Utilisation Profiles of Bridging Function Chain for Healthcare Use Cases

Kassem, J.A.; Belloum, A.; Müller, T.; Grosso, P.

DOI
10.1109/eScience55777.2022.00085

Publication date
2022

Document Version
Final published version

Published in
eScience '22 : Democratizing science : 2022 IEEE 18th International Conference on e-Science

Utilisation Profiles of Bridging Function Chain for Healthcare Use Cases

1st Jamila Alsayed Kassem
Informatics Institute, MNS group
University of Amsterdam
Amsterdam, Netherlands
j.alsayedkassem@uva.nl

2ndAdam Belloum
Informatics Institute, MNS group
University of Amsterdam
Netherlands, Amsterdam
A.S.Z.Belloum@uva.nl

3rd Tim Müller
Informatics Institute, CCI group
University of Amsterdam
Netherlands, Amsterdam
t.muller@uva.nl

4th Paola Grosso
Informatics Institute, MNS group
University of Amsterdam
Amsterdam, Netherlands
p.grosso@uva.nl

***Abstract*—On the road towards personalised medicine, one of the main challenges is to enforce security and network low-level policies to secure data-sharing. The proposed dynamic framework defines the topology of the service chains to enforce network and security policies by instantiating Virtual Network Functions (VNF's) on the fly via light-weight and easily-deployable containers. In this paper, we profile the resource utilisation of chained VNF's deployed to enable data movement within different healthcare use cases. We provide example configurations that map to a couple of use cases (e-Health record query and heath data streaming), then we monitor and collect CPU utilisation of the different VNF compositions. In the considered policies we can: discard flow, protect (encrypt) and transmit, or allow with no protection. To enforce each policy, we deploy a firewall function (relatively heavy-weight function), encryption function, and a decryption function (light-weight stream cipher). As a result, we analyse the behaviour of the VNF services with various setups, and then we aim to further use this analysis to build the placement heuristic according to available and trusted clusters resources. Subsequently, we will recommend heuristic-based placement based on collected profiling data of the resource usage and limits for high availability, optimal performance, and minimal resource waste.**

***Index Terms*—Virtual Network Function, Programmable Infrastructures, Network Policy, Resource profiling.**

I. Introduction

In line with Next-generation networks, network functions are getting increasingly virtualised and highly programmable. Network Function Virtualisation (NFV) decouples functions; such as packet encryption and firewalling; from specialised hardware to make services deployable on general-use virtual servers [1]. This allows on demand management, dynamic shipment and placement of services, and can potentially provide reliable network performance.

NFV offers the framework to configure, chain, manage, and orchestrate functions according to agreed network policies. Furthermore, containerising these functions can accomplish fast deployment, high reusability, and low setup overhead [2]. The controlled service scheduling and placement, namely containerised NFV-based (CNFV) services, is a functionality that aligns with what we are trying to accomplish in supporting different health use cases. In the EPI[1] project, we identify a number of use cases that are representative of the broad applicability of personalised medicine in the healthcare domain. We simulate two of the use cases by simulating different workloads and profiling of resources:

- **Use case 1**: Hosting a centralised health data registry and maintaining health data entries of different healthcare institutions, and the user can query an entry in the dataset.
- **Use case 2**: Streaming data via IoT devices and wearable to monitor the users' health and provide timely intervention.

The EPI Framework employs CNFV to implement network service chains, and we configure the setup to adapt to different healthcare use cases' policies and requirements. The policies describe the communication constraints, traffic filtering rules, trust, and the required Network Functions that needs to be instantiated and deployed. The framework can also be used with use cases other than healthcare by using the same framework's handles to generally translate and map policies to VNF chain. To implement the security services, we need middle-boxes hosting the NF's; known as proxies. The proxied services can vary from lightweight to heavyweight functions depending on the type of function and the implementation's optimisation. Moreover, the resource consumption behaviour also depends on the expected workload whilst running a specific use case. Hence, we need to build a profile of the resources required under labelled use cases and the associated network policy. Furthermore, we have to allocate sufficient computing resources to ensure smooth deployment and high

The EPI project is funded by the Dutch Science Foundation in the Commit2Data program

[1] https://enablingpersonalizedinterventions.nl/

978-1-6654-6124-5/22/$31.00 ©2022 IEEE
DOI 10.1109/eScience55777.2022.00085

availability throughout the execution of the application. The challenge that still remains is the optimal and adaptive placement and configuration of these services to available resources across trusted clusters of proxy nodes.

II. Related Work

With similar research efforts, [3] [4] propose policy enforcement systems using SDN tools (OpenFlow), to ”steer traffic” through a policy-defined service chain. To reduce the resources needed by the middle boxes, they define ways to optimise the switch table sizes and flow control rules. On the other hand, employing OpenFlow switches can potentially introduce new challenges, such as dedicating specialised hardware and having to configure and add SDN components to the infrastructure.

Subsequently, adding virtualisation capabilities is needed, and resource allocation must be effectively orchestrated and managed in order to avoid over/under-provisioning, and to maintain end-to-end latency that are equivalent to those seen in conventional networks. It is crucial to ensure the network services resilience, knowing that the implementation can get increasingly complex (like a firewalling with intensive rules, capturing application level semantics, and potentially deep packet inspections). According to [5], overloading middle boxes (e.g. proxies) is a typical reason for service failure.

A plethora of work is done on NFV chaining and placement, but different strategies are employed. As an example, the MIDAS [6] framework places the services chain according to the cross-border on-path strategy, where each service is location specific (*e.g.* a web proxy needs to be place near the end-client). On the other hand, when the path is loosely controlled, then the placement really depends on the hardware resources available, and that is implemented using varying algorithms (*e.g.* the least busy host placement algorithm [7]). Others consider QoS-driven optimisation placement of the NFV. The used methods vary from using heuristic algorithms [8], Linear programming, and ML-based tools (*e.g.* Reinforcement learning).

In an effort to optimally deploy and maintain the CNFV chains, we add with this work-in-progress to existing literature the defined deployment stages after a network policy is formalised. The placement is decided on based on available trusted proxy clusters matched with the profiled data, knowing the labelled use case running. We showcase the exploratory profiling results for designing the heuristic algorithm.

III. The Function Chains Deployment

In previous work [9], we proposed the EPI framework (EPIF), a dynamic framework that automates the programming of the underlying networks to secure health data-sharing. Data-sharing is secured by orchestrating and managing CNFV; called *Bridging Functions* (BF); to add security value on each communicating node, irrespective of each node’s computing capabilities. The BFs can be instantiated and chained to form a *Bridging Function Chain* (BFC) and enforce increasingly complex policies.

The *adaptation* of the underlying networks is done after querying network policies and translating them into setup actions, for example, party A can talk to party B if A is able to encrypt and decrypt via a stream cipher. The automated setup of the infrastructure is also required to achieve reachability of the end-points nodes, optimal security across collaborating domains, reasonable network performance, bridging services availability, hardware selection and scalability, and ultimately, abiding to policy requirements.

EPIF has different components that are crucial to automate the data sharing processes between participating parties. The main components of the framework are the application orchestrators, infrastructure orchestrator, the policy management system, and the logic area generator. The two orchestrators are decoupled (conceptually) to orchestrate and manage different types of functions; the workflow functions, and the network functions, respectively. The policy management system is used as a reasoner to query policies and maintain agreement. The logic area generator [10] is in place to assign security areas to end nodes, and subsequently deduce security requirements and policies. The proxy node acts as framework actor, where the BFC pool is hosted and instantiated when ordered by the orchestrator.

After resolving the BFC rule per source-destination flow, the EPIF instantiates and deploys the services chain to secure connection according to the set policy. The challenge of optimally provisioning these services remains, and so we define with this work three stages for optimal placement and performance of the chain. Fig. 1 illustrates these steps, starting with introducing a new healthcare use case with the expected associated BFC workload. Then we start the profiling stages where we test the setup, collect resource metrics via the metric server, calculate statistical information, and store the labelled data for the next stage. The Heuristic placement queries the available *trusted* proxy clusters via the cluster manager, and matches the required resources with the currently available resources. Trust is modelled according to the node’s competence, and the contractual agreement of the institution it runs within. The orchestrator controls the placement of the service chain accordingly to maximise trust and resource utilisation. This would achieve optimal placement of BFC (or semi-optimal, considering inaccurate profiling in case of small number of test-runs with high standard deviations), and serves as booster to the Q-learning agent in the next stage. Next, the run-time maintenance utilises a Reinforcement Learning agent, and is asked to monitor QoS metrics (latency, throughput, etc.) and reactively scale in/out. The placement of the replicated scaled out services can be done across the multiple trusted proxy clusters (not just the chosen placement cluster).

A simple BFC deployment is illustrated in Fig. 2, where we have two types of nodes:

- The master node: acting as the infrastructure orchestrator where we can configure, deploy, and orchestrate microservices
- The worker node: acting as the proxy node hosting and running the BFC. The worker nodes are spread over two

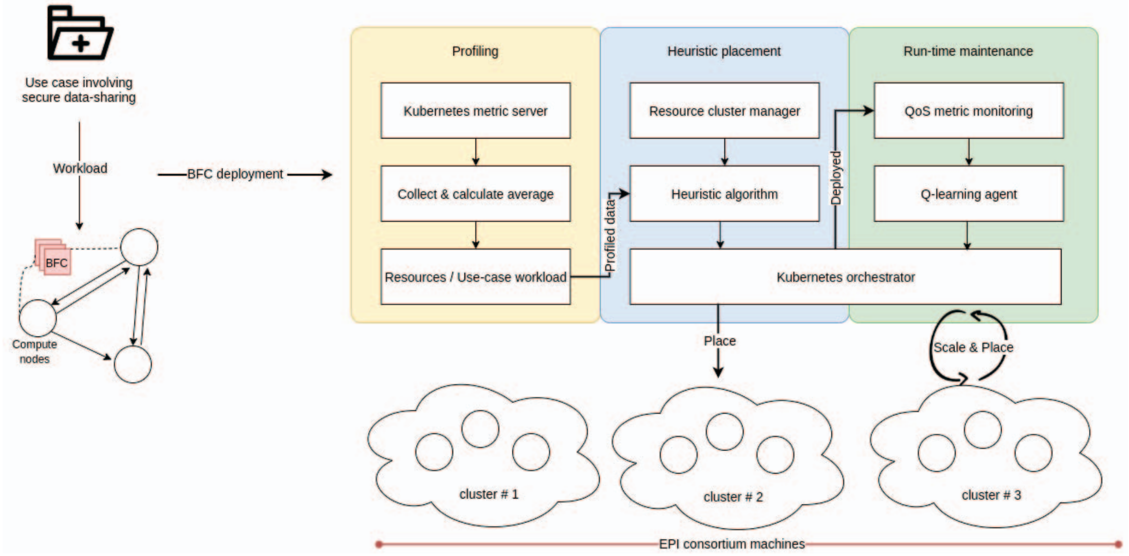

Fig. 1. The BFC deployment stages after introducing a new health use case:Profiling, Heuristic placement, run-time maintenance.

different clusters A and B both running on UvA machines in this case.

Fig. 2 also shows the example BFC implementations and the proxy service. The proxy service has an important functionality of traffic intercepting and redirecting to route through the BFC. With that we eliminate the need to manually plan, compose, and configure routes across the service chain (example: $Client \rightarrow NF_1 \rightarrow NF_2 \rightarrow NF_3 \rightarrow Server$), by introducing the SOCKS(5|6) [11] [12] proxy that intercepts and reroutes packets seamlessly to the end-points nodes.

We focus in this paper on the first stage, and we populate the profile data storage with resource consumed running use case 1 and 2.

IV. Experiments

We design the following experiment to evaluate CPU usage of different BF chains. We do not consider memory because our cases do not pose a strain on the resources. That is due to the type of functions implemented. We deploy this framework (shown is Fig. 2) using four identical Ubuntu 18.04 VMs with 2 cores, 2 GB RAM, and 20 GB HD. In our effort to run this in a controlled environment, we containerise all services and orchestrate with Kubernetes. All the needed services run on the worker nodes, and that is made available via dedicating an external port that can be pinged and reached from outside the node.

The SOCKS6 proxy we implement is a standard proxying protocol for TCP and UDP connections. SOCKS6 is an optimised iteration of SOCKS5, and it introduces minor tweaks to the *Authentication handshake*. The stateful Python-based firewall function cross-checks the packets across a couple of rules to either accept or reject traffic, then redirects it back to the proxy to route to the next stop (next BF or destination). We implemented the encryption and decryption functions with ChaCha20 stream cipher [13]. It is a light-weight function that improves on the Salsa algorithm [14] by increasing per-round diffusion without decreasing the performance.

The metric server is a cluster-wide data aggregator of resource usage and will scrape the relevant resource metrics of each pod. We use the metric server to gather 100 samples of the CPU usage / pod, and then calculate statistical values (average and standard deviation error). All relevant code is made available on GitHub[2] for reproducibility purposes.

We assign a fixed CPU request and limit to all containers of 500milliCores and 600milliCores, respectively. We run four different configurations. The first use case is the one with lower sending rate of 100kB/s, and we rerun this experiment twice with 1 then 10 concurrent clients. Similarly, we also run the same experiment with use case simulating a heavier workload of 1000kB/s. We purposely generate traffic that will be accepted and forwarded, because we need traffic to propagate across the chain

V. Profiling Results

The varying composition, length, and order of the functions within a chain can result in different CPU consumption per pod. That is further apparent in Fig. 3, Fig. 4, Fig. 5, Fig. 6.

Considering the proxy service, the CPU utilisation does not change across topologies. This is due to the minimal proxy's functionality of redirecting traffic according to the chain, and the fact that most processing is done on the NIC (Network Interface Card).

While as, the other services are affected by the use case applied and the number of concurrent users using the system. First, the firewall service has the highest CPU utilisation being the heavier-weight function. Compared to the other topologies,

[2] https://github.com/epi-project/EPI-kube-scaling

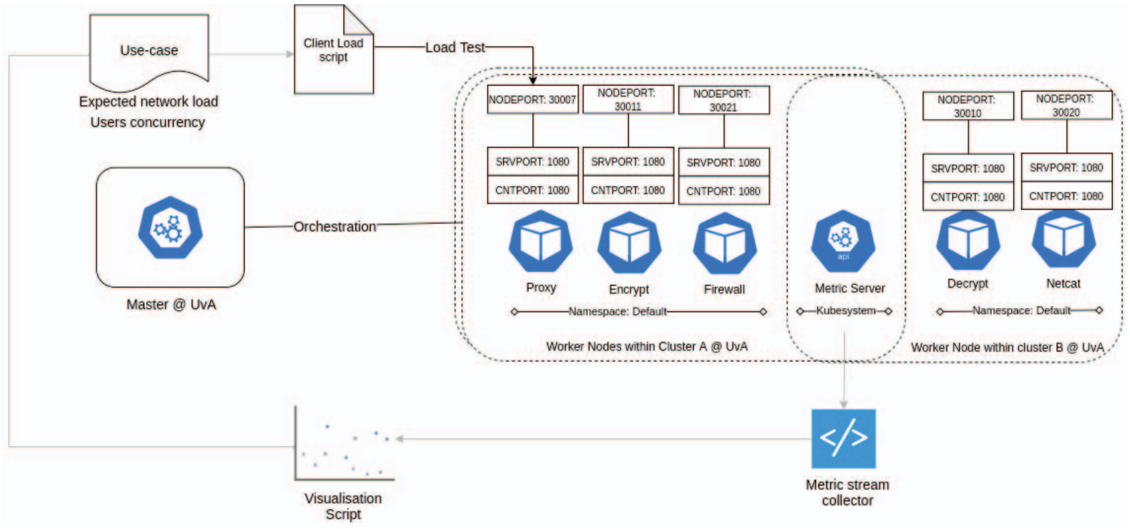

Fig. 2. The setup of the EPI function chains at UvA with one master node as the orchestrator and the worker nodes across cluster A & B hosting the topology of BFC.

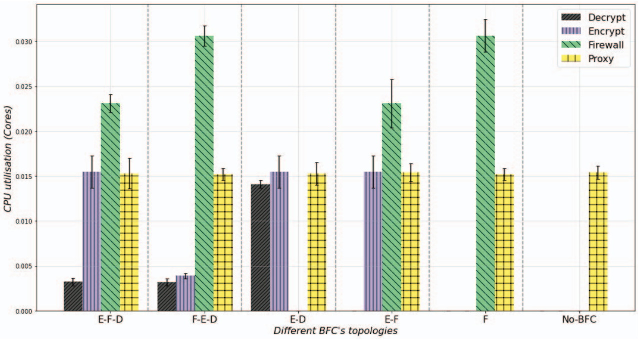

Fig. 3. Use case 1: CPU utilisation per pod and the throughput at the end server with one client generating 1000 kB/s traffic, and grouped by service BFC topologies composed of encryption (E), decryption (D), and firewall (F) services.

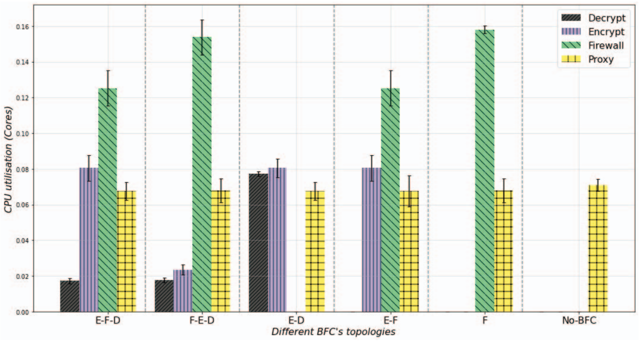

Fig. 4. Use case 1: CPU utilisation per pod and the throughput at the end server with ten clients generating 1000 kB/s traffic, and grouped by service BFC topologies composed of encryption (E), decryption (D), and firewall (F) services.

the consumption average is higher within the F-E-D and F chains, and that is related to the first order of the service.

Second, the encryption and decryption services expectedly have approximately equal CPU usage across the F-E-D and E-D topologies. That is due to similar compiling instructions and code implementation. Unlike with the E-F-D topology, where the decryption service has a much lower CPU average. This is caused by the ordering of the firewall in the middle of the chain. This reduction is caused by the different processing rate at the firewall, and this can differ with different implementations (larger number of rules, code optimisation, etc.) Furthermore, the reduction is also affecting the needed CPU power by both services (encryption and decryption) in the F-E-D chain. Similarly, the causing factor being the firewall placement at the beginning of the chain.

Intuitively, the CPU utilisation of the services varies with the different use case, and increases with more clients, while the behaviour is consistent throughout the experiments. These values are generated automatically after initiating a profiling stage, and this serves as an input to the next stages.

To prove the relation between the BFC topologies' composition and the received packets' rate reduction, we measure packets received per second at the Socat end-server. Fig. 7 shows that the throughput reduction percentage / topology while running the four experiments. The topology chains with no firewall have a much higher throughput than the other setups.

This throughput is slightly lower than having no BFC at all. To counter this effect, we employ QoS-based scaler that monitors the throughput across each function, and scale out (or in) by replicating the service (horizontal scale) and place the pods according to heuristic.

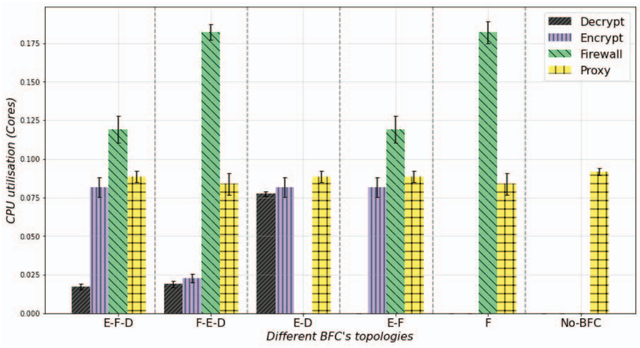

Fig. 5. Use case 2: CPU utilisation per pod and the throughput at the end server with one client generating 1000 kB/s traffic, and grouped by service BFC topologies composed of encryption (E), decryption (D), and firewall (F) services.

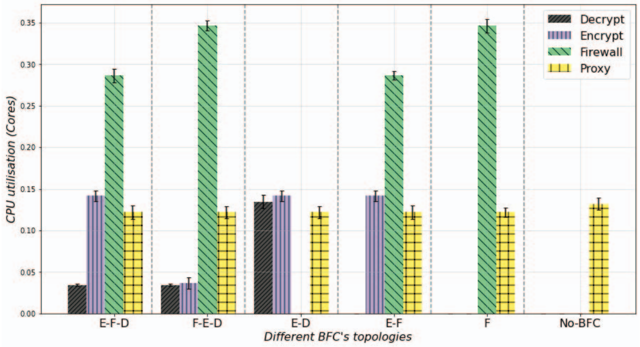

Fig. 6. Use case 2: CPU utilisation per pod and the throughput at the end server with ten clients generating 1000 kB/s traffic, and grouped by service BFC topologies composed of encryption (E), decryption (D), and firewall (F) services.

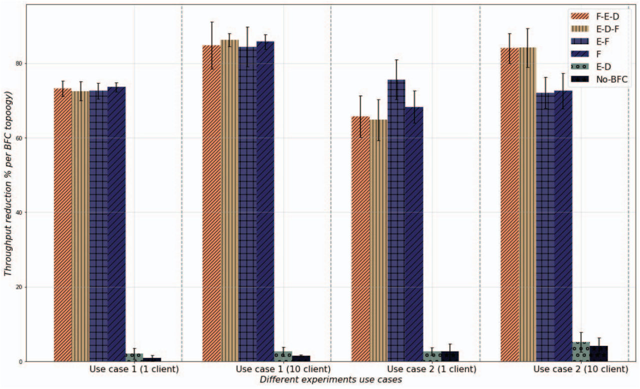

Fig. 7. The monitored throughput reduction profile for the different use cases with different BFC setups

VI. Conclusion and Future Work

When running healthcare use cases and data-sharing applications, it is crucial to enforce policies; namely network and security policies. The EPI framework handles querying policies, translating it to network functions setups, and enforce traffic redirection through that route. The setup of these functions is done through stages within the EPI framework: profiling use cases, resources aware placement, run-time network performance maintenance to minimise overhead and by that minimising throughput reduction/ service.

We extend on the first stage of profiling, and we experiment with two exemplary healthcare use cases, and collect the resources utilised per use case, and populate the statistical data table accordingly. Moreover, the composition of BFC topologies also plays a role in CPU workload / service. The BFC topology with heavier, less optimised implementations in between services result in reduction of the received packets rate at the next BF.

Our goal is to relate the profiling results to the applied healthcare use case by estimating the resource usage trends, and based on that we configure BFC service description with expected utilisation interval. We can use this information to assign services to trusted clusters, correlating available resources with the consumption rate.

In this paper, we provide insights on the resource utilisation trends across VNFs chains. Moreover, we build the handles to monitor services' performance, and that's the groundwork for the other stages of optimal deployment. We also mention the concept of use case-based dynamic trust, and that will be further explored and modelled in the upcoming work as a heuristic deciding factor. In our future studies, we aim to use these data to automatically place and the service across trusted clusters and update its configuration. We will also introduce the scalers to the setup. The scaler proposed is a Q-learning-base horizontal scaler. We plan to compare the EPI deployment's stages and configuration methods to a random placement on any cluster. The comparison will be done according to the QoS performance of the service chains, packet loss, and the deployment success rate.

We will also extend on the network orchestrator's collaboration with the policy management system and application orchestrator. This collaboration will promote high programmability and automatic adaptiveness of the infrastructure. Finally, we aim to deploy the framework with real test-beds and utilise this by introducing real health data, and enforcing different policies and service topologies accordingly.

References

[1] Jun Wu, Zhifeng Zhang, Yu Hong, and Yonggang Wen, "Cloud radio access network (c-ran): a primer," *IEEE Network*, vol. 29, no. 1, pp. 35–41, 2015.

[2] Richard Cziva and Dimitrios P. Pezaros, "Container network functions: Bringing nfv to the network edge," *IEEE Communications Magazine*, vol. 55, no. 6, pp. 24–31, 2017.

[3] John Bellessa, Evan Kroske, Reza Farivar, Mirko Montanari, Kevin Larson, and Roy H. Campbell, "Netodessa: Dynamic policy enforcement in cloud networks," in *2011 IEEE 30th Symposium on Reliable Distributed Systems Workshops*, 2011, pp. 57–61.

[4] Lin Cui, Fung Po Tso, and Weijia Jia, "Heterogeneous network policy enforcement in data centers," in *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, 2017, pp. 552–555.

[5] Phillipa Gill, Navendu Jain, and Nachiappan Nagappan, "Understanding network failures in data centers: Measurement, analysis, and implications," *SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. 350–361, aug 2011.

[6] Ahmed Abujoda and Panagiotis Papadimitriou, "Midas: Middlebox discovery and selection for on-path flow processing," in *2015 7th International Conference on Communication Systems and Networks (COMSNETS)*, 2015, pp. 1–8.

[7] Stuart Clayman, Elisa Maini, Alex Galis, Antonio Manzalini, and Nicola Mazzocca, "The dynamic placement of virtual network functions," in *2014 IEEE Network Operations and Management Symposium (NOMS)*, 2014, pp. 1–9.

[8] Abeer A. Z. Ibrahim, Fazirulhisyam Hashim, Nor K. Noordin, Aduwati Sali, Keivan Navaie, and Saber M. E. Fadul, "Heuristic resource allocation algorithm for controller placement in multi-control 5g based on sdn/nfv architecture," *IEEE Access*, vol. 9, pp. 2602–2617, 2021.

[9] Jamila Alsayed Kassem, Onno Valkering, Adam Belloum, and Paola Grosso, "Epi framework: Approach for traffic redirection through containerised network functions," in *2021 IEEE 17th International Conference on eScience (eScience)*, 2021, pp. 80–89.

[10] Jamila Alsayed Kassem, Cees De Laat, Arie Taal, and Paola Grosso, "The epi framework: A dynamic data sharing framework for healthcare use cases," *IEEE Access*, vol. 8, pp. 179909–179920, 2020.

[11] Marcus D. Leech, "SOCKS Protocol Version 5," RFC 1928, Mar. 1996.

[12] Vladimir Olteanu and Dragos Niculescu, "SOCKS Protocol Version 6," Internet-Draft draft-olteanu-intarea-socks-6-11, Internet Engineering Task Force, Nov. 2020, Work in Progress.

[13] Yoav Nir and Adam Langley, "ChaCha20 and Poly1305 for IETF Protocols," RFC 7539, May 2015.

[14] Simon Josefsson, Joachim Strombergson, and Nikos Mavrogiannopoulos, "The Salsa20 Stream Cipher for Transport Layer Security," Internet-Draft draft-josefsson-salsa20-tls-04, Internet Engineering Task Force, Nov. 2013, Work in Progress.