. Shaw, "Self-healing: softening precision to avoid brittleness: position paper for woss'02: workshop on self-healing systems," in *Proc. of the first workshop on Self-healing systems*, 2002, pp. 111–114.

[81] R. Mijumbi *et al.*, "Darn: Dynamic baselines for real-time network monitoring," in *Proc. of the 4th IEEE Conference on Network Softwarization and Workshops*, 2018, pp. 37–45.

[82] M. Crosbie, G. Spafford *et al.*, "Applying genetic programming to intrusion detection," in *Working Notes for the AAAI Symposium on Genetic Programming*, 1995, pp. 1–8.

[83] N. Ye and Q. Chen, "An anomaly detection technique based on a chi-square statistic for detecting intrusions into information systems," *Quality and reliability engineering international*, vol. 17, no. 2, pp. 105–112, 2001.

[84] Z. Tan *et al.*, "A system for denial-of-service attack detection based on multivariate correlation analysis," *IEEE transactions on parallel and distributed systems*, vol. 25, no. 2, pp. 447–456, 2013.

[85] S. Staniford, J. A. Hoagland, and J. M. McAlerney, "Practical automated detection of stealthy portscans," *Journal of Computer Security*, vol. 10, no. 1-2, pp. 105–136, 2002.

[86] I. C. Paschalidis and Y. Chen, "Statistical anomaly detection with sensor networks," *ACM Transactions on Sensor Networks (TOSN)*, vol. 7, no. 2, pp. 1–23, 2010.

[87] S. S. Varadhan, "Asymptotic probabilities and differential equations," *Communications on Pure and Applied Mathematics*, vol. 19, no. 3, pp. 261–286, 1966.

[88] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *Proc. of the eighth IEEE international conference on data mining*, 2008, pp. 413–422.

[89] X. Tao *et al.*, "A parallel algorithm for network traffic anomaly detection based on isolation forest," *International Journal of Distributed Sensor Networks*, vol. 14, no. 11, p. 1550147718814471, 2018.

[90] W. Zhang, Q. Yang, and Y. Geng, "A survey of anomaly detection methods in networks," in *Proc. of the International Symposium on Computer Network and Multimedia Technology*, 2009, pp. 1–3.

[91] H. Ringberg *et al.*, "Sensitivity of pca for traffic anomaly detection," in *Proc. of the ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, 2007, pp. 109–120.

[92] R. Kumari, Sheetanshu, M. K. Singh, R. Jha, and N. Singh, "Anomaly detection in network traffic using k-mean clustering," in *Proc. of the 3rd International Conference on Recent Advances in Information Technology*, 2016, pp. 387–393.

[93] D. He *et al.*, "Software-defined-networking-enabled traffic anomaly detection and mitigation," *IEEE Internet of Things Journal*, vol. 4, no. 6, pp. 1890–1898, 2017.

[94] A. George and A. Vidyapeetham, "Anomaly detection based on machine learning: dimensionality reduction using pca and classification using svm," *International Journal of Computer Applications*, vol. 47, no. 21, pp. 5–8, 2012.

[95] I. Syarif, A. Prugel-Bennett, and G. Wills, "Unsupervised clustering approach for network anomaly detection," in *Proc. of the International conference on networked digital technologies*, 2012, pp. 135–145.

[96] D. Brauckhoff, K. Salamatian, and M. May, "Applying pca for traffic anomaly detection: Problems and solutions," in *Proc. of the International Conference on Computer Communications*. IEEE, 2009, pp. 2866–2870.

[97] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *science*, vol. 313, no. 5786, pp. 504–507, 2006.

[98] Z. Lu *et al.*, "The expressive power of neural networks: A view from the width," in *Proc. of the 31st International Conference on Neural Information Processing Systems*, 2017, pp. 6232–6240.

[99] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks*, vol. 61, pp. 85–117, 2015.

[100] J. An and S. Cho, "Variational autoencoder based anomaly detection using reconstruction probability," *Special Lecture on IE*, vol. 2, no. 1, pp. 1–18, 2015.

[101] M. Sakurada and T. Yairi, "Anomaly detection using autoencoders with nonlinear dimensionality reduction," in *Proc. of the MLSDA 2nd workshop on machine learning for sensory data analysis*, 2014, pp. 4–11.

[102] D. Gong *et al.*, "Memorizing normality to detect anomaly: Memory-augmented deep autoencoder for unsupervised anomaly detection," in *Proc. of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 1705–1714.

[103] C. Zhou and R. C. Paffenroth, "Anomaly detection with robust deep autoencoders," in *Proc. of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, 2017, pp. 665–674.

[104] M. Verleysen and D. Francois, "The curse of dimensionality in data mining and time series prediction," in *Proc. of the International work-conference on artificial neural networks*. Springer, 2005, pp. 758–770.

[105] S. Har-Peled, P. Indyk, and R. Motwani, "Approximate nearest neighbor: Towards removing the curse of dimensionality," *Theory of computing*, vol. 8, no. 1, pp. 321–350, 2012.

[106] G. Hughes, "On the mean accuracy of statistical pattern recognizers," *IEEE Trans. Inf. Theory*, vol. 14, pp. 55–63, 1968.

[107] A. Jain, A. Karandikar, and R. Verma, "An adaptive prediction based approach for congestion estimation in active queue management (apace)," in *GPorc. of the IEEE Global Telecommunications Conference*, 2003, pp. 4153–4157.

[108] A. Gulenko *et al.*, "A system architecture for real-time anomaly detection in large-scale nfv systems," *Procedia Computer Science*, vol. 94, pp. 491–496, 2016.

[109] M. Kourtis *et al.*, "Statistical-based anomaly detection for nfv services," in *Proc. of the IEEE Conference on Network Function Virtualization and Software Defined Networks*, 2016, pp. 161–166.

[110] C. Doersch, "Tutorial on variational autoencoders," *arXiv preprint arXiv:1606.05908*, 2016.

[111] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[112] ——, "An introduction to variational autoencoders," *arXiv preprint arXiv:1906.02691*, 2019.

[113] R. J. Williams, "Learning representations by back-propagating errors," *Nature*, p. 533–536, 10 1986.

[114] P. Malhotra, L. Vig, G. Shroff, and P. Agarwal, "Long short term memory networks for anomaly detection in time series," in *Proc. ogf the 23rd European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, 2015, pp. 89–94.

[115] A. Graves, S. Fernández, M. Liwicki, H. Bunke, and J. Schmidhuber, "Unconstrained online handwriting recognition with recurrent neural networks," in *Proc. of the Advances in Neural Information Processing Systems*, 2008.

[116] S. Fernández, A. Graves, and J. Schmidhuber, "An application of recurrent neural networks to discriminative keyword spotting," in *Proc. of the International Conference on Artificial Neural Networks*, 2007, pp. 220–229.

[117] Z. Zhao, W. Chen, X. Wu, P. C. Chen, and J. Liu, "Lstm network: a deep learning approach for short-term traffic forecast," *IET Intelligent Transport Systems*, vol. 11, no. 2, pp. 68–75, 2017.

[118] D. Wierstra, J. Schmidhuber, and F. Gomez, "Evolino: Hybrid neuroevolution/optimal linear search for sequence learning," in *Proc. of the 19th International Joint Conference on Artificial Intelligence*, 2005, pp. 853–858.

[119] Z. Cui *et al.*, "Deep bidirectional and unidirectional lstm recurrent neural network for network-wide traffic speed prediction," *arXiv preprint arXiv:1801.02143*, 2018.

[120] Z. Zhao *et al.*, "Lstm network: a deep learning approach for short-term traffic forecast," *IET Intelligent Transport Systems*, vol. 11, no. 2, pp. 68–75, 2017.

[121] X. Ma *et al.*, "Long short-term memory neural network for traffic speed prediction using remote microwave sensor data," *Transportation Research Part C: Emerging Technologies*, vol. 54, pp. 187–197, 2015.

[122] A. Dalgkitsis, M. Louta, and G. T. Karetsos, "Traffic forecasting in cellular networks using the lstm rnn," in *Proc. of the 22nd Pan-Hellenic Conference on Informatics*, 2018, pp. 28–33.

[123] I. Alawe *et al.*, "Improving traffic forecasting for 5g core network scalability: A machine learning approach," *IEEE Network*, vol. 32, no. 6, pp. 42–49, 2018.

[124] T. Ergen and S. S. Kozat, "Unsupervised anomaly detection with lstm neural networks," *IEEE transactions on neural networks and learning systems*, vol. 31, no. 8, pp. 3127–3141, 2019.

[125] J. Ali-Tolppa *et al.*, "Self-healing and resilience in future 5g cognitive autonomous networks," in *Proc. of the ITU Kaleidoscope: Machine Learning for a 5G Future*, 2018, pp. 1–8.

[126] Y. Dai, Y. Xiang, and G. Zhang, "Self-healing and hybrid diagnosis in cloud computing," in *Proc. of the IEEE International Conference on Cloud Computing*, 2009, pp. 45–56.

[127] G. Ciocarlie *et al.*, "On the feasibility of deploying cell anomaly detection in operational cellular networks," in *Proc of the IEEE Network Operations and Management Symposium)*, 2014, pp. 1–6.

[128] Q. Liao and S. Stanczak, "Network state awareness and proactive anomaly detection in self-organizing networks," in *Proc. of hte IEEE Globecom Workshops*, 2015, pp. 1–6.

[129] C. Zhang *et al.*, "A deep neural network for unsupervised anomaly detection and diagnosis in multivariate time series data," in *Proc. of the AAAI Conference on Artificial Intelligence*, 2019, pp. 1409–1416.

[130] S.-W. Cheng *et al.*, "Software architecture-based adaptation for grid computing," in *Proc. of the 11th IEEE International Symposium on High Performance Distributed Computing*, 2002, pp. 389–398.

[131] G. Valetto and G. Kaiser, "A case study in software adaptation," in *Proc. of the first workshop on Self-healing systems*, 2002, pp. 73–78.

[132] M. Qin *et al.*, "Machine learning aided context-aware self-healing management for ultra dense networks with qos provisions," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 12, pp. 12 339–12 351, 2018.

[133] P. Langley and H. A. Simon, "Applications of machine learning and rule induction," *Communications of the ACM*, vol. 38, no. 11, pp. 54–64, 1995.

[134] C. Sammut, S. Hurst, D. Kedzier, and D. Michie, "Learning to fly," in *Machine Learning Proceedings 1992*. Elsevier, 1992, pp. 385–393.

[135] T. M. Mitchell, S. Mabadevan, and L. I. Steinberg, "Leap: A learning apprentice for vlsi design," in *Machine learning*. Elsevier, 1990, pp. 271–289.

[136] G. Tesauro, "Reinforcement learning in autonomic computing: A manifesto and case studies," *IEEE Internet Computing*, vol. 11, no. 1, pp. 22–30, 2007.

[137] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[138] J. Pei *et al.*, "Optimal vnf placement via deep reinforcement learning in sdn/nfv-enabled networks," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 2, pp. 263–278, 2019.

[139] Z. Zhang, *et al.*, "Q-placement: Reinforcement-learning-based service placement in software-defined networks," in *Proc. of the IEEE 38th International Conference on Distributed Computing Systems*, 2018, pp. 1527–1532.

[140] Y. Jin, M. Bouzid, D. Kostadinov, and A. Aghasaryan, "Model-free resource management of cloud-based applications using reinforcement learning," in *Proc. of hte 21st Conference on Innovation in Clouds, Internet and Networks and Workshops*, 2018, pp. 1–6.

[141] C. H. T. Arteaga, F. Risso, and O. M. C. Rendon, "An adaptive scaling mechanism for managing performance variations in network functions virtualization: A case study in an nfv-based epc," in *Proc. of the 13th International Conference on Network and Service Management*, 2017, pp. 1–7.

[142] E. H. Bouzidi, A. Outtagarts, and R. Langar, "Deep reinforcement learning application for network latency management in software defined networks," in *Proc. of the IEEE Global Communications Conference*, 2019, pp. 1–6.

[143] T. V. Phan *et al.*, "Q-mind: Defeating stealthy dos attacks in sdn with a machine-learning based defense framework," in *Proc. of the IEEE Global Communications Conference*, 2019, pp. 1–6.

[144] L. S. Sampaio *et al.*, "Using nfv and reinforcement learning for anomalies detection and mitigation in sdn," in *Proc. of the IEEE Symposium on Computers and Communications*, 2018, pp. 00 432–00 437.

[145] H. Yu *et al.*, "A survey of trust and reputation management systems in wireless communications," *Proceedings of the IEEE*, vol. 98, no. 10, pp. 1755–1772, 2010.

[146] H. Chen *et al.*, "Reputation-based trust in wireless sensor networks," in *Proc. of the International Conference on Multimedia and Ubiquitous Engineering*, 2007, pp. 603–607.

[147] T. Ciszkowski *et al.*, "Towards quality of experience-based reputation models for future web service provisioning," *Telecommunication Systems*, vol. 51, no. 4, pp. 283–295, 2012.

[148] Q. He, D. Wu, and P. Khosla, "Sori: A secure and objective reputation-based incentive scheme for ad-hoc networks," in *Proc. of the IEEE Wireless Communications and Networking Conference*, vol. 2, no. 1, 2004, pp. 825–830.

[149] J. Ahn, M. Park, H. Shin, and J. Paek, "A model for deriving trust and reputation on blockchain-based e-payment system," *Applied Sciences*, vol. 9, no. 24, p. 5362, 2019.

[150] B. Isong *et al.*, "Trust establishment framework between sdn controller and applications," in *Proc. of the 18th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*, 2017, pp. 101–107.

[151] L. Wen *et al.*, "Distributed bayesian network trust model in virtual network," in *Proc. of the Second International Conference on Networks Security, Wireless Communications and Trusted Computing*, 2010, pp. 71–74.

[152] S. Betgé-Brezetz, G.-B. Kamga, and M. Tazi, "Trust support for sdn controllers and virtualized network applications," in *Proc. of the 2015 1st IEEE Conference on Network Softwarization*, 2015, pp. 1–5.

[153] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[154] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of control, signals and systems*, vol. 2, no. 4, pp. 303–314, 1989.

[155] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994.

[156] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, pp. 1735–80, 12 1997.

[157] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[158] H. Hasselt, "Double q-learning," *Advances in neural information processing systems*, vol. 23, no. 1, pp. 2613–2621, 2010.

[159] A. Diamanti, J. M. S. Vilchez, and S. Secci, "Lstm-based radiography for anomaly detection in softwarized infrastructures," in *Proc. of the 32nd International Teletraffic Congress*, 2020, pp. 28–36.

[160] ——, "The syrroca ai-empowered network automation platform," in *2021 24th Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*, 2021, pp. 140–142.

[161] J. Rubio-Loyola *et al.*, "Scalable service deployment on software-defined networks," *IEEE Communications Magazine*, vol. 49, no. 12, pp. 84–93, 2011.

[162] M. Behringer *et al.*, "A reference model for autonomic networking," *IETF Internet Draft*, May 2018.

[163] "Zero-touch network and service management requirements," https://www.etsi.org/technologies/zero-touch-network-service-management.

[164] ETSI GR ENI 004 V1.1.1, "Experiential networked intelligence; terminology for main concepts," October 2019.

[165] A. a. Boubendir, "Network slice life-cycle management towards automation," in *Proc. of the IFIP/IEEE Symposium on Integrated Network and Service Management*, 2019, pp. 709–711.

[166] V. Q. Rodriguez, F. Guillemin, and A. Boubendir, "5g e2e network slicing management with onap," in *Proc. of the 23rd Conference on Innovation in Clouds, Internet and Networks and Workshops*, 2020, pp. 87–94.

[167] A. Boubendir *et al.*, "5g edge resource federation: Dynamic and cross-domain network slice deployment," in *Proc. of the th IEEE Conference on Network Softwarization and Workshops*, 2018, pp. 338–340.

[168] "The open radio access network (oran) alliance," https://www.o-ran.org, accessed on 15/10/2021.

[169] "Open network automation platform (onap)," https://www.onap.org., accessed on 15/10/2021.

[170] "Syrroca github repositery," https://github.com/SYRROCA.

[171] A. Packet, "Session initiation protocol (sip) info method and package framework," 2011.

[172] "Openimscore," http://openimscore.sourceforge.net/, accessed on 15/10/2021.

[173] "Kubernetes," https://github.com/kubernetes/kubernetes/, accessed on 15/10/2021.

[174] "Sipp," http://sipp.sourceforge.net/, accessed on 15/10/2021.

[175] P. O. V. De Melo *et al.*, "Surprising patterns for the call duration distribution of mobile phone users," in *Proc. of the Machine Learning and Knowledge Discovery in Databases, European Conference*, 2010, pp. 20–24.

[176] "Nodeexporter," https://github.com/prometheus/node_exporter, accessed on 15/10/2021.

[177] "Cadvisor," https://github.com/google/cadvisor, accessed on 15/10/2021.

[178] K. Fokianos and B. Kedem, "Regression theory for categorical time series," *Statistical science*, vol. 18, no. 3, pp. 357–376, 2003.

[179] G. M. Davis and K. B. Ensor, "Multivariate time-series analysis with categorical and continuous variables in an lstr model," *Journal of Time Series Analysis*, vol. 28, no. 6, pp. 867–885, 2007.

[180] W. W. Wei, *Time series analysis*. Pearson College Div, 2006.

[181] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[182] K. Pearson, "X. on the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 50, no. 302, pp. 157–175, 1900.

[183] R. L. Plackett, "Karl pearson and the chi-squared test," *International Statistical Review*, no. 1, pp. 59–72, 1983.

[184] J. Brownlee, *Deep learning for time series forecasting: predict the future with MLPs, CNNs and LSTMs in Python*. Machine Learning Mastery, 2018.

[185] D. R. Wilson and T. R. Martinez, "The general inefficiency of batch training for gradient descent learning," *Neural networks*, vol. 16, no. 10, pp. 1429–1451, 2003.

[186] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. of hte International conference on machine learning*, 2015, pp. 448–456.

[187] G. E. Hinton *et al.*, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012.

[188] N. Srivastava *et al.*, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[189] C. J. Willmott and K. Matsuura, "Advantages of the mean absolute error (mae) over the root mean square error (rmse) in assessing average model performance," *Climate research*, vol. 30, no. 1, pp. 79–82.

[190] R. J. Hyndman and A. B. Koehler, "Another look at measures of forecast accuracy," *International journal of forecasting*, vol. 22, no. 4, pp. 679–688, 2006.

[191] T. Chai and R. R. Draxler, "Root mean square error (rmse) or mean absolute error (mae)?–arguments against avoiding rmse in the literature," *Geoscientific model development*, vol. 7, no. 3, pp. 1247–1250, 2014.

[192] N.-B. Heidenreich, A. Schindler, and S. Sperlich, "Bandwidth selection for kernel density estimation: a review of fully automatic selectors," *AStA Advances in Statistical Analysis*, vol. 97, no. 4, pp. 403–433, 2013.

[193] L. Song, "Cognitive networks: standardizing the large scale wireless systems," in *Proc. of the 5th IEEE Consumer Communications and Networking Conference*, 2008, pp. 988–992.

[194] H. a. Alaiz-Moreton, "Multiclass classification procedure for detecting attacks on mqtt-iot protocol," *Complexity*, vol. 1, 2019.

[195] NGMN, "5g white paper," vol. 1, February 2015.

[196] "Open5gs," https://github.com/open5gs/open5gs, accessed on 15/10/2021.

[197] D. Clark, R. Braden, and S. Shenker, "Integrated services in the internet architecture: An overview," *RFC 1633*, 1994.

[198] L. Z. al., "Resource reservation protocol (rsvp)–version 1 functional specification," *RFC 2205*, 1997.

[199] C. S. L. Sony and K. Cho, "Traffic data repository at the wide project," *In Proc. of the USENIX Annual Technical Conference: FREENIX Track*, pp. 263–270, 2000.

[200] A. F. al., *SMOTE for learning from imbalanced data: progress and challenges, marking the 15-year anniversary.* Journal of artificial intelligence research, 2018.

[201] P. J. Rousseeuw, *Silhouettes: a graphical aid to the interpretation and validation of cluster analysis.* Journal of computational and applied mathematics, 1987.

[202] J. F. al., "Prefix-preserving ip address anonymization," *In Computer Networks 46*, pp. 253–272, 2004.

[203] S. Raschka, *Python machine learning.* Packt Publishing Ltd, 2015.

[204] T. Afonja, *Accuracy Paradox.* Towards Data Science, 2017.

[205] N. V. C. al., "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research 16*, pp. 321–357, 2002.

[206] A. T. al., "Linear discriminant analysis: A detailed tutorial," *AI communications 30*, vol. 2, pp. 169–190, 2017.

[207] D. Arthur and S. Vassilvitskii, "k-means++: The advantages of careful seeding," in *In Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, 2007, pp. 1027–1035.

le cnam

Alessio DIAMANTI

A novel network automation architecture: from anomaly detection to dynamic reconfiguration

HESAM UNIVERSITÉ

Résumé : Les technologies de softwarisation des réseaux entraînent de nouvelles architectures de réseau qui remettent en question les systèmes de gestion des défaillance existants et la caractérisation de la résilience. En effet, la coordination entre les différents composants logiciels pour, par exemple, l'orchestration, la commutation et la gestion des machines virtuelles et des conteneurs, implique différents points de supervision et de nouvelles sources de défaillance et de bogues. Dans cette thèse, nous proposons un framework d'automatisation de réseau qui détecte les anomalies et caractérise l'état de résilience d'un service de réseau virtualisé. Un algorithme basé sur les Long Short Term Memory Autoencoder analyse une série temporelle multidimensionnelle construite à partir de centaines de métriques collectées au niveau des couches physique, virtuelle et de service. Il apprend les conditions de fonctionnement nominales de l'infrastructure, sur la base desquelles les déviations (anomalies) par rapport à la référence apprise sont détectées et analysées. Le framework produit une caractérisation des déviations utilisée pour élaborer le graphe d'état et la visualisation sous forme de radiographie. Tandis que cette dernière visualise de manière compacte la propagation des anomalies à travers les trois couches composant un réseau virtualisé, le graphe d'état vise à établir l'état de résilience de la plateforme comme base d'un algorithme de réorchestration qui s'appuie sur une nouvelle technique de gestion de la résilience basée sur la réputation. Le framework est implémenté et validé par des tests expérimentaux sur la plateforme Kubernetes hébergeant un services de cœur de réseau virtualisé conteneurisé et open-source.

Mots clés : Automatisation des réseaux, Détection d'anomalies, Reconfiguration, NFV, Machine learning

Abstract : Legacy and novel network services are expected to be migrated and designed to be deployed in fully virtualized environments which lead to novel network architectures that challenge legacy fault management systems and resilience characterization. Indeed, the coordination among the different software components for, e.g., orchestration, switching, and virtual machine and container management creates different monitoring points, besides novel sources of faults and bugs. In this thesis, we propose a network automation framework that detects anomalies and characterizes the resiliency state of a virtualized network service. A Long-Short-Term-Memory-Autoencoder-based algorithm analyzes a multidimensional time-series built from hundreds of metrics collected at the physical, virtual, and service layers. It learns the nominal working conditions of both the infrastructure and the service, and for each type of resource (i.e., CPU, network, memory, and disk); it then detects and analyzes deviations (anomalies) from the learned reference. The produced deviations characterization is finally used to generate both the transition state graph and the innovative radiography visualization. The latter compactly visualizes the propagation of anomalies across all the layers down from the physical and up to the service, highlighting the temporal evolution as well. The former aims at establishing the virtualized platform state as the basis for a re-orchestration algorithm that leverages a novel reputation-based resiliency management technique. We implement and validate the proposed framework through experimental tests on the Kubernetes platform hosting a containerized, open-source, and virtualized network core service.

Keywords : Network automation, Anomaly detection, Reconfiguration, NFV, Machine learning