Closed-loop Orchestration for Cloud-native Mobile IPv6

Ákos Leiter[1], Edina Lami[1,2], Attila Hegyi[1], József Varga[1], and László Bokor[2,3]

***Abstract*—With the advent of Network Function Virtualization (NFV) and Software-Defined Networking (SDN), every network service type faces significant challenges induced by novel requirements. Mobile IPv6, the well-known IETF standard for network-level mobility management, is not an exemption. Cloud-native Mobile IPv6 has acquired several new capabilities due to the technological advancements of NFV/SDN evolution. This paper presents how automatic failover and scaling can be envisioned in the context of cloud-native Mobile IPv6 with closed-loop orchestration on the top of the Open Network Automation Platform. Numerical results are also presented to indicate the usefulness of the new operational features (failover, scaling) driven by the cloud-native approach and highlight the advantages of network automation in virtualized and softwarized environments.**

***Index Terms*—IP mobility, CN-MIPv6, ONAP, failover, scaling**

I. INTRODUCTION

Network Function Virtualization (NFV) and Software Defined Networking (SDN) have not left any Network Function (NF) untouched. Meanwhile, cloud systems, either virtual machine or container-based, have created new execution environments. Nowadays, cloud-native service provisioning can bring in failover and scaling scenarios more straightforwardly than ever before. This implicitly indicates the usage of orchestration, which helps to organize the right amount of resources to the right place in time. In this paper, we use Open Network Automation Platform (ONAP) [1] for the practical experiments. With ONAP, we can run automatic failover and scaling scenarios when specific circumstances are met. In our experiments, we entirely rely on the closed-loop orchestration platform of ONAP. This is where Mobile IPv6 (MIPv6) [2] comes into the picture, whose functionality and operational procedures can be extended using the cloud. MIPv6 is part of a broader protocol family called IP-level mobility management. The current technological trends of cloudification spanned over the NFV, and SDN paradigms, together with orchestration requirements, do not leave MIPv6-based mobility support untouched. Cloud-native Mobile IPv6 (CN-MIPv6) [3] [4]was proposed to meet the expectations and apply the benefits of the trends mentioned above.

This paper aims to show how failover and scaling can be applied to CN-MIPv6 in the context of closed-loop orchestration. Here we define failover as the time of restoring redundancy. Furthermore, the article presents numerical results and calculations on the utilization of failover, scaling, and availability in the case of cloud-native IP-level mobility management. At the end of the paper, an analysis shows the benefits of network automation from the reliability and redundancy point of view.

The remaining sections are organized as follows. Section II presents related works. The connection between ONAP and CN-MIPv6 is elaborated in Section III, followed by the measurements and numerical calculations in Section IV. Conclusions and possible future research directions are in Section V and Section VI, respectively.

II. RELATED WORKS

A. Literature

Cloudification of 5G network functions has been a trending paradigm. *Du et al.* [5] consider the cloud-native bases of 5G Access and Mobility Management Function (AMF).

Another member of IP-based mobility management, such as Proxy Mobile IPv6 [6], has also been shaped to be cloud-native: cloud-native Proxy Mobile IPv6 (CN-PMIPv6) [7]. Flow Mobility, a concept standardized for Proxy Mobile IPv6 (PMIPv6) and discussed in Section III for MIPv6, makes it possible to separate IP flows by 5-tuples and assign individual mobility policies for each flow.

In our work, the Single-Point-of-Failure (SPOF) problem is solved by immediate and automatic redeployment of the mobility anchor (*Home Agent*). Furthermore, the number of anchors can be increased dynamically with automatic scaling. Obviously, there are other strategies for mitigating the SPOF problem in IPv6-based mobility management. With the help of SDN and Openflow, IPv6-based mobility management can be implemented in many different ways [8] [9] [10] [11] [12].

Dimitris Giatsios et al. [13] examine the failover of Network Slices. The paper written by *Veronica Quintuna Rodriguez et al.* examines ONAP-based deployment and management of Network Slices [14] [15]. ONAP has also been used to enhance access discovery and selection functions in the 5G core network in an article by *Rahul Banerji et al.* [16].

Predictive failover of Virtual Network Functions (VNF) in the context of edge computing is presented by *Huawei Huang et al.* [17]. A unique programming language is shown to ensure fault tolerance in SDN by *Reitblatt et al.* [18]. Scaling and failover can also be executed with Kubernetes on the Pod level by Kubernetes Horizontal Pod Autoscaler (HPA) [19] and Kubernetes Deployment [20]. But these are not suitable for inter-Kubernetes cluster failover and scaling: ONAP can manage many Kubernetes clusters simultaneously. As our concept opens the way to machine learning applications of

[1] Nokia Bell Labs, Budapest, Hungary, (E-mail: {akos.leiter, attila.hegyi, jozsef.varga}@nokia-bell-labs.com; edina.lami@nokia.com)

[2] Department of Networked Systems and Services, Faculty of Electrical Engineering and Informatics, Budapest University of Technology and Economics, Budapest, Hungary, (E-mail: bokorl@hit.bme.hu)

[3] ELKH-BME Cloud Applications Research Group, BME Informatics Building, Budapest, Hungary

IPv6-based mobility management, it is worth mentioning that mobility management itself is also examined in the field of Machine Learning optimization [21] [22] [23].

There is a good collection of mobility management issues in 5G networks with low-latency services by *Johanna Heinonen et al.* [24]. The identified challenges include topology-aware gateway selection, handover management, and gateway relocation. Our proposal inherits features of automatic gateway relocation from the system architecture level: appropriate functions are triggered automatically in case of failures.

Network automation and mobility management are not only in the interest of scientific papers. There are several other forums and publications, like blogs, which are worth mentioning (e.g., [25] [26] [27]). To the best of our knowledge, no publicly available literature deals with integrating closed-loop orchestration, cloud-native IPv6-based mobility management, and ONAP. The whole operation model of CN-MIPv6 – presented in this paper – is a novel concept. Of course, the scope of our article is limited to failover and scaling scenarios of CN-MIPv6, but these are the essential use cases demonstrating the power of our scheme. The microservice architecture CN-MIPv6 used in this paper for evaluation can be found in [4].

B. *Orchestration and ONAP in a nutshell*

Orchestration (in telecommunication) is about ensuring the right amount of resources at the right time and location for a particular service. In practice, this means enough pieces of virtual machines and containers must be placed behind a specific service.

Figure 1 depicts another function of orchestration which operates with control loops. The system watches the particular service's actual states continuously and consistently enforces the desired state. This is how closed-loop orchestration or Kubernetes Operators [28] work on a very high level.

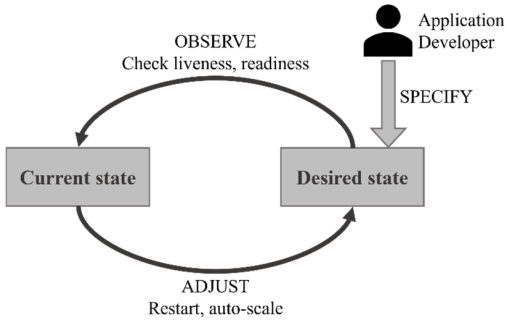

Figure 1 – Orchestration in general [29]

ONAP is an open-source network automation and closed-loop orchestration platform. A simplified view of its architecture can be seen in Figure 2. Its main component is the Service Orchestrator (SO) [30], which is responsible for executing abstract steps (Building Blocks, BB) to instantiate a particular Network Service (NS) instance. Network Service Models are designed in the Service Design and Creation (SDC) component [31]. These two components indicate that ONAP strictly separates the design time (Day 0) and the deployment time (Day 1). Particular workflows and controllers can be written to any NS with the help of the Controller Design Studio (CDS) [32]. CDS can also store Day 2 configuration changes or any other workflows intended to the part of a closed-loop control function. Software components of a Network Service Model, including Custom Controllers (Controller Blueprint Archive, CBA), are encapsulated as Vendor Software Package (VSP). In the case of CNF, a Helm chart is included in the VSP.

The Policy Framework [33] is the heart of the closed-loop functions of ONAP. It stores what to do when a particular event happens with an NF. The Policy Framework uses Policy Models, which hide the actual API calls to SO and CDS. Every Policy Model is assigned to a Network Service Model. The Onset event is sent out by a custom-made Analytic Application (AA) to trigger a Policy Model. Deciding when to send out an Onset message is entirely up to the developer of AA. The primary source of measurements for AA is VES messages sent out by a running NF. Every message, including Onset, *VNF Event Streaming (VES),* is transferred via DMAAP [34], which is the central message bus of ONAP.

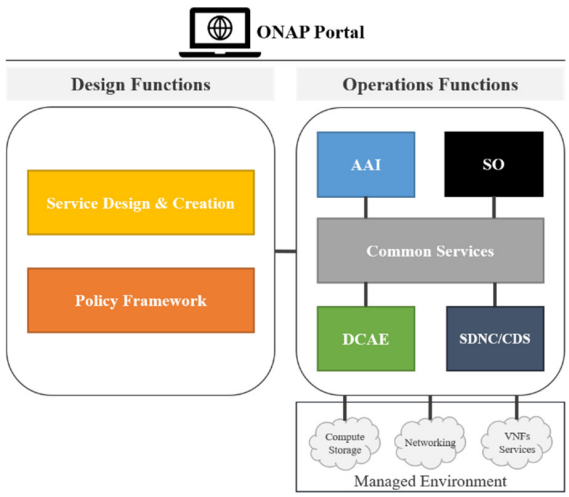

Figure 2 – The ONAP architecture [35]

III. ONAP CONSIDERATIONS FROM THE CN-MIPV6 SERVICE PROVISIONING POINT OF VIEW

Our proposed approach of closed-loop orchestration in the context of CN-MIPv6 is depicted in Figure 3. On one hand, a Kubernetes layer is responsible for executing readiness and liveness probes for basic health checks. On the other hand, there is a broader scope of the control loop based on ONAP. This is where complex logic of automatic workflows is placed, such as analytics and machine learning-based operations. Of course, multiple control loops can be deployed here, such as autoscaling, auto-healing, etc. Furthermore, the scope of these control loops is not only restricted to one Kubernetes cluster. This is where inter-cloud control logic can be deployed.

In the context of ONAP and CN-MIPv6 integration, we use a multi-site VNF during the failover and scaling particularly. When a new instance of *Home Agent Packet Procesor* (*HA-PP)* is created, a new Virtual Function (VF) module is added to the existing VNF but with a different cloud region. This VF module hosts the corresponding Helm charts of *HA-PP*, which are added as a VSP to that particular Network Service Model.

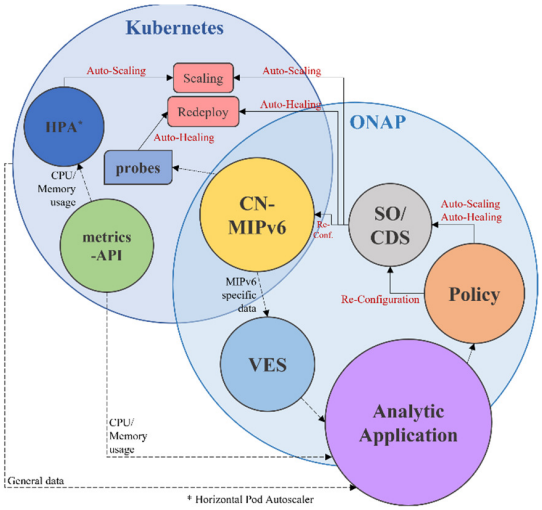

Figure 3 – Orchestration levels of CN-MIPv6

IV. MEASUREMENTS AND NUMERICAL ANALYSIS

A. *Testbed design*

The testbed runs on a Nokia Airframe server with Ubuntu 18.04 LTS. On the top of one VM, Kubernetes 1.20.2 is installed, which hosts the *HA-PPs*. From a simplicity point of view, we executed a namespace-to-namespace failover and scaling in our experiments. Of course, such use cases can also be applicable in a multi-cloud environment. Figure 4 shows the low-level design of the testbed from networking point-of-view. *Mobile Node (MN)* connects to a router (R) in order to add additional network between *MN* and *HA* (not being directly connected via home link). *Corresponding Node (CN)* represent a node which is not inside the mobility management domain.

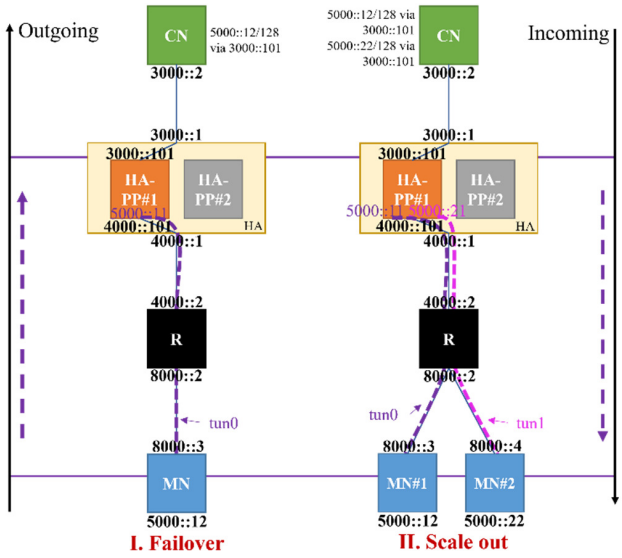

Figure 4 – Low-level design of the testbed

We added readiness and liveness probes to our *HA-PP* Pod. These probes monitor whether the corresponding RAW socket for sending and receiving is open. The following values are used:

- Initial delay of Readiness Probe: 20 sec
- PeriodSeconds of Readiness Probe: 90 sec
- Initial delay of Liveness Probe: 15 sec
- PeriodSeconds of Liveness Probe: 60 sec

B. *Utilization for Failover*

Failover is the process when the traffic to a malfunctioning network function is offloaded to a working one. Our measurements logged how long it takes for an automatic failover managed by ONAP toolsets. In the context of CN-MIPv6, this means adding a new *HA-PP* to a different namespace, and at the end of the failover, this new *HA-PP* will be the new anchor point.

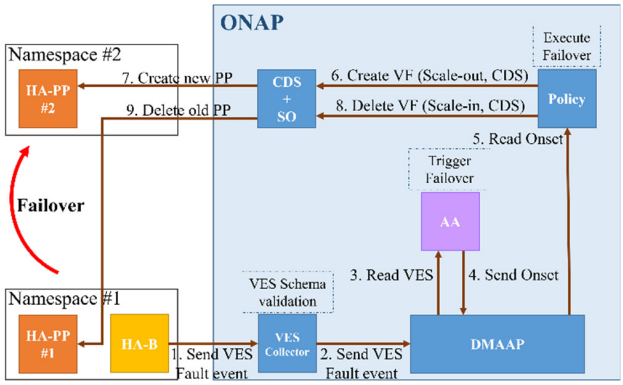

Figure 5 – ONAP-based execution workflow of the failover use case

Failover execution steps from the ONAP point of view are depicted in Figure 5 with the following explanation:

1. VES fault message is sent out from *Home Agent Backend (HA-B)*.
2. VES collector catches VES fault message, does scheme validation, and sends it out to DMAAP.
3. AA reads the corresponding DMAAP topic continuously, whether or not a new message arrives. If yes, based on its own logic, it decides what Policy Model to trigger via the Onset message.
4. Onset message is sent out to the corresponding topic of DMAAP.
5. The Policy framework reads that topic in DMAAP.
6. The Policy calls the Scale-out workflow (Create VF module).
7. CDS and SO create the new VF module (Helm chart) instance in a different namespace.
8. The Policy executes Scale-in workflow in CDS.
9. CDS and SO delete the existing VF module (Helm chart) in the original namespace. The failover management procedure is finished.

Figure 6 shows the network traffic during failover. The content of the data plane is emulated by ICMPv6 messages. First, *MN* attaches to *HA-PP#1*. When *HA-PP#1* fails, a new *Binding Update (BU)* is sent out to *HA-PP#2* from *MN,* and the *IPv6inIPv6 tunnel* is set up to the new *HA-PP*. The *CN* is now reachable via *HA-PP#2*. *MN* can detect *HA* failure as routing is no longer working towards the tunnel. The details of the IPv6 address allocation during failover can be found at the end of the paper in Figure 12 and Figure 13. *Home Address* (*HoA)* is represented by the tunnel endpoint IPv6 address of MN, which is permanent. After the failover, the routing in connection with the *HA-PP* and *CN* is modified as the *MN,* and *HoA* is reachable via a different forwarding entity (different *HA-PP).* Of course, this can be updated by any dynamic routing protocol or static routing, but this is not in the scope of this paper.

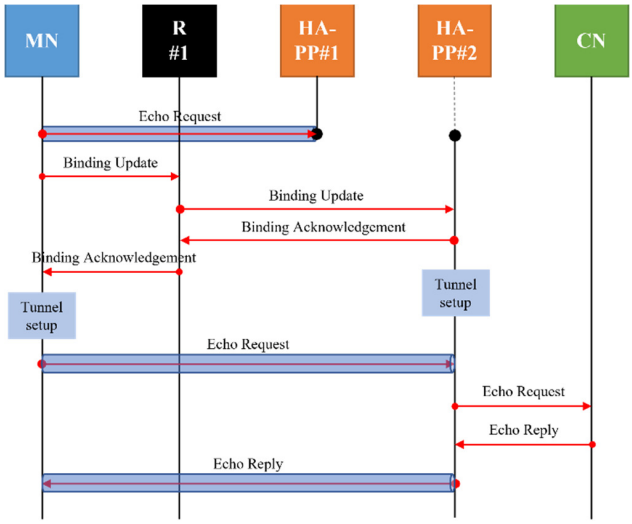

Figure 6 – CN-MIPv6 signaling flow in case of ONAP- based failover

A complete test framework is used with several measurement points for the process mentioned above. The exact measurement steps of failover can be seen at the end of the paper in Figure 14. The test framework is a separate entity measuring service outage time. By service outage, we mean the time when the bidirectional tunnel is broken and the new one is set up on a newly created *HA-PP*. Furthermore, the test framework connects to all the elements to initialize BU and adjust routing before and after failover, details in Figure 12, Figure 13, Figure 14, Figure 15, Figure 16 and Figure 17.

We have executed the failover scenario 100 times. The corresponding box plot can be seen in Figure 7. Meanwhile, numerical results are shown in Table 1. The failover is executed within 106.26 sec on average (median: 103.93, stdev: 30.42. The maximum is 210.19 sec, while the minimum is 58.83 sec. During measurements, we experienced that, sometimes, ONAP waited for an uncertain time during the same process. This explains the stdev and the high difference between the minimum and maximum.

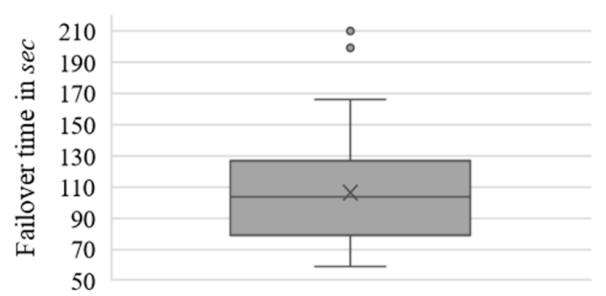

Figure 7 – Failover time results

TABLE I
NUMERICAL RESULTS OF FAILOVER TIME (SEC)

MIN	AVG	Median	MAX	STDEV
58.83	106.26	103.92	210.19	30.42

C. *Utilisation for Scaling*

Scaling is the workflow when additional executors are added or removed to/from the system in order to deal with the changed traffic. In the case of our CN-MIPv6 mobility management service, we emulated how to add a new *HA-PP* when a certain traffic threshold is reached. At the end of the automatic scaling, both *HA-PP* components will serve the traffic.

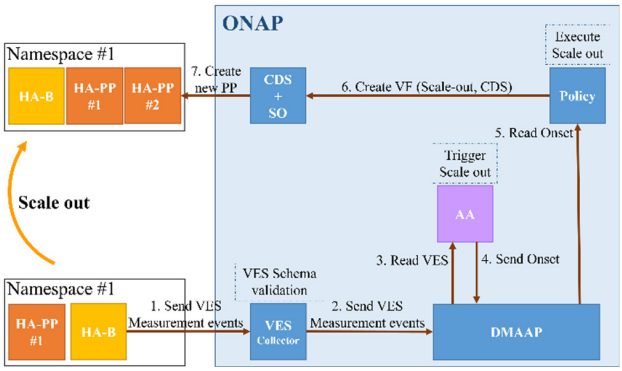

Figure 8 – ONAP-based execution workflow of scaling

Scaling execution steps from the ONAP point of view - shown in Figure 8 – are the followings:

1. *HA-B* sends out VES Measurement event.
2. VES Collector validates its schema and puts it to the corresponding DMAAP topic.
3. AA gets the VES message and calculates if scaling is needed or not.
4. AA sends out the Onset message to DMAAP to trigger a Policy model.
5. The Policy gets the Onset message.
6. The Policy calls for scale-out workflow in CDS.
7. CDS and SO executes the scale-out request and create a new VF module instance in a different namespace.

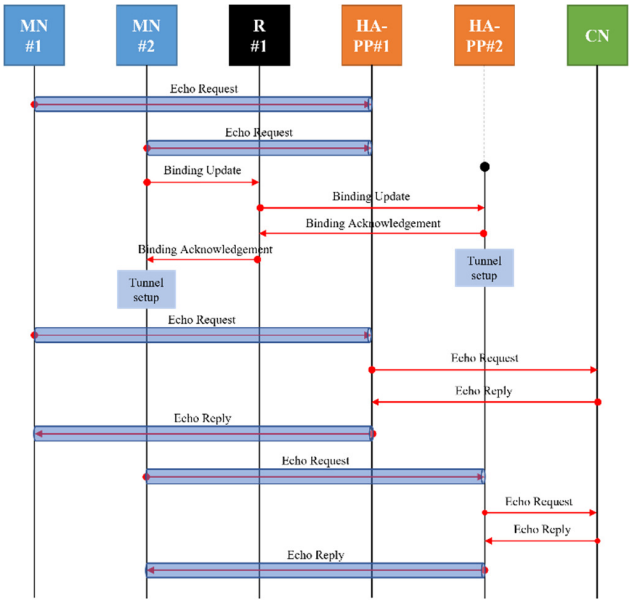

Figure 9 – Signaling of CN-MIPv6 in case of scaling

Figure 9 shows how scaling is executed from the CN-MIPv6 point of view. First, two *MNs (MN#1, MN#2)* are attached to one *HA-PP (HA-PP#1)*. After the ONAP recognizes the overload of *HA-PP#1*, then the management system creates a new one. From this point, *MN#2* is reregistered via sending BU to the newly created *HA-PP#2*, and the bidirectional tunnel is set up to *HA-PP#2* in the case of *MN#2*. Thus, the *CN* is reachable via *HA-PP#2* in the case of *MN#2*.

At the end of this paper, Figure 15 and Figure 17 present the IPv6 address allocation during scaling. Similar logic can be seen for IPv6 address allocation as in the case of failover.The detailed measurement steps of the scaling use case can be found at the end of the paper in Figure 16. This also uses the same elements as the failover scenario. Meanwhile, Pods' network interface capacities are limited to 1 Gbps to simulate the limited capacity of a network device and packet processing. The original throughput was about cc 900 Mpbs for the *MNs* using one *HA-PP* instance. This leads to a common base on measurement, and it is easier to compare results. The scaling scenario has been executed 20 times. Figure 10 and Figure 11 show the box plot of the results in Mbps and percentage, respectively. Table 2 shows the numerical results of the average gain for both *MNs*. After the scaling, the throughput gain for *MN#1* is 362.1 Mbps (67.94%), while on *MN#2,* it is 329.5Mbps (65.4%). *MN#2* shows a little less throughput gain, as it has an unconnected period which includes new binding message exchanges and tunnel setup time, while *MN#1* is continuously connected to the *HA-PP#1*.

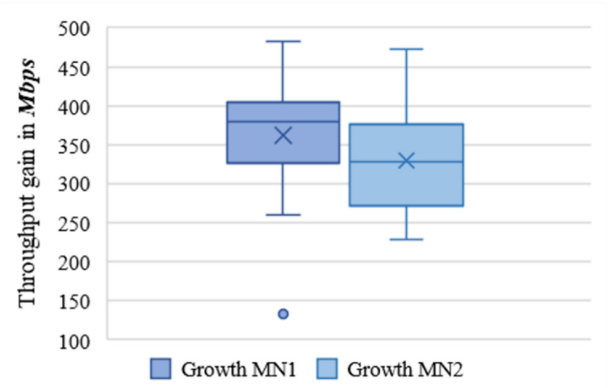

Figure 10 – Measured throughput gain after scaling out (in Mbps)

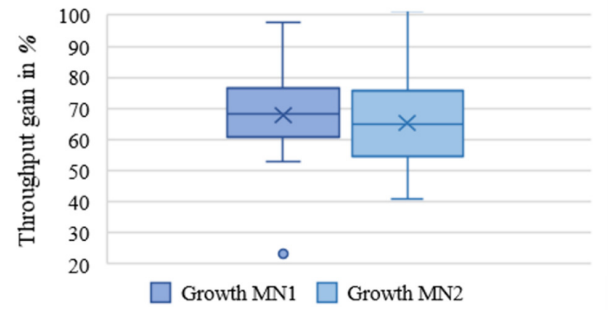

Figure 11 – Measured throughput gain after scaling out (in percentage)

TABLE II
NUMERICAL RESULTS OF AVERAGE THROUGHPUT GAIN AFTER SCALING

MN1 throughput gain (Mbps)	MN1 throughput gain (%)	MN2 throughput gain (Mbps)	MN2 throughput gain (%)
362.1	67.94	329.5	65.4

D. Availability and redundancy calculations

The availability of a telecommunication system is one of the most critical quality metrics. By automating the failover procedure of a network function, the availability of the network function significantly improves.

A short recap, the availability of a single NF can be calculated by

$$A_S = \frac{MTBF}{\text{MTBF+MTTR}} \quad (1)$$

where MTBF is Mean Time Between Failures, MTTR is Mean Time to Repair.

Telco services provide high availability; for Ulra-Reliably Low-Latency (URLLC) services, these expectations are growing further. When the availability expectations for service are in the range of five or six nines for the service components, including the network function software, the availability expectation is even higher. These figures are achieved by applying redundancy schemes. There are no widely agreed/accepted figures for software MTBF and

MTTR values, so we calculated with a range of values (MTBF from a month to two years, MTTR from 5 minutes to 45 minutes). We applied different redundancy schemes for these values to ensure that the availability of the redundant solutions is above six nines. Obviously, the reduced MTTR in every case improves the availability. This improvement allows the deployment of lighter redundancy schemes for the network function when its failover is automated and the same availability is still provided. For example, a 2N redundant deployment allows deploying a 3+1 redundant scheme, and the same availability is provided (thus, for four software instances, three instances become active instead of two [36] [37]). Suppose the original MTTR was on the higher end (i.e., above 15 minutes). Even the 6+1 redundant scheme provides the same availability (for all MTBF values) as the 2N redundant solution without failover automation. Thus a significant amount of resources can be saved. Note that it is also possible to use the "saved availability budget" for other components of the system (e.g., lowering hardware availability by employing less personnel and saving cost) and keep end-to-end availability on the same level or simply offer better availability for customers.

With 2N and 3N redundant systems, the availability calculation is the following:

$$A_{2N} = 1 - (1 - A_S)^2 \quad (2)$$

$$A_{3N} = 1 - (1 - A_S)^3 \quad (3)$$

In Section IV. B, we have shown the average time for failover, which is 106.23 sec. This value can be considered as the MTTR of the function we evaluate.

The goal of these calculations is the following: in our ONAP-based failover case, it is possible to reduce the level of redundancy. This means the number of deployed instances can be decreased in order to save resources. Meanwhile, the availability of service is not jeopardized. For calculation simplicity reasons, we apply 99.999% availability for NF, but in a real-world scenario, higher availability is expected.

1N non-redundant case:

Based on the above-mentioned equations, we calculated the MTBF value for the 1N system if availability must be at least 99.999%, which is 123 days. This means the frequency of system collapse cannot be less than 123 days; otherwise, the system availability cannot reach 99.999%, supposed the cc.106 sec failover time

We also consider that this MTBF value describes the by-default behavior of our software system.

2N redundant case:

With the MTTR=106.23 sec value and 2N redundancy, the MTBF value is 9.3 hours ($MTBF_{2N}$=9.3 hours) if minimum 99.999% availability is kept. Thus, if the $MTBF_{2N}$ is 9.3 hours or less, the system is below the target of 99.999% availability. So, if every 9.3 hours, there is an failure with the given MTTR value, the system will still operate on at least 99.999% availability.

3N redundant case:

With the MTTR=106.23 sec value and 3N redundancy, the MTBF value is 1.3 hours ($MTBF_{3N}$=1.3 hours) if minimum 99.999% availability is kept. Thus, if the $MTBF_{3N}$ is 1.3 hours or less, the system is below the target of 99.999% availability. So, if every 1.3 hours, there is an error with the given MTTR value, the system still operates on at least 99.999% availability.

We can save resources if the same availability can be kept with fewer redundant pairs. If we know that our system has an error on average every 123 days, then let's check the maximum MTTR value that is allowed with 2N redundancy to keep at least 99.999% availability. This is 9.365 hours ($MTTR_{2N}$=9.365 hours). This means that every 2N system with MTBF = 123 days can be reduced to our 1N system, where the MTTR is minimum 9.365 hours. Because this MTTR value reaches a certain level, where the availability of the 2N system does not keep 99.999% even though it may be better than the 1N, but the over-availability requirement is not fulfilled. But using 1N instead of 2N means a service outage as there is no active pair to maintain service. To circumvent this, a 3N redundancy system is needed. A similar logic is applied: if the MTBF = 123 days, then let's calculate the min MTTR when the 3N system does not add availability gain (not reaching 99.999%) compared to the 2N system. Every 3N system can be reduced to a 2N system where the MTTR is between 12 min and 65 hours. The whole calculation is represented in Figure 18 at the end of the paper.

V. CONCLUSION

In this paper, we have presented that the redundancy restoration time (restoring a cold backup) can be decreased to the level of minutes with network automation and orchestration. Note that according to the GSMA [38], a new instance deployment of a Physical Network Function lays in the range of days. In this paper, we have shown that this can be decreased to the range of minutes. This definitely shows the power of network automation; meanwhile, cloud, virtualization, and containerization are utilized as well.

This failover time is also considered to restore the cold backup of a particular network function. We also believe that this does not only pertain to CN-MIPv6; general conclusions can be drawn for any NF.

Even though we have shown that the redeployment time has gone to the range of minutes, there are further possibilities for optimization: detailed measurements are needed to conclude the minimum value of liveness and readiness probe to minimize the failover or scaling time. Right now, they are arbitrary. A new measurement point can also be added to the test framework, which watches the ONAP internal states to identify the slowest part of the execution accurately. Failover time may be higher in a real-world scenario because our testbed does not deal with complex routing and configuration. This is similar to the scaling as well because the gain may be lower due to the cloud's actual computing and network load uncertainty. We have also shown that, with the help of the scaling procedure, significant throughput gain can be reached.

Improvement of failover time also has a positive impact on service availability. It saves resources for the automated NF function or any other part of the chain of processes contributing to service provision.

VI. FUTURE WORK

Current Kubernetes network approaches are not "network-native". More and more novel Kubernetes networking approaches are emerging [39] [40], which are the natural evolution steps of the current version of our PoC implementation. For a simplified but more advanced use case, using an external load balancer instead of a direct Pod connection is also in the scope of future implementations.

ONAP has also paved the way for machine learning applications, which may not only deal with traffic level optimization in failover or scaling. Machine learning can add new perspectives to predictive mobility management and other potential application areas.

REFERENCES

[1] 'Open Network Automation Platform (ONAP)'. https://www.onap.org/ (accessed Jul. 28, 2022).

[2] C. Perkins, D. Johnson, and J. Arkko, *Mobility Support in IPv6*. IETF, 2011. [Online]. Available: http://www.ietf.org/rfc/rfc6275.txt

[3] Á. Leiter, L. Bokor, and I. Kispál, 'An Evolution of Mobile IPv6 to the Cloud', in *Proceedings of the 18th ACM Symposium on Mobility Management and Wireless Access*, New York, NY, USA, 2020, pp. 137–141. DOI: 10.1145/3416012.3424633.

[4] Á. Leiter et al., 'Cloud-native IP-based mobility management: a MIPv6 Home Agent standalone microservice design', in *2022 13th International Symposium on Communication Systems, Networks and Digital Signal Processing (CSNDSP)*, 2022, pp. 252–257. DOI: 10.1109/CSNDSP54353.2022.9908059.

[5] K. Du, X. Wen, L. Wang, and T.-T. Nguyen, 'A Cloud-Native Based Access and Mobility Management Function Implementation in 5G Core', in *2020 IEEE 6th International Conference on Computer and Communications (ICCC)*, 2020, pp. 1251–1256. DOI: 10.1109/ICCC51575.2020.9345262.

[6] S. Gundavelli (Ed.), K. Leung, V. Devarapalli, K. Chowdhury, and B. Patil, *Proxy Mobile IPv6*. Fremont, CA, USA: RFC Editor, 2008. DOI: 10.17487/RFC5213.

[7] Á. Leiter, N. Galambosi, and L. Bokor, 'An Evolution of Proxy Mobile IPv6 to the Cloud', in *Proceedings of the 19th ACM International Symposium on Mobility Management and Wireless Access*, New York, NY, USA: Association for Computing Machinery, 2021, pp. 107–115. [Online]. Available: DOI: 10.1145/3479241.3486684

[8] S. Kim, H. Choi, P. Park, S. Min, and Y. Han, 'OpenFlow-based Proxy mobile IPv6 over software defined network (SDN)', in *2014 IEEE 11th Consumer Communications and Networking Conference (CCNC)*, Jan. 2014, pp. 119–125. DOI: 10.1109/CCNC.2014.6866558.

[9] S. M. Raza, D. S. Kim, D. Shin, and H. Choo, 'Leveraging proxy mobile IPv6 with SDN', *Journal of Communications and Networks*, vol. 18, no. 3, Art. no. 3, Jun. 2016, DOI: 10.1109/JCN.2016.000061.

[10] K. M. Sue, S. Kamolphiwong, T. Kamolphiwong, and L. Damyos, 'SDN Based Fast Handover over IP Mobility', in *2019 23rd International Computer Science and Engineering Conference (ICSEC)*, 2019, pp. 345–350. DOI: 10.1109/ICSEC47112.2019.8974787.

[11] K. Hee Lee, 'Mobility Management Framework in Software Defined Networks', *International Journal of Software Engineering and Its Applications*, vol. 8, no. 8, pp. 1–10.

[12] Á. Leiter, M. S. Saleh, L. Pap, and Bokor, 'Survey on PMIPv6-based Mobility Management Architectures for Software-Defined Networking', *Infocommunications Journal*, vol. XIV, no. 2, DOI: 10.36244/ICJ.2022.2.1.

[13] D. Giatsios, K. Choumas, P. Flegkas, T. Korakis, and D. Camps-Mur, 'SDN implementation of slicing and fast failover in 5G transport networks', in *2017 European Conference on Networks and Communications (EuCNC)*, 2017, pp. 1–6. DOI: 10.1109/EuCNC.2017.7980671.

[14] V. Q. Rodriguez, F. Guillemin, and A. Boubendir, 'Automating the deployment of 5G Network Slices using ONAP', in *2019 10th International Conference on Networks of the Future (NoF)*, 2019, pp. 32–39. DOI: 10.1109/NoF47743.2019.9015043.

[15] V. Q. Rodriguez, F. Guillemin, and A. Boubendir, '5G E2E Network Slicing Management with ONAP', in *2020 23rd Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*, 2020, pp. 87–94. DOI: 10.1109/ICIN48450.2020.9059507.

[16] R. Banerji et al., 'ONAP Based Pro-Active Access Discovery and Selection for 5G Networks', in *2020 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*, 2020, pp. 1–6. DOI: 10.1109/WCNCW48565.2020.9124724.

[17] H. Huang and S. Guo, 'Proactive Failure Recovery for NFV in Distributed Edge Computing', *IEEE Communications Magazine*, vol. 57, no. 5, pp. 131–137, 2019, DOI: 10.1109/MCOM.2019.1701366.

[18] M. Reitblatt, M. Canini, A. Guha, and N. Foster, 'FatTire: Declarative Fault Tolerance for Software-Defined Networks', in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, New York, NY, USA, 2013, pp. 109–114. DOI: 10.1145/2491185.2491187.

[19] 'Kubernetes: Horizontal Pod Autoscaler'. https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/ (accessed Jul. 15, 2022).

[20] 'Deployment element of Kubernetes'. https://kubernetes.io/docs/concepts/workloads/controllers/deployment/ (accessed Mar. 12, 2021).

[21] J. W. Mwangoka, P. Marques, and J. Rodriguez, 'Cognitive Mobility Management in Heterogeneous Networks', in *Proceedings of the 8th ACM International Workshop on Mobility Management and Wireless Access*, New York, NY, USA, 2010, pp. 37–44. DOI: 10.1145/1868497.1868504.

[22] M. Simsek, M. Bennis, and I. Guvenc, 'Mobility management in HetNets: a learning-based perspective', *EURASIP Journal on Wireless Communications and Networking*, vol. 2015, no. 1, p. 26, Feb. 2015, DOI: 10.1186/s13638-015-0244-2.

[23] R. Boutaba, N. Shahriar, M. A. Salahuddin, S. R. Chowdhury, N. Saha, and A. James, 'AI-Driven Closed-Loop Automation in 5G and beyond Mobile Networks', in *Proceedings of the 4th FlexNets Workshop on Flexible Networks Artificial Intelligence Supported Network Flexibility and Agility*, New York, NY, USA, 2021, pp. 1–6. DOI: 10.1145/3472735.3474458.

[24] J. Heinonen, P. Korja, T. Partti, H. Flinck, and P. Pöyhönen, 'Mobility management enhancements for 5G low latency services', in *2016 IEEE International Conference on Communications Workshops (ICC)*, 2016, pp. 68–73. DOI: 10.1109/ICCW.2016.7503766.

[25] Chantel Soumis, 'AMM: WHA T IS AUTOMA TED MANAGEMENT?' https://www.valicomcorp.com/blog/2018/4/2/amm-what-is-automated-mobility-management (accessed Mar. 21, 2022).

[26] Pallavi Vanacharla, 'SP360: Service Provider Winning business customers with automated mobility management'. https://blogs.cisco.com/sp/winning-business-customers-with-automated-mobility-management (accessed Mar. 21, 2022).

[27] Pallavi Vanacharla, 'Digital Transformation Automation: Moving beyond manual mobility management'. https://blogs.cisco.com/digital/automation-moving-beyond-manual-mobility-management (accessed Mar. 21, 2022).

[28] 'Kubernetes Operators'. https://kubernetes.io/docs/concepts/extend-kubernetes/operator/ (accessed Jan. 15, 2022).

[29] 'An introduction to closed-loop automation'. https://developer.ibm.com/articles/an-introduction-to-closed-loop-automation/ (accessed Mar. 21, 2022).

[30] 'ONAP – Service Orchestrator'. https://wiki.onap.org/pages/viewpage.action?pageId=1015834 (accessed Jul. 15, 2022).

[31] 'ONAP – Service Design and Creation'. https://wiki.onap.org/pages/viewpage.action?pageId=1015837 (accessed Jul. 15, 2022).

[32] 'ONAP – Controller Design Studio'. https://docs.onap.org/projects/onap-ccsdk-cds/en/latest/ (accessed Jul. 15, 2022).

[33] 'ONAP – Policy Framework'. https://docs.onap.org/projects/onap-policy-parent/en/latest/architecture/architecture.html (accessed Jul. 15, 2022).

[34] 'ONAP - Data Movement as a Platform'. https://wiki.onap.org/pages/viewpage.action?pageId=3247130 (accessed Jul. 15, 2022).

[35] F. Slim, F. Guillemin, A. Gravey, and Y. Hadjadj-Aoul, 'Towards a dynamic adaptive placement of virtual network functions under ONAP', in *2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, 2017, pp. 210–215. DOI: 10.1109/NFV-SDN.2017.8169880.

[36] A. Hilt, I. Bakos, and G. Járó, 'Reliability and availability modelling of telecommunication servers on cloud', in *2015 17th International Conference on Transparent Optical Networks (ICTON)*, 2015, pp. 1– 4. DOI: 10.1109/ICTON.2015.7193412.
[37] J. Varga, A. Hilt, J. Bíró, C. Rotter, and G. Jaro, 'Reducing operational costs of ultra-reliable low latency services in 5G', *Infocommunications Journal*, vol. X, pp. 37–45, 2018, DOI: 10.36244/ICJ.2018.4.6.
[38] GSMA, 'Migration from Physical to Virtual Network Functions: Best Practices and Lessons Learned'. https://www.gsma.com/futurenetworks/5g/migration-from-physical-to-virtual-network-functions-best-practices-and-lessons-learned/ (accessed Jul. 15, 2022).
[39] 'Kubernetes – Network Speciel Interest Group'. https://github.com/kubernetes/community/tree/master/sig-network (accessed Sep. 28, 2021).
[40] 'Network Service Mesh'. https://networkservicemesh.io/ (accessed Mar. 12, 2021).

Ákos Leiter has graduated as a Computer Engineer MSC in Department of Networked Systems and Services (HIT), Budapest University of Technology and Economics (BME) in 2015, specialized in Computer Networks. His theses was about proposing an operator-centric, dynamic flow mobility protocol with IP in Evolved Packet Core. Currently he is a PhD candidate at the Department of Networked Systems and Services in the Multimedia Networks and Services Laboratory (MEDIANETS) and a Research Engineer at Nokia Bell Labs. His main research field is Network Function Virtualization and Software Defined Networking including Orchestration and Network Automation. His work-in-progress Phd theses is about the cloudification of Mobile IPv6 protocol family on the top of Kubernetes.

Edina Lami is a solution expert at Nokia, technical lead of the Nokia Core Slicing team since 2022. She received her MSc degree in computer engineering from Budapest University of Technology and Economics in 2022. She wrote her thesis about the closed-loop orchestration of cloud-native IP-based mobility management. She joined Nokia Bell Labs in 2020 as a Research Engineer Trainee where she was involved in mobility management and orchestration related research topics.

Attila Hegyi received the MSc in computer science and mathematics from the University of Szeged (SZTE) in 2010. He had worked for multiple companies in the telecommunication domain as software engineer and currently he is a senior research engineer at Nokia Bell Labs. His main research topics are in the field of cloud-native network automation, multi-cloud orchestration and edge computing.

József Varga, PhD, is a senior research engineer at Nokia, member of the 'Multi Cloud Orchestration' research group in Nokia Bell Labs. He received his MSc in computer science and mathematics from the University of Szeged in 1991, PhD in IT from the University of Veszprém in 2002. He was an assistant professor at the University of Szeged, then at the University of Veszprém. He joined Nokia in 1999, he was involved in IP Multimedia Subsystem development, then represented Nokia as a standardization delegate in 3GPP from 2004 to 2011. In 2011 he joined Nokia Research Center (now Nokia Bell Labs) dealing with topics like SDN, virtualization, and orchestration. Currently he is focusing on resource management in 6G, including the economic aspects. He co-authored more than 10 papers and more than 10 granted patents.

László Bokor received his Ph.D. degree in computer engineering from Budapest University of Technology and Economics in 2014. He is currently an associate professor at the Dept. of Networked Systems and Services where he leads the Commsignia-BME HIT Automotive Communications Research Group. He is a member of the HTE, the Hungarian Standards Institution's Technical Committee for ITS (MSZT/MB 911), the TPEGoverC-ITS Task Force within the TPEG Application Working Group of TISA, the ITS Hungary Association, and the BME's MediaNets Laboratory. In recognition of his professional work and achievements in mobile telecommunications, he received the HTE Silver Medal (2013), the HTE Pollák-Virág Award (2015, 2022), and the HTE Gold Medal (2018). He was a recipient of the UNKP-16-4-I. Post-Doctoral Fellowship in 2016 from the New National Excellence Program of the Ministry of Human Capacities of Hungary. In 2018 he was awarded the Dean's Honor (BME VIK) for education and research achievements in the field of communication of autonomous vehicles; in 2020, he received the BME HIT Excellence in Education Award.

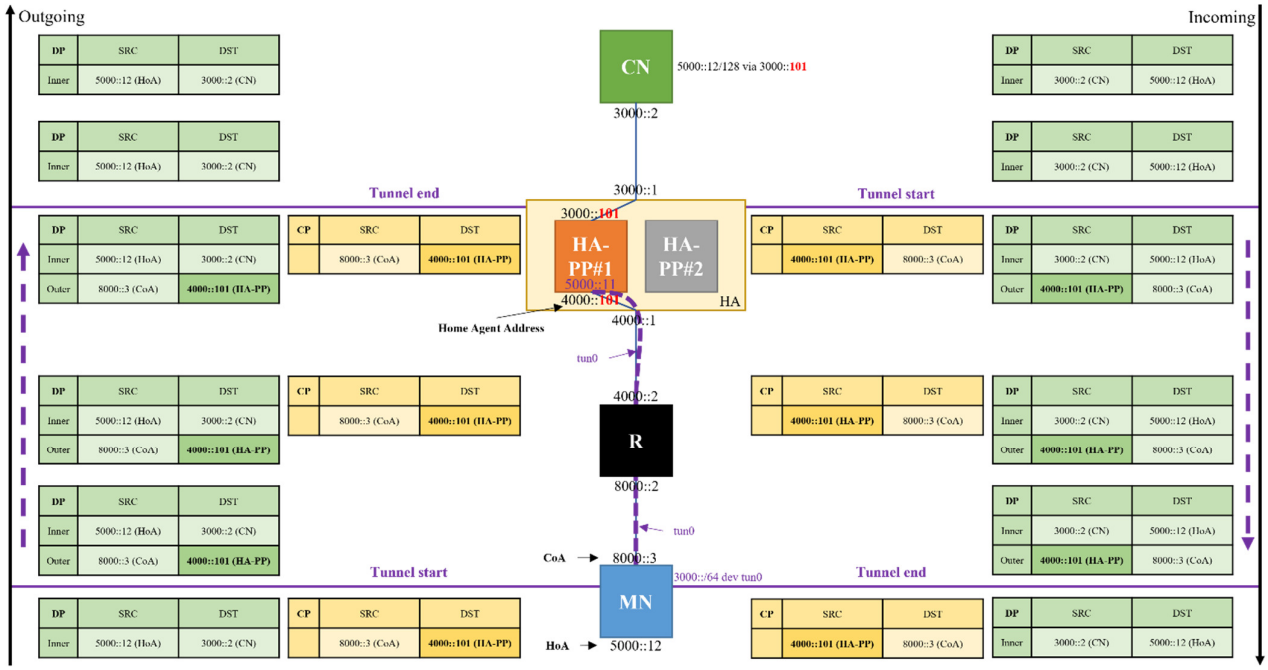

Figure 12 – IPv6 address allocation before failover

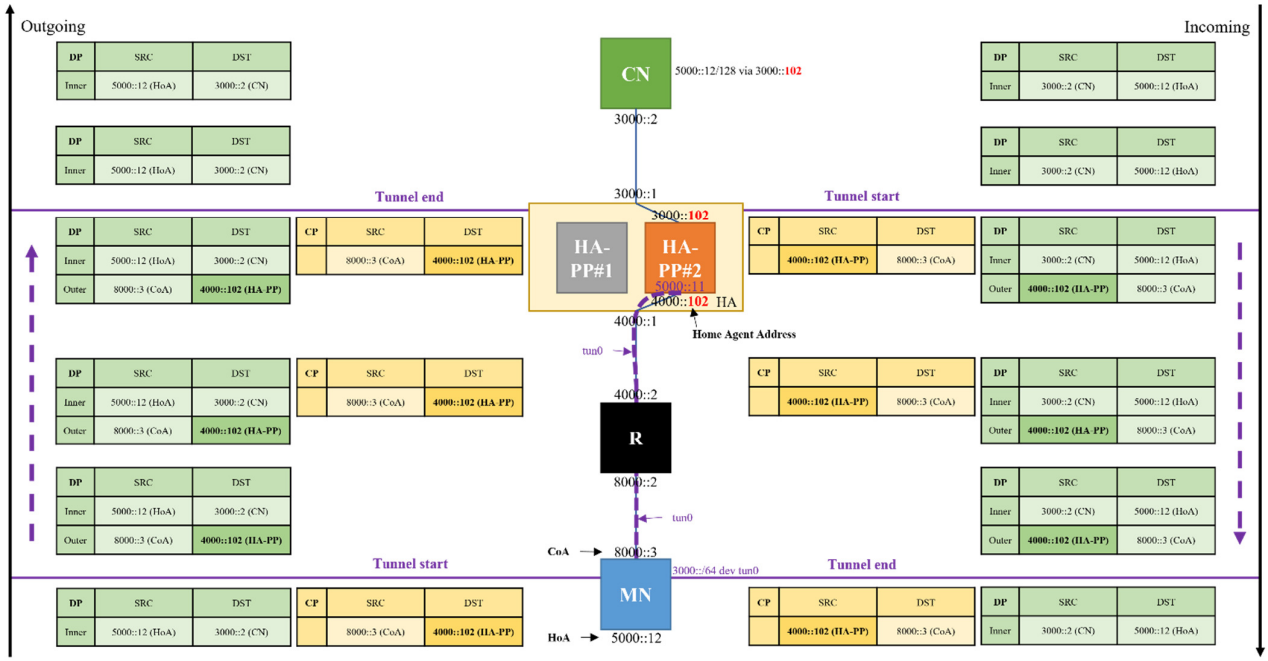

Figure 13 – IPv6 address allocation after failover

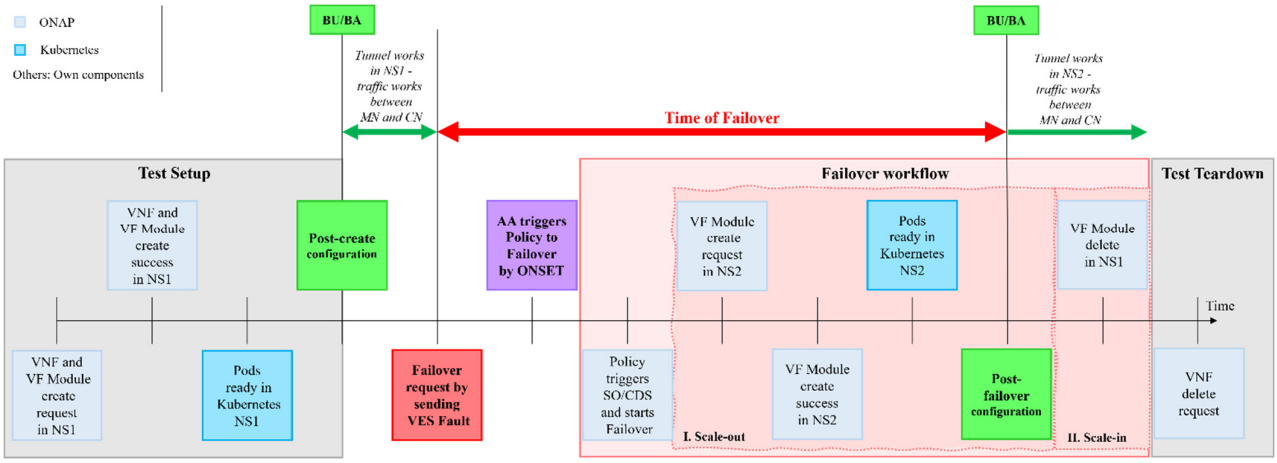

Figure 14 – Detailed description of the failover use case

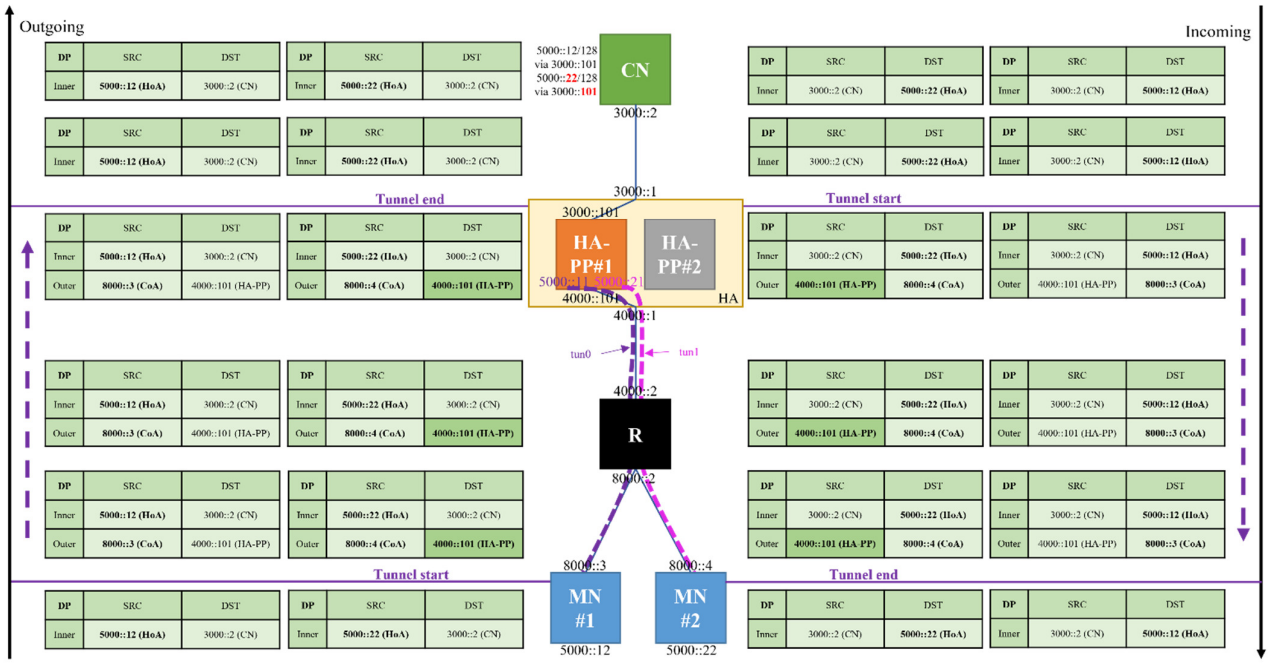

Figure 15 – IPv6 address allocation before scaling

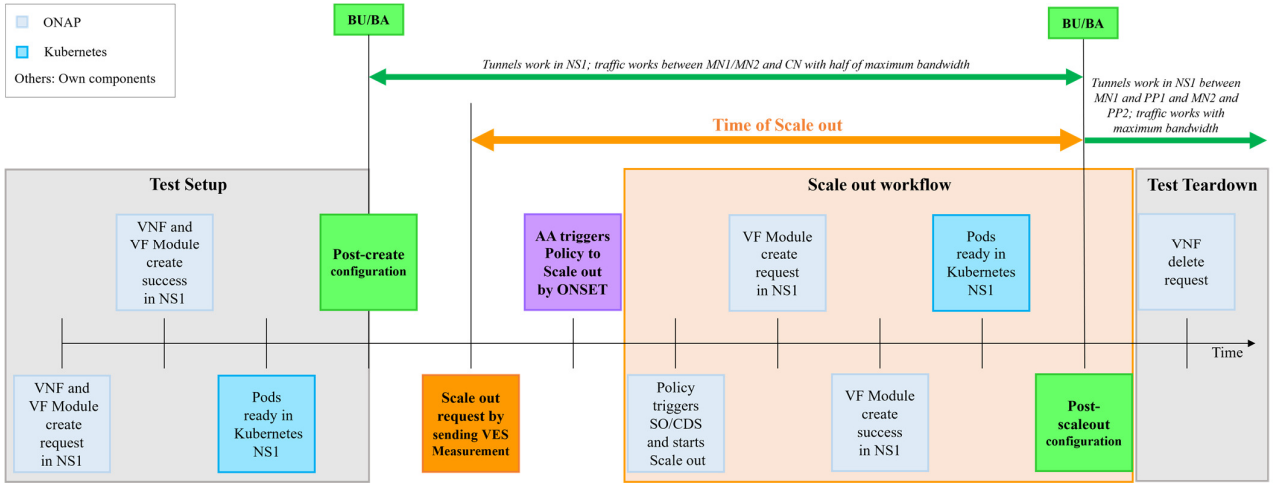

Figure 16 – Detailed description of the scaling use case

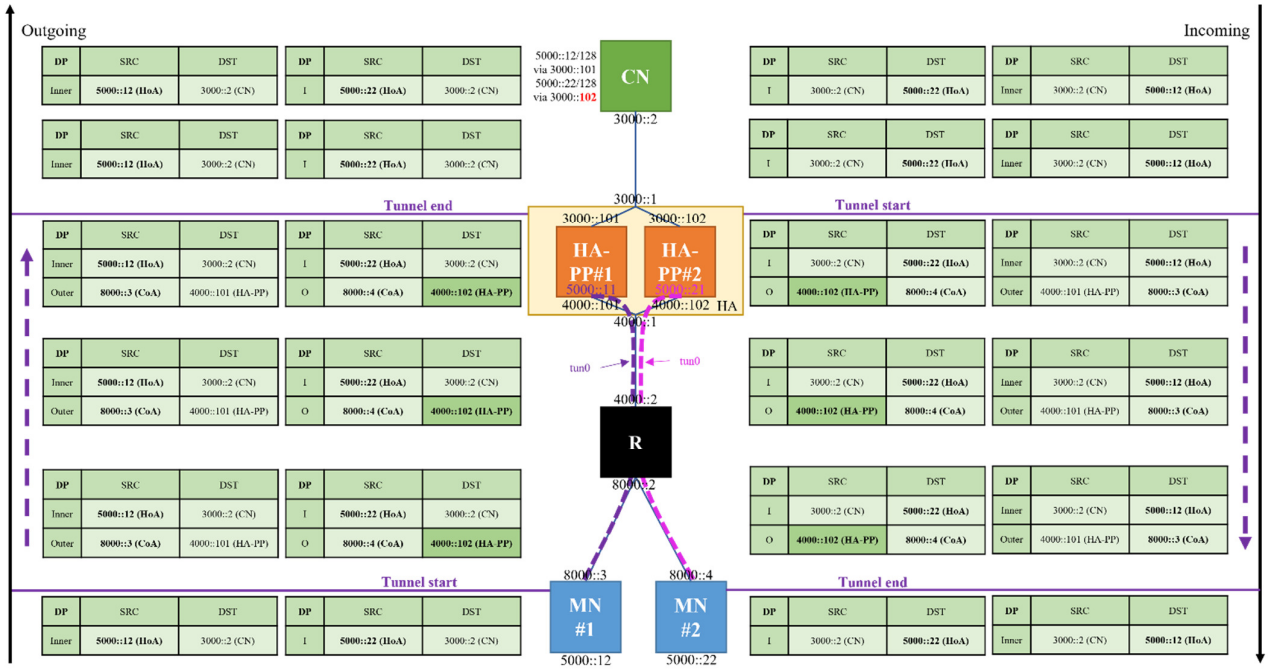

Figure 17 – IPv6 address allocation after scaling

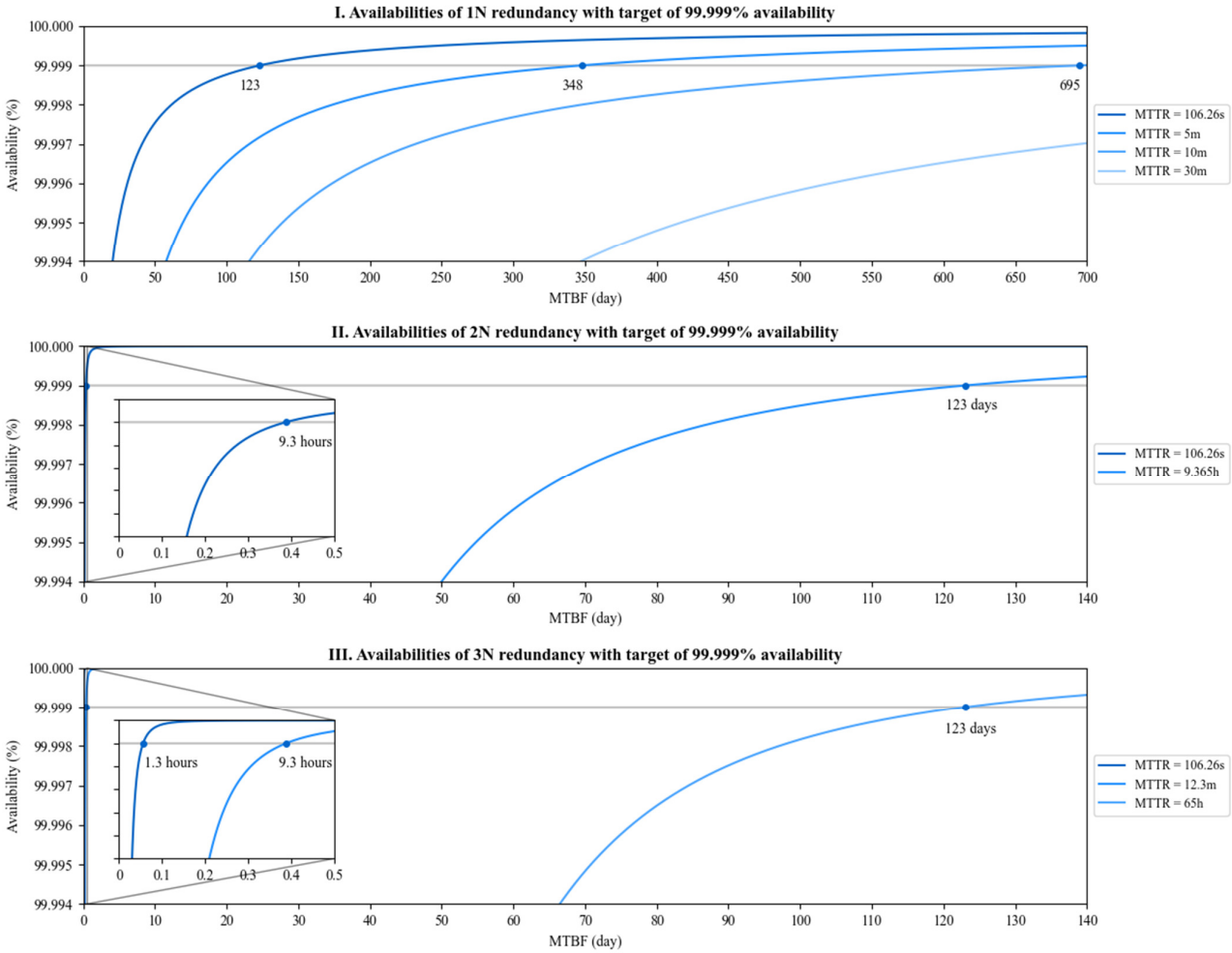

Figure 18 – Availability and redundancy calculations