CHIMA: a Framework for Network Services Deployment and Performance Assurance

Elia Battiston*, Daniele Moro†, Giacomo Verticale*, Antonio Capone*
* Politecnico di Milano, Italy
elia.battiston@mail.polimi.it, giacomo.verticale@polimi.it, antonio.capone@polimi.it
† Open Networking Foundation
daniele@opennetworking.org

***Abstract*—Network Function Virtualization has dramatically increased the flexibility in the deployment of network services, however the execution of virtual functions on compute nodes equipped with general purpose hardware can result in worse performance compared to the middleboxes they aim to replace. The use of programmable network hardware to perform part of the processing at line rate can drastically increase the throughput while retaining the flexibility.**

This work presents a new framework, called CHIMA, which extends the capabilities of other frameworks proposed in the literature for the deployment of heterogeneous Service Function Chains (SFCs). Heterogeneous SFCs comprise a combination of virtual functions meant to be executed in containers running on general purpose hardware and of functions for programmable switches written using the P4 language. CHIMA exploits programmable data planes to perform real time monitoring of the services through In-band Network Telemetry and uses the collected information to guarantee the requested levels of performance by redeploying and rerouting sections that are affected by adverse conditions, allowing applications with critical requirements to be deployed as SFCs.

The solution has been tested by emulating various topologies and services on the FOP4 platform with bmv2 switches. The analysis shows that the system is capable of detecting faults in the order of hundreds of milliseconds, and the overhead it causes in the process of redeployment is negligible compared to the startup time of functions. Measurements also reveal that the current bottleneck for the runtime relocation of heterogeneous functions is the redeployment and reconfiguration of P4 programs.

***Index Terms*—Network Function Virtualization, Service Function Chains, Programmable Data Planes, In-band Network Telemetry**

I. Introduction

The use of Virtual Network Functions (VNFs) makes it easier and faster to manage the provisioning of network services. This, however, introduces a tradeoff between flexibility and performance since the execution of packet processing logic on regular CPUs is less efficient, both in terms of throughput and power consumption. Recent advances in the field of Programmable Data Planes, and In-Network Computing in particular, showed that offloading sections of these services to programmable switches is a viable way to eliminate the tradeoff, bringing back the processing performance to the level that is offered by specialized hardware middleboxes. In addition, the ability to define arbitrary logic with the P4 language enables the development of other features alongside regular forwarding and processing, such as the real time monitoring of flows with In-band Network Telemetry (INT) [1]. Such techniques can be exploited for more than diagnosis of errors or logging, since real time feedback on the performance of a service could allow an orchestrator to take immediate action in response to congestion or faults.

This paper builds on the existing techniques for the deployment of heterogeneous service function chains, such as the one proposed by Moro *et al.* [2], and describes and assesses the CHIMA (CHain Installation, Monitoring and Adjustment) framework, which introduces the following novelties:

- The possibility of specifying the performance requirements, such as the maximum latency and the maximum jitter, either for sections of the service or for its entirety.
- The real-time monitoring of the service's communications with In-band Network Telemetry.
- An extension mechanism to INT for measuring the execution time of virtual network functions, which results in a better knowledge of the latency experienced by portions of the service.
- A mechanism to detect events that result in a failure to meet the performance targets and their solution through the rerouting or redeployment of the affected components.

We develop a prototype of CHIMA and use it to evaluate the feasibility of the proposed solution [3].

The remainder of the paper is structured as follows. Section II gives an overview of the literature for the treated subjects. Section III describes the context and assumptions for which the system has been designed. Section IV explains the design of the proposed solution. Section V reports how the performance of the framework was evaluated, and the obtained results. Finally, Section VI presents concluding observations.

II. Related work

The problem of placing the VNFs of a Service Function Chain (SFC) while determining the optimal path for their communication has been studied extensively. CHIMA leverages on that literature and, instead, focuses on the deployment and monitoring that follows the deployment decision. Addis *et al.* [4] consider the joint problem of placement and chaining of the VNFs and propose a mixed integer linear programming model for its optimization. Khoshkholghi *et al.* [5] also focus on the

This work is partially funded by EU Grant no. 101016577 project AI-SPRINT.

optimization of latency and costs in the placing of SFCs, but do so with heuristic-based algorithms. A similar goal to the one of this work is pursued by Mechtri *et al.* [6], who propose an SFC orchestration framework that takes the monitoring of the deployed service into consideration, but do not propose solutions to guarantee their performance.

All of the above only take into consideration SFCs composed by VNFs that target homogeneous compute architectures. CHIMA supports the deployment of heterogeneous SFCs that take advantage of programmable data planes and significantly increase the achievable throughput. The concept of decomposed VNFs that take advantage of programmable network hardware is explored by Moro *et al.* [2], who propose a model and the heuristics for the optimization of the placement of functions that provide implementations for multiple types of hardware. In a similar fashion, DPPx [7] also enables the installation of data plane programs on P4 switches for the enhancements of NFVs. Additionally, Moro *et al.* [2] propose the use of an extensible template pipeline and segment routing for the deployment of user functions on programmable data planes. The template pipeline of CHIMA is inspired from the one of [2].

Other approaches for the composition of P4 functions have been studied in Hyper4 [8], which defines a P4 program that is able to emulate other P4 programs provided at runtime by the control plane and in P4Visor [9], which suggests a technique to merge multiple P4 programs into a single one, while retaining the functionality of all of them. An alternative take on the acceleration of SFCs with programmable data planes is explored with P4sc [10] and in [11], both of which consider the implementation of whole function chains on single switches, lacking the possibility of combining components of different technologies.

This paper also proposes the relocation of the components of the chains at runtime as a solution to tackle performance degradation. The optimization for this kind of readjustment has been explored in [12], proposing an algorithm for the real-time migration of VNFs and demonstrating the possibility of lowering the latency with it. The performance of similar reconfiguration scenarios in microservices orchestrators, which however happen in case of failures instead of degradation, has been studied by Vayghan *et al.* [13]. CHIMA uses In-band Network Telemetry (INT) to monitor the deployed SFCs and understand when to execute the redeployment of the network functions. A similar approach is studied with IntOpt [14], which aims at optimizing the overhead introduced while achieving optimal measurements of a deployment, but lacks the measurement of the execution time of functions, preventing the calculation of the end-to-end metrics seen by packets. The same is done by IntSight [15], which has a similar goal to CHIMA in trying to detect requirement compliance. Instead, Choi *et al.* [16] propose the use of INT to heal the performance of services at runtime, but only considering the alteration of traffic flows between fixed endpoints. The relocation of P4 VNFs at runtime is studied in depth by P4NFV [17], which also enables the migration of stateful functions while preserving their consistency.

III. System Model

The proposed framework operates on a system composed of both programmable switches that can be targeted by a P4 compiler and compute nodes capable of running containers. The ONOS SDN controller [18] is used to manage the forwarding of the packets; the computing nodes are configured using Docker. The system can be used for the deployment of network services in the form of heterogeneous Service Function Chains, composed of two types of functions:

- **General purpose functions:** designed to run on the compute nodes. These are available to the framework as container images.
- **P4 functions:** intended to be compiled and run on P4 programmable switches. These are available as P4 programs in which a P4 `control` block with a compatible signature is defined.

Each service chain is associated to an SLA that specifies the maximum end-to-end delay and jitter. The CHIMA framework is responsible for measuring the end-to-end delay and jitter, also across the nework functions, by means of In-Network Telemetry and for comparing the collected measurements to the SLA of each chain. In case a chain violates the SLA, the chain is rerouted through a new path which satisfies the SLA. The calculation of the SLA-constrained path is managed by the orchestrator and is outside of the scope of this paper.

In the current implementation of CHIMA, each function of this chain supports one successor at most, but could be easily extended to cover more complex service graphs. Figure 1 shows an abstract representation of an SFC and how it can be deployed on the available devices.

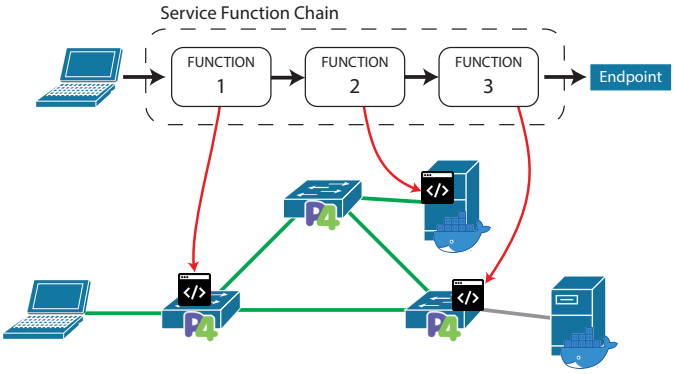


Fig. 1: Logical view of a Service Function Chain and how it can be mapped to a physical topology

IV. The CHIMA framework

CHIMA is a framework for *CHain Installation, Monitoring and Adjustment*. Its goal is to ease the placement and installation of applications that take advantage of programmable data planes, managing the orchestration of heterogeneous SFCs in the network and guaranteeing their performance through

relocation of functions at runtime when necessary. This section will give an overview of its implementation.

A. Framework components

The CHIMA framework consists of multiple components, distributed over a supported network as shown in Figure 2.

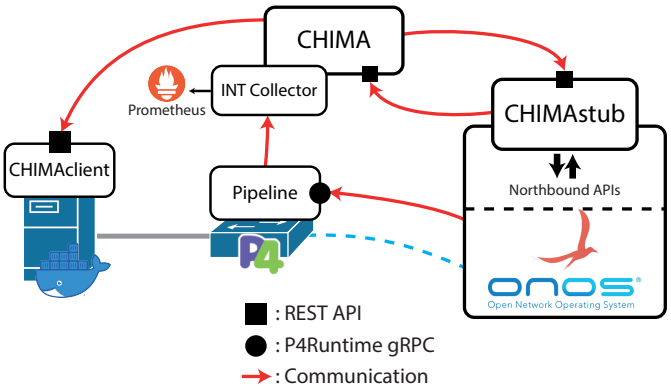


Fig. 2: Logical placement of the framework components in the network and their interactions

1) CHIMAstub: This component is an ONOS application. It exposes topology information such as the available switches, hosts and links, and events like the activation and deactivation of devices to CHIMA. Moreover, it allows the interaction with the network's devices through an extension of the ONOS REST APIs. These enable the installation of new pipelines on programmable switches, and the configuration of INT monitoring. ONOS also manages the forwarding of traffic that is not handled by CHIMA, by capturing ARP, LLDP and DHCP packets and installing according rules on switches.

2) CHIMAclient: This process runs on hosts, and enables the routing scheme explained in Section IV-C by applying the MPLS header stack to the packets of managed services.

3) P4 pipeline: This is installed on all the switches. On top of providing basic forwarding, it supports In-band Network Telemetry according to the INT v1.0 specification [1]. This pipeline is used as a template for the inclusion of user provided P4 functions at runtime.

4) INT Collector: This component is co-located with the core CHIMA process. It receives INT reports generated by the switches running the P4 pipeline, extracts the telemetry data and maintains a moving average of the measurements, as explained in Section IV-D. These values are periodically provided to CHIMA for the comparison with requirements, and sent to a server of the Prometheus monitoring system that stores them and makes them easily accessible to users.

5) CHIMA: The CHIMA module is the central manager of the system. Its tasks are to:

- Build and maintain an internal representation of the network topology.
- Compute a deployment strategy based on the available topology information.
- Inject user provided P4 functions into the template pipeline.
- Carry out the installation of P4 pipelines through CHIMA Stub.
- Deploy Docker containers on hosts using Docker Compose.
- Compute performance metrics for communication paths, using data provided by the INT Collector, and compare them with user set requirements.
- Alter the placement of functions and reroute traffic accordingly in case of a requirement violation.

B. Template Pipeline

The P4 switches are programmed with a pipeline containing an extensible section. The service functions are deployed into this section. A dispatcher makes sure that the various functions are applied only to the packets belonging to a specific service chain.

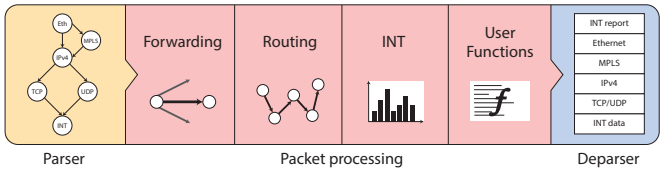


Fig. 3: Stages of the template pipeline

The processing sections of the pipeline, depicted in Figure 3, are the following:

1) Forwarding: The pipeline is based on the `basic.p4`[1] pipeline included in ONOS and inherits its forwarding mechanism. The treatment of packets is determined by the reactive forwarding rules installed by the controller.

2) Routing: The routing of packets between functions of a service managed by CHIMA is handled separately, and bypasses the usual forwarding, as explained in Section IV-C.

3) INT: The implementation of In-band Network Telemetry is inherited from the ONOS `int.p4` pipeline. The control plane logic for this part of the pipeline is the `inbandtelemetry` ONOS application, which translates INT intents into the rules to be installed on the switch.

4) User functions: The template pipeline includes a section in which user code can be inserted. Users provide functions to execute on packets in the form of `control` blocks with a predefined signature, such as the one presented in Listing 1. Control blocks allow the definition of arbitrary computation on the packet by defining tables, instantiating other controls, using registers, etc. The access to parsed headers and metadata is provided through the parameters of the control, enabling the user's function to change the content of the packet and even altering the predetermined egress port.

[1] https://github.com/opennetworkinglab/onos/tree/master/pipelines/basic

```
control decrement_ttl(
    inout headers_t hdr,
    inout local_metadata_t local_metadata,
    inout standard_metadata_t std_metadata)
{
    apply {
        if(hdr.ipv4.isValid()) {
            hdr.ipv4.ttl = hdr.ipv4.ttl - 1;
        }
    }
}
```

Listing 1: Example of a simple user defined control that can be submitted to CHIMA as a function

When CHIMA determines one or more P4 functions of a service have to be deployed on a switch, their code is injected, and will only be executed for packets of the correct service.

5) Setting up the pipeline: The decision of what user functions must be deployed in each switch is made by an orchestrator. The orchestrator is outside of the scope of this paper, but we note that the orchestrator described in [2] is compatible with the CHIMA framework.

Once the orchestrator provides the CHIMA module with the decisions, CHIMA compiles the resulting P4 program using the p4c compiler. The installation of the pipeline at runtime can be achieved with the `SetForwarding-PipelineConfig` RPC call of P4Runtime. Its implementation in ONOS is exploited by creating a Pipeconf with the pipeline files that is then bound to the device in ONOS's distributed map. In addition to this, the installation process involves the reconciliation of rules installed in the device's tables. In general, reconfiguration of the pipeline may cause significant downtime, but platform specific features like Tofino Fast Refresh [19], can greatly accelerate this process.

C. Routing

After all the functions have been installed, the correct routing of packets between them has to be configured along the prescribed path.

1) Segment Routing over MPLS: CHIMA routing is based on Segment Routing over MPLS (SR-MPLS), which uses values called Segment IDs (SIDs) to instruct switches on the operations to execute on a packet by embedding them in the packet itself. In particular, we implement the approach defined by RFC8660 [20], in which SIDs are represented as MPLS labels. We define three SR Local Blocks (SRLB), whose semantic for specific segments is dependent on the node that executes it.

- `0x40000 - 0x7FFFF`: used for the execution of user defined P4 functions that have been deployed on the switch.
- `0x80000 - 0xBFFFF`: used for packet forwarding. These segments can be classified as Adjacency SIDs according to RFC8402.
- `0xC0000 - 0xFFFFF`: used to implement the custom extension of INT for the measurement of the delay introduced by general purpose functions, explained in Section IV-D2. While `0xC0000` is used to instruct switches to forward INT data to hosts, the remaining values of the block are used to identify the executed function.

Keeping in mind that MPLS labels are 20 bits long, their 2 most significant bits are used to identify their SRLB, while the remaining 18 can be interpreted as an argument for the action to be performed. The evaluation of segments on a switch continues until the bottom of the stack is reached or an Adjacency SID is found, for which a Penultimate Hop Popping approach is used. Figure 4 shows an example of how MPLS label stacks are used by the framework to perform routing.

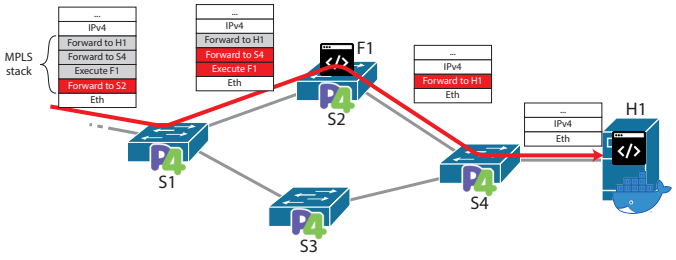


Fig. 4: Example of the framework's use of segment routing

2) Encapsulation and segment distribution: In CHIMA, the MPLS encapsulation of packets belonging to managed services is performed by CHIMAclient. At the heart of CHIMAclient is an eBPF filter that inspects the packets egressing the host. Services are identified using the tuple of source and destination IPv4 addresses. If the tuple is known, it means that the filter has a label stack that represent the series of segments used to implement its pre-computed path. In this case, the stack of MPLS label headers is inserted between the Ethernet and IPv4 headers. During this process, the EtherType field of the Ethernet header is set to `0x8847` to allow proper parsing.

These stacks are computed by the CHIMA process based on the result of an optimization model, and then installed on the CHIMAclient of specific hosts by contacting their REST APIs.

D. In-band Network Telemetry

The INT implementation used by the framework is derived from the `int.p4` pipeline included in ONOS, which is designed to be managed by the `inbandtelemetry` application, with which CHIMA interfaces through CHIMAstub. This implementation is based on the INT v1.0 specification, and uses embedded metadata with headers located over TCP or UDP.

1) Collection of INT data: The CHIMA process includes an INT collector as one of its modules, which has been adapted form an existing eBPF based implementation [21]. The source of the obtained values is identified by the pair of IDs of the switches at the two ends of a link. For each incoming INT packet, the collector computes the packet delay d_p. The collector also maintains a running Exponentially Weighted Moving Average (EWMA) of the delay d, which is updated for each incoming packet as follows:

$$d \leftarrow (1-\alpha)\, d + \alpha d_p \tag{1}$$

The average value of jitter is also measured following the same procedure. Periodically, at each polling interval, the running values of delay and jitter are also conveyed to a server of the Prometheus monitoring system.

The α parameter acts as a smoothing factor, and can be modified by the user when running CHIMA to tune the response to transient variations of the metrics.

2) Measurement of general purpose function times: In order to measure the time it takes for a packet to reach a particular function, it is necessary to consider the time to pass through all the previous functions in the chain. While P4 functions run in constant time over specialised hardware, general purpose compute nodes cannot assure the same level of stability.

In this paper we focus on the measurement of functions that process UDP packets, with each packet containing a single application message. This makes it possible to correlate the time needed by the packet to traverse the function's host and the time of execution of the function itself.

A general purpose function deployed on a compute host will process packets forwarded by the switch to which it is directly connected. When its execution ends, the resulting packet will be sent back on the same switch to continue its routing across the remaining functions. The measurement is performed by altering the forwarding behavior of packets with INT headers by using MPLS Segment ID `0xC0000`. INT data will be left in the packet, and new INT transit headers will be added before the packet's egress. The egress timestamp included in these headers will be considered the start of the function's execution. This requires the function to be aware of this data and leave it untouched. The resulting packet will then include previous INT data and the function output as a payload. Segment IDs in the range `0xC0001-0xFFFFF`, added to the egress packet by the CHIMAclient, uniquely identify the executed function, as shown in Figure 5. This information is then used by the switch to mark the end of the function's execution with the value of the ingress timestamp, enabling the computation of its extent.

V. Numerical Results

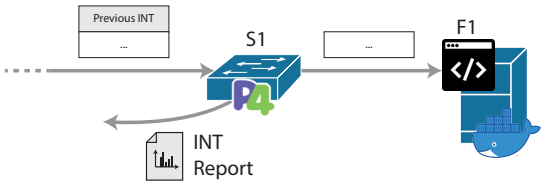
A. Methodology

We evaluated the CHIMA framework over multiple topologies and services, as shown in Figure 6, whose relevant characteristics are shown in Table I. Their complete definition can be found in the project repository [3].

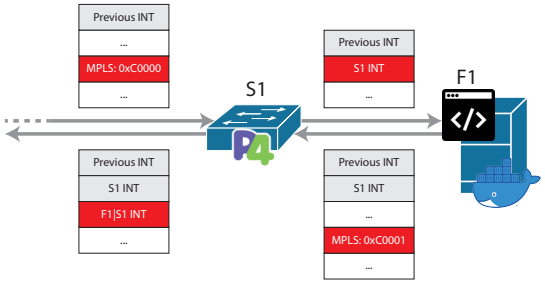
TABLE I: Characteristics of the presented test cases.

Topology	Switches	Containers	P4 func.
mesh	7	3 (1)	0 (0)
datacenter	6	2 (0)	1 (1)
unbalanced	4	4 (1)	2 (1)
minimal	5	2 (2)	2 (2)
medium	7	3 (3)	3 (3)
large	9	4 (4)	4 (4)

The number of functions moved upon network failure is in parentheses.

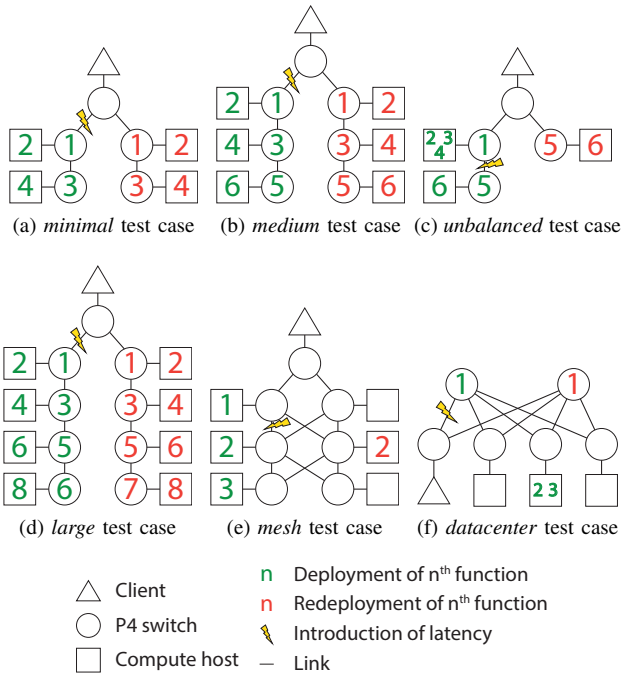


(a) Regular processing of an INT packet when forwarded to a host



(b) Additional measurement of the function's time using Segment IDs

Fig. 5: Comparison of the content of packets and the forwarding behavior with regular INT and with CHIMA's extension



(a) *minimal* test case (b) *medium* test case (c) *unbalanced* test case

(d) *large* test case (e) *mesh* test case (f) *datacenter* test case

Fig. 6: Topologies used to evaluate the redeployment times

The test cases have been simulated with the FOP4 [22] platform, using bmv2 instances as switches. To evaluate the detection and redeployment performance, the framework has been instrumented to record the timestamps of relevant events. Figure 7 shows the complete list of values obtained during a test run. All measurements have been performed on a bare-metal installation of Ubuntu 20.04 LTS, running on an Intel

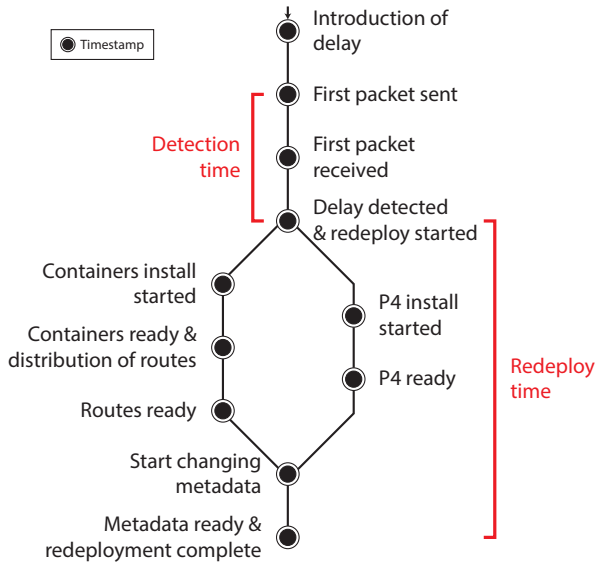


Fig. 7: Representation of the timestamps collected during one test run, their relationship in time, and how performance measurements are computed on them.

Core i7-6700 CPU with 64GB of RAM.

Each experiment starts with the creation of the simulated topology on FOP4, followed by the deployment of the related service with CHIMA. After the service is properly set up, the latency of the targeted link is artificially increased to a level that causes requirements to be exceeded. The event is detected when one of the path-wise measurements reaches the value of the requirement, and causes the redeployment process to begin.

B. Detection delay

The first set of measurements have the objective of determining how much time is needed by the framework to detect the introduction of a perturbation, depending on the values of user-configurable parameters.

The detection delay is computed as the time CHIMA takes to detect an exceeded requirement after the first packet of a perturbed application is sent. Assuming the properties of topology and service to be constant, we can consider the detection delay to be a function of the polling interval and the EWMA coefficient. For this reason, all the measurements of this section have been performed on the *minimal* topology, shown in Figure 6a.

a) Polling interval: Figure 8 shows the average detection delay versus the rate at which the userspace component of the eBPF INT collector polls new EWMA values. Each experiment was repeated 30 times and we report the average result. These measurements have been performed with $\alpha = 2^{-3}$ in (1). For comparison, Figure 8 also shows the time taken by a request to traverse the function chain end-to-end with a red line.

As expected, the results present a clear linear trend, directly proportional to p. A constant contribution is given by the time

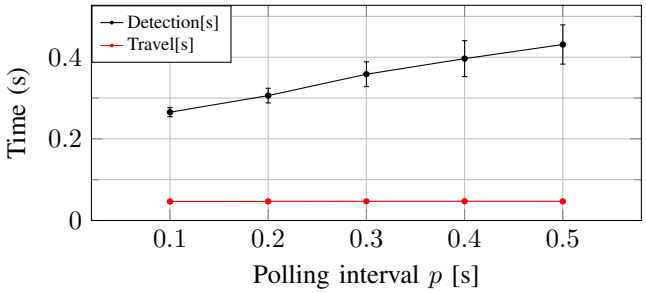


Fig. 8: Delay in the detection of an exceeded requirement with different intervals for the polling of new measurements from the INT collector. The bars indicate the 95% confidence interval.

needed for the EWMA-smoothed measurements to cross the threshold, while the slope of the curve is due to the polling frequency. As expected, the mean delay due to the polling latency is about $\frac{p}{2}$.

b) EWMA coefficient: The second parameter is the smoothing coefficient for the computation of the EWMA in equation (1). While running these tests, the Polling interval was set to 0.1 s.

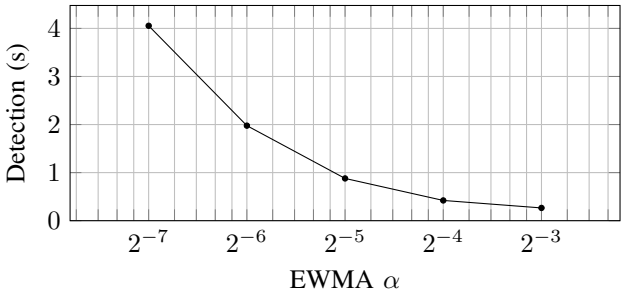


Fig. 9: Delay in the detection of an exceeded requirement with different coefficients for the computation of the EWMA on link measurements. The bars indicate 95% confidence intervals.

Larger values of α result in more weight given to recent data, as shown by Figure 9, in which smaller coefficients cause a larger time needed for convergence to the new measured values. This data shows that different values of α can be used to tune the response of the framework in case of short-lived congestion events. A different value of α can be set for each chain. In case of long or complex SFCs, for which the redeployment process can be expensive in terms of time and resources, a smaller α yields smoother measurements and avoid route flapping and rearrangements that could decrease the chain's performance as much as the degradation itself. In other scenarios, where maintaining performance requirements at all times is critical, α can be set to a bigger value. This will result in noisier measurements that more closely follows the raw values obtained from INT, allowing the framework to minimize delays in the detection of a problem. For these reasons, the optimal value for this parameter should be deter-

TABLE II: Breakout of redeployment times for different topologies. 95% CI.

Topology	P4 (s)	Containers (s)	Paths (ms)	Metadata (ms)
mesh	-	1.13 ±0.04	54.23 ±5.07	2.68 ±0.13
datacenter	5.20 ±0.03	-	288.60 ±5.30	2.79 ±0.16
unbalanced	5.19 ±0.03	0.97 ±0.01	60.57 ±2.48	2.81 ±0.17
minimal	6.06 ±0.04	1.50 ±0.02	48.79 ±5.01	3.50 ±0.19
medium	6.93 ±0.03	2.12 ±0.03	125.89 ±17.95	4.81 ±0.97
large	8.28 ±0.09	2.54 ±0.08	477.55 ±90.06	6.10 ±1.26

mined for each chain based on the deployed service and the intended application.

C. Redeployment time

Another crucial measurement to outline the framework performance is the time needed to complete a redeployment. Table I presents a comparison of the relevant characteristics among the test cases for which redeployment times have been measured.

The results presented in Figure 11 show the total redeployment times for these cases, along with the most significant contributing factors. Additionally, all recorded contributes are detailed in Table II. The time needed by CHIMA to complete the service's reconfiguration is consistent with the performance of other orchestrators in similar cases, according to the analysis of Vayghan *et al.* [13].

Since the redeployment of the different components is executed in parallel, as shown in Figure 10, the recorded times will be equal to the delay caused by the slowest one. The installation of P4 functions proves to be the dominant factor if present, causing the total time to be in the order of seconds. The two contributions to this delay are the reconfiguration of the switch's pipeline and the reinstallation of the flow rules in the pipeline's tables. The former is due to the use of bmv2 switches and could be reduced to tens of milliseconds with vendor specific features. The latter, instead, is caused by ONOS management of programmable data planes, which is not structured for time sensitive pipeline changes. Improvements to target this use case could drastically decrease delays. Moreover, the introduction of features to enable a partial reconfiguration of the pipeline would further decrease these figures. In such a scenario, there would be no need for the controller to install the same set of rules after each reconfiguration, since they are independent of the set of user functions deployed on the switch. The times for path distribution and metadata adjustment are entirely attributed to the CHIMA framework logic, but are less significant than the previous ones, adding minimal overhead.

VI. Conclusion

In this paper, we propose the CHIMA framework for the deployment, monitoring and real-time readjustment of heterogeneous SFCs. CHIMA allows the specification of performance requirements for functions and services and monitors it by means of the telemetry powered by programmable data planes. This opens the opportunity for applications with

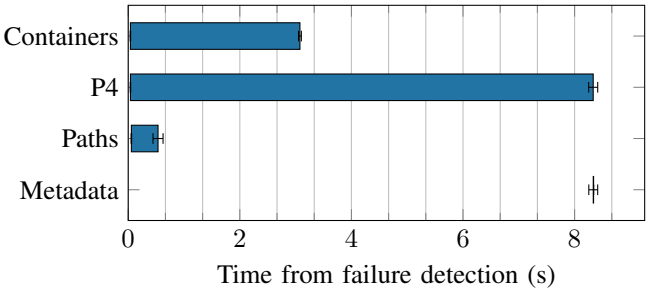


Fig. 10: Temporal relationship between the operations that contribute to redeployment times, measured on the *large* test case (Figure 6d). 95% CI.

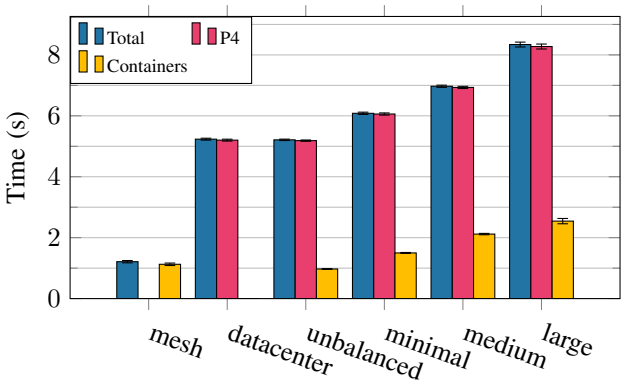


Fig. 11: Time for the complete redeployment of a service, along with the contributes of P4 and Container redeployment, in different test cases. 95% CI.

critical performance demands to use existing networks for their communication. Measurements include the execution time of functions on compute nodes, allowing the constraints to reflect real delays experienced by packets, and not just the ones caused by the network. We developed a prototype and tested it through emulation on the FOP4 platform with bmv2 switches. The results showed that the detection of exceeded requirements happens in the order of hundreds of milliseconds, and can be tuned by the user to achieve the desired level of responsiveness. Analysis of the redeployment process showed that the overhead introduced by the system is negligible compared to the time needed for the startup of functions, and real time relocation of VNFs to achieve desired levels of performance is feasible.

References

[1] T. P. A. W. Group *et al.*, "In-band Network Telemetry (INT) Dataplane Specification - Version 1.0," 2018. [Online]. Available: https://github.com/p4lang/p4-applications/raw/master/docs/INT_v1_0.pdf

[2] D. Moro, G. Verticale, and A. Capone, "Network function decomposition and offloading on heterogeneous networks with programmable data planes," *IEEE Open Journal of the Communications Society*, vol. 2, pp. 1874–1885, 2021.

[3] "CHIMA: CHain Installation, Monitoring and Adjustment." [Online]. Available: https://github.com/ANTLab-polimi/CHIMA

[4] B. Addis, D. Belabed, M. Bouet, and S. Secci, "Virtual network functions placement and routing optimization," in *2015 IEEE 4th International Conference on Cloud Networking (CloudNet)*. IEEE, 2015, pp. 171–177.

[5] M. A. Khoshkholghi, M. G. Khan, K. A. Noghani, J. Taheri, D. Bhamare, A. Kassler, Z. Xiang, S. Deng, and X. Yang, "Service function chain placement for joint cost and latency optimization," *Mobile Networks and Applications*, vol. 25, no. 6, pp. 2191–2205, 2020.

[6] M. Mechtri, C. Ghribi, O. Soualah, and D. Zeghlache, "NFV orchestration framework addressing SFC challenges," *IEEE Communications Magazine*, vol. 55, no. 6, pp. 16–23, 2017.

[7] T. Osiński, H. Tarasiuk, L. Rajewski, and E. Kowalczyk, "DPPx: A P4-based Data Plane Programmability and Exposure framework to enhance NFV services," in *2019 IEEE Conference on Network Softwarization (NetSoft)*. IEEE, 2019, pp. 296–300.

[8] D. Hancock and J. Van der Merwe, "Hyper4: Using P4 to virtualize the programmable data plane," in *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*, 2016, pp. 35–49.

[9] P. Zheng, T. Benson, and C. Hu, "P4visor: Lightweight virtualization and composition primitives for building and testing modular programs," in *Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies*, 2018, pp. 98–111.

[10] D. Zhang, X. Chen, Q. Huang, X. Hong, C. Wu, H. Zhou, Y. Yang, H. Liu, and Y. Chen, "P4SC: A High Performance and Flexible Framework for Service Function Chain," *IEEE Access*, vol. 7, pp. 160 982–160 997, 2019.

[11] J. Lee, H. Ko, H. Lee, and S. Pack, "Flow-aware service function embedding algorithm in programmable data plane," *IEEE Access*, vol. 9, pp. 6113–6121, 2020.

[12] D. Cho, J. Taheri, A. Y. Zomaya, and P. Bouvry, "Real-time virtual network function (VNF) migration toward low network latency in cloud environments," in *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*. IEEE, 2017, pp. 798–801.

[13] L. A. Vayghan, M. A. Saied, M. Toeroe, and F. Khendek, "Microservice based architecture: Towards high-availability for stateful applications with kubernetes," in *2019 IEEE 19th International Conference on Software Quality, Reliability and Security (QRS)*. IEEE, 2019, pp. 176–185.

[14] D. Bhamare, A. Kassler, J. Vestin, M. A. Khoshkholghi, and J. Taheri, "IntOpt: In-band network telemetry optimization for nfv service chain monitoring," in *ICC 2019-2019 IEEE International Conference on Communications (ICC)*. IEEE, 2019, pp. 1–7.

[15] J. Marques, K. Levchenko, and L. Gaspary, "Intsight: diagnosing slo violations with in-band network telemetry," in *Proceedings of the 16th International Conference on emerging Networking EXperiments and Technologies*, 2020, pp. 421–434.

[16] N. Choi, L. Jagadeesan, Y. Jin, N. N. Mohanasamy, M. R. Rahman, K. Sabnani, and M. Thottan, "Run-time performance monitoring, verification, and healing of end-to-end services," in *2019 IEEE Conference on Network Softwarization (NetSoft)*. IEEE, 2019, pp. 30–35.

[17] M. He, A. Basta, A. Blenk, N. Deric, and W. Kellerer, "P4NFV: An NFV architecture with flexible data plane reconfiguration," in *2018 14th International Conference on Network and Service Management (CNSM)*. IEEE, 2018, pp. 90–98.

[18] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow *et al.*, "ONOS: towards an open, distributed sdn os," in *Proceedings of the third workshop on Hot topics in software defined networking*, 2014, pp. 1–6.

[19] A. Bas, "Leveraging stratum and tofino fast refresh for software upgrades," *Accessed: Jul*, vol. 4, p. 2021, 2018.

[20] A. Bashandy, C. Filsfils, S. Previdi, B. Decraene, S. Litkowski, and R. Shakir, "Segment Routing with the MPLS Data Plane," RFC 8660, Dec. 2019. [Online]. Available: https://rfc-editor.org/rfc/rfc8660.html

[21] "INT Collector - Atmosphere project." [Online]. Available: https://github.com/eubr-atmosphere/distributed-network-federation-probe/tree/master/int_collector

[22] D. Moro, M. Peuster, H. Karl, and A. Capone, "FOP4: Function offloading prototyping in heterogeneous and programmable network scenarios," in *2019 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. IEEE, 2019, pp. 1–6.