A Priori State Synchronization for Fast Failover of Stateful Firewall VNFs

Nicholas Gray*, Claas Lorenz†, Alexander Müssig*, Steffen Gebert*, Thomas Zinner*, Phuoc Tran-Gia*
*University of Würzburg, Würzburg - {nicholas.gray,alexander.muessig,steffen.gebert,zinner,trangia}@informatik.uni-wuerzburg.de
†genua GmbH, Kirchheim - claas_lorenz@genua.de

***Abstract*—Network Functions Virtualization (NFV) replaces physical middleboxes with software instances running network functions in cloud environments. To support this new paradigm, it is necessary to port the code basis from highly specialized hardware devices to virtual machines running on COTS hardware. In order to fully exploit the inherent capabilities of cloud environments it is further necessary to redesign the software to support a large amount of distributed, cooperating function instances instead of single, isolated and monolithic instances. This development can be observed for network functions like stateful firewalling. Until now, available software firewalls lack support for active/active operation in clustered environments, which hinders horizontal scalability. This is due to the fact that the required synchronization of connection states among the cluster's instances is an impediment that still has to be resolved. Therefore, this work investigates different synchronization strategies and mechanisms, which allow to share connection states among the cluster to maintain scalability and high-availability.**

I. Introduction

Current firewall functions are implemented as hardware middleboxes and located within the physical network paths. To protect the network from external threats, the firewall is placed at the perimeter where different security boundaries are connected. In order to ensure reliable connectivity, such firewalls are deployed in high availability (HA) setups using 1:1 protection. Therefore, two identical devices are deployed, one as active instance forwarding the traffic, while the other is running as dedicated backup instance, ready to take over once the active instance fails. Replication of all connection states of a stateful firewall towards the standby device allows this instance to continue operation within minimal downtime in case of a failure. The connected hosts automatically communicate with the currently active instance, as the whole cluster uses one virtual IP address, which the active instance occupies and announces using ARP. As only half of the deployed resources can be used and scalability is limited, an increase of capacity often requires a replacement of both physical devices. Further, they are placed within the physical path and thus are hard to exchange without service interruption.

To tackle these drawbacks, which occur for firewalling as well as for other types of middleboxes, the concept of *Network Function Virtualization* (NFV, [1]) receives increased popularity. The tightly integrated middlebox appliances are replaced by flexible *Virtualized Network Functions* (VNFs) running as software instances. Such a VNF deployment benefits from the advantages of cloud computing, i.e., scalability, improved energy efficiency, and increased flexibility. Currently available software implementations already perform very efficient packet and state inspection [2]. However, these implementations do not yet allow fine-grained, horizontal scalability, as expected by NFV implementations. Following the example of hardware firewalls, their scalability is limited to a 1:1 protection with a dedicated backup instance for failover.

Different architectures for stateful NFV-based firewalling are proposed in [3]. In this context, the synchronization of connection states among all cluster firewall nodes is stated as a major challenge. Such synchronization is required for seamless failover, as well as scale in and scale out procedures. This stems from the fact that every packet can induce a transition of the connection state, which needs to be propagated among all cluster nodes. Therefore, effective strategies for this synchronization are required and discussed, as well as evaluated, in the remainder of this work. Based on a proof-of-concept implementation, which allows to select particular synchronization strategies and levels, the effects on the data plane performance are evaluated. The evaluations vary the degree of state synchronization regarding different TCP connection states, as well as the access strategy to the shared state. Furthermore, the influence of increased synchronization load caused by different cluster sizes is investigated.

The remainder of the work is structured as follows: Section II describes current software firewall implementations and their synchronization strategies, before detailing the concept of a firewall VNF based on cloud principles in Section III. The proof-of-concept implementation used to evaluate the effects on the data plane performance is described in Section IV. This is followed by the description and discussion of the results in Section V-B, which are conducted by running in a testbed. Finally, Section VI concludes this work and gives directions for future work.

II. Background and Related Work

The feature set of a firewall implementation can be categorized as follows:

Packet filter: Based on a static rule set, all packets matching these rules are forwarded. For every packet, only the configured rule set is investigated. This very limited approach can be easily implemented using SDN, as no dynamic provisioning of rules to the networking devices is required and only source and destination IP addresses and transport layer ports are checked.

Stateful firewall: Packet headers up to the transport layer are investigated and checked against the static rule set, as well as the list of active connections. Any packet which can

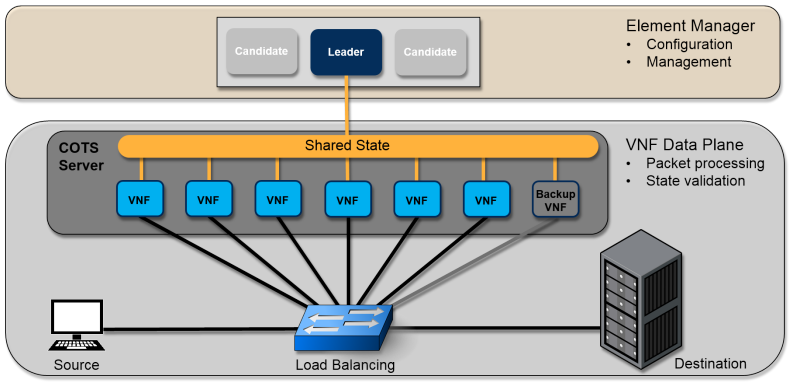


Fig. 1: Concept of the distributed VNF firewall including load balancing among horizontally scaled VNF instances and cluster of element manager instances.

be assigned to an active connection, including the reverse direction, is allowed to pass through. By validating the TCP state machine, the conformance with the protocol is checked based on the flags in the TCP headers. An implementation only using SDN features is possible, yet slow [3].

Application-layer firewall: The most complex variant of firewalling investigates the application layer protocol, e.g., HTTP. Due to the complexity of these protocols, such firewall implementations require tremendous amounts of CPU power and often cause false positives.

Given the disadvantages of application-layer firewalling, most networks use stateful firewalling, which is also investigated in this work.

An implementation of a DMZ, a special firewall setup, based on ClickOS [4] is described in [5]. The authors conclude that current virtualization infrastructure does not yet provide satisfying features for NFV deployments. While the scalability aspect of VNFs is mentioned, no further discussion is led. In [6] the authors investigate synchronization mechanisms merely of the state of application instead of the entire virtual machine, to provide fast failovers. In contrast, a guideline for migrating the local state of NFVs to a shared memory cluster is discussed in [7]. OpenNF [8] provides a framework for developing and synchronizing the state of NFV applications in SDN-enabled networks. The performance of the *OpenDaylight* SDN controller in a clustered setup is investigated in [9]. The authors investigate the behavior in case of database partitioning as well as controller failover.

III. VNF-based Cloud Firewall

In highly dynamic cloud environments with frequently changing virtual networks, the premise of a clearly definable gateway position is not given anymore. In this work, we focus on the realization of the *VNF-centric* approach proposed in [3]. By applying means of *Software Defined Networking* (SDN) and NFV, this serves as a foundation for implementing a scalable and fault tolerant cloud firewall.

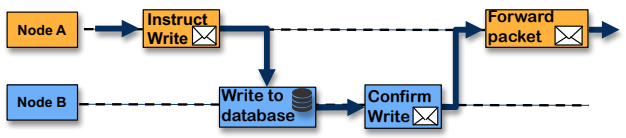
A. Concept of a VNF-based Firewall

Figure 1 illustrates the proposed architecture of the proposed VNF-based cloud firewall. It is comprised of two dimensions – the packet processing on the data plane and the control and management plane functionality on top.

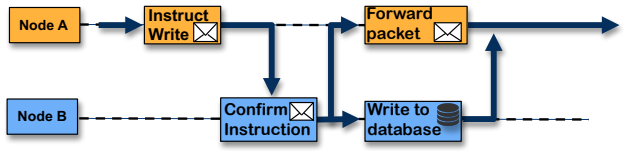
In the data plane, the packets flowing through the network are processed by the cluster consisting of multiple virtual firewall instances running on *Commodity-off-the-Shelf* (COTS) server hardware. All of these VNF instances have access to the shared state table and could potentially take over any connection that was previously handled by another instance. Depending on the resource usage, additional instances can be added or removed as needed. Each VNF instance connects to the firewall's *element manager* (EM, [10]), which is responsible for coordinating the firewall cluster.

The element manager takes over control and management functionality by monitoring the cluster state. This includes the configuration of the firewall rules, supervision of the health of connected VNF instances, as well as signaling additional resource demands to the NFV infrastructure. The EM cluster itself applies a leader election to decide about the particular instance, which actively coordinates the whole cluster, dictates the configuration, and serves as representative to external entities.

On the network side, load balancing functionality provided by the SDN is used to distribute the load among the firewall instances. The load balancing applies hashing to ensure that all packets of a connection are forwarded to the very same instance, which ensures that synchronization delays have no negative effect during normal operations. In the case of scale in or scale out, the load balancer's buckets are recalculated so that the IP address space is equally distributed among the instances. One, or in large clusters more, instances are running as shared backup to take over the traffic of a failing instance, but until then only participate in the state synchronization passively. In contrast to reconfiguring the load balancing in the SDN switches to equally distribute the traffic among the remaining instances, such 1:1 replacement can be done with a single request, e.g., using OpenFlow's group action. This $n+1$ setup offers a good trade-off between resource utilization and fast failover.



(a) Synchronous.



(b) Asynchronous.

Fig. 2: Data write access strategies.

B. State Synchronization Strategies

Given the need of a low-latency, a priori state synchronization among all cluster nodes in conjunction with high rates of incoming connections, the need for effective synchronization mechanisms is apparent. A multitude of options exist, how this shared state can be managed and accessed. These options will be described in the following and evaluated later on.

1) Synchronization Level: Due to the stateful traffic processed by the firewall cluster, an implementation choice is given by the granularity of state changes shared with the other instances. Regarding the TCP states, the following options are available:

Full: Any state change is synchronized among the whole cluster, i.e., all TCP state changes imposed by the three-way handshake and connection termination.

Established: Only the long-lasting state after the successful handshake, as well as information about successfully closed connections is propagated within cluster-wide, i.e. the `ESTABLISHED` and `CLOSED` states. All intermediate states are only held locally.

2) Synchronization Strategy: The required consistency of the state table update among the cluster nodes offers another option with potential influence on data plane performance.

Sync: After sending information about the new connection, the instance waits for write confirmations of *all* other instances, as illustrated in Figure 2a. This ensures that this particular state change is available on the backup instance, in case of its own failure.

Async: After sending information about the new connection, the instance waits only for the confirmation that all other nodes received this update, ignoring the fact that they did not persist this change, yet. This procedure is illustrated in Figure 2b.

Based on these options, the firewall cluster can be optimized for a consistent operation or tuned towards increased performance. Besides these, further implementation-specific options can also have an influence on a distributed VNF implementation.

IV. Implementation

In this section the prototypical implementation[1] of the firewall VNF, which is later used to evaluate the previously described synchronization options, is described. Additional options offered by the particular software stack, which have an influence on the consistency versus performance trade-off are discussed.

This prototype is built in Erlang/OTP [11], a functional language designed for high availability and concurrency. As implementation for the shared state holding the firewall rules as well as the particular status of every currently active TCP connection, Erlang's built-in Mnesia database is used. Mnesia operates as in-memory database, offers high throughput key/value access, and allows SQL-style transactions to ensure consistent writes. Replication of data towards other cluster nodes can be specified on a per-table basis. The two components, namely the element manager and firewall VNF instances, are described in the following.

A. Element Manager (EM)

As described in Section III-A, the EM is responsible for configuration and management of the overall firewall cluster. In the current implementation, high-availability mechanisms are out of focus and only a single instance is used. The leader election part can be added by using an established distributed consensus algorithm like Raft [12].

The EM instance reads its configuration from a file, which includes the desired cluster size, as well as the static firewall rules. It listens for other Erlang VMs to join its cluster and register as firewall instances, as well as instances leaving the cluster. In each of these events, the current number of active and backup instances is evaluated, and potentially additional actions are triggered. As described in Section III-A, a defined number of instances are kept as backup for failure cases, while any change regarding the active instances receiving traffic, the load balancer is updated. Therefore, the EM connects to the built-in REST API of the Ryu controller [13], which manages the connected OpenFlow switches.

B. VNF Instances

The VNF instances execute the actual filtering functionality by inspecting all incoming traffic and forwarding only permitted packets towards their destination. During startup, the VNF instance connects to the EM and receives its runtime configuration from there. After synchronizing the shared state provided by the other cluster nodes, the VNF is ready to receive traffic. Whether an instance is used actively for packet processing or remains in the status of a backup instance is the EM's decision. For evaluating different synchronization modes, the prototypical implementation offers two different database tables to hold the state of active TCP connections. One table is replicated among all cluster nodes, while the other is held locally only. Depending on the currently investigated synchronization mode, the particular table is used to store, e.g., intermediate states like `SYN-RECEIVED` with the whole cluster or only keep this information for the current node.

[1]The source code is uploaded upon paper acceptance

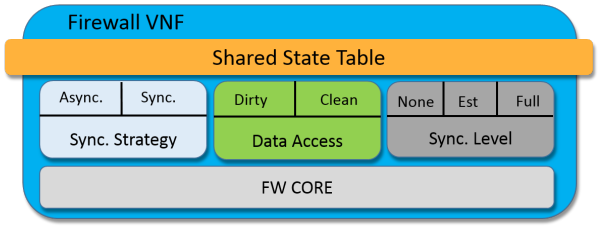


Fig. 3: Implemented layers to access the shared state including available options.

For receiving packets from the network interface, a RAW socket is opened, which provides the complete Ethernet frames to the firewall instance. As this implementation does not terminate TCP connections, no TCP stack is required. Nevertheless, the headers of Layer 2 - 4 need to be parsed in order to be compared against the configured rules and also the flags of incoming packets are checked against the TCP state machine. For performance reasons, the state table is checked for an active connection first. If no connection is found, the rule table is consulted, which is by far more complex, as wild card rules need to be evaluated. In case the packet is part of an already existing or new, but valid connection, it is forwarded towards the destination using the VNF instance's egress interface. Otherwise, the packet is dropped. For every state change, this information is stored in the particular table, which is then synchronized with the other cluster nodes or not.

C. Implementation-specific Synchronization Options

In addition to the synchronization strategies described in Section III-B, the used Erlang Mnesia database offers another option that might have an influence on the consistency versus performance trade-off. Similar to SQL databases, write operations into the shared state can be done using transactions or via a direct access, potentially loosing consistency during concurrent access. Furthermore, a third synchronization level *None* is implemented, which serves as a baseline for the evaluation and keeps all connection states only on the local node, where even the `ESTABLISHED` state is not shared. The resulting options of the prototypical implementation are depicted in Figure 3 and evaluated for their influence on the connection setup times in the next section.

V. Evaluation

After describing the used testbed setup, the previously introduced options for state synchronization will be evaluated regarding their influence on the connection setup times.

A. Methodology

To evaluate the different synchronization strategies implemented in the firewall VNF, measurements in the testbed depicted in Figure 4 were conducted. The element manager, the firewall instances, as well as the hosts generating the data plane traffic are running as KVM-based virtual machines on one physical server[2]. The VMs[3] are running Erlang 18 on Debian Linux 8.5 and use the VirtIO network driver. For generating the data plane traffic that is filtered by the firewall cluster, HTTP requests using *A*pache Bench (ab, [14]) are sent at a specified concurrency level towards a Varnish HTTP caching server [15], which operates completely in memory to avoid any bottlenecks. By configuring ab's concurrency level, different load can be induced on the firewall cluster. An instance of *Open vSwitch* running on the host system acts as a load balancer and is managed by the Ryu SDN controller.

One variant of this testbed (cf. Section V-B3) is an expansion towards additional physical and virtual hosts running within the same local network. To evaluate the influence of an increased cluster size with instances running on remote hosts, additional instances that only participate in the synchronization are launched within an OpenStack cloud.

The clients implemented by Apache Bench establish a TCP connection to the Varnish server and download a 1 Byte HTML file. As the keep-alive mechanism is disabled, the server immediately tears down the connection after serving the file. Per run, a total of 10,000 downloads are initiated and each run is repeated 10 times. Based on this, the following figures mark the 95% confidence interval. The connection setup times, which are the objective of all following evaluations, are measured within the firewall and denote the time interval between receiving the client's `SYN` packet and its second packet, which completes the three-way handshake.

B. Results

Based on measurements, the synchronization strategies are evaluated regarding their impact on the connection setup times when passing through the firewall VNF. At first, the impact of asynchronous and synchronous write modes to the shared state are investigated, followed by the different levels of synchronizing particular TCP states. Finally, the influence of different VNF cluster sizes and the impact on the experienced connection setup times is described.

1) Influence of Write Modes to Shared State: The synchronization level is set to only share the `ESTABLISHED` state among the cluster. Figure 5 shows the measured connection setup times for load levels of 25 and 100 concurrent connections. As expected, the dirty data access outperforms the more consistent transactional data access by 2 ms and up to 15 ms for a load levels of 25 and 100 concurrent connections respectively. Furthermore, it can be seen that the number of concurrent connections has significantly higher impact on the connection setup times than the chosen synchronization strategy. The comparison of the different synchronization methods shows that the synchronous write mode performs worse than the asynchronous approach when used in combination with the dirty write mode, as the instance processing a packet has to wait for a write confirmation by all other instances. In contrast, the combination of synchronous writes together with transactional data access poses an exception to this observation. For both concurrency levels, the asynchronous strategy results in higher connection setup times.

2) Influence of Synchronization States: Figure 6 details the investigation of connection setup times. Again, the y-axis displays the mean connection setup time measured at the

[2]Dual Intel Xeon L5420, four cores at 2.50 GHz, 16 GB RAM
[3]EM: 2 CPU cores, 1 GB RAM; VNF instances: 4 CPU cores, 3 GB RAM

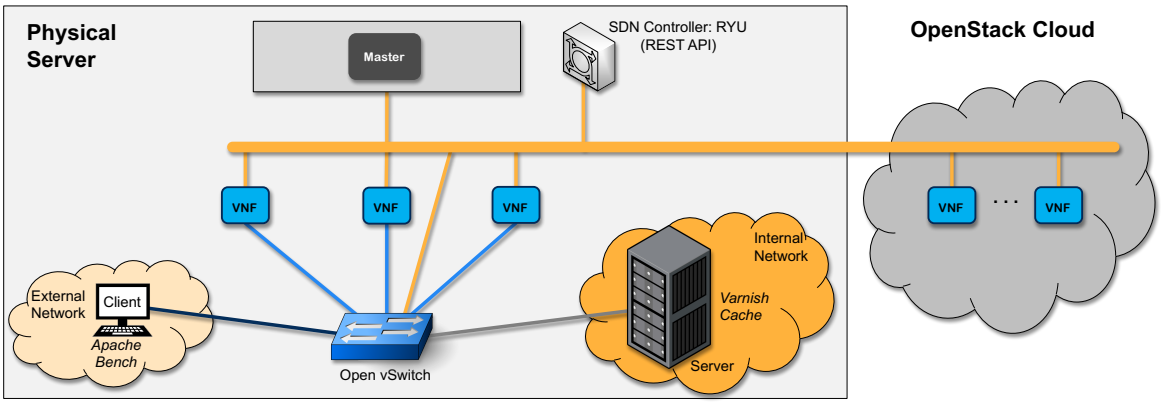


Fig. 4: Testbed for conducting performance measurements.

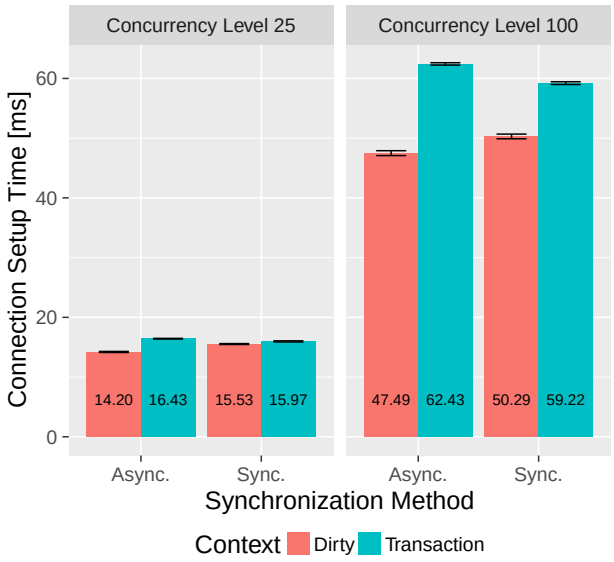


Fig. 5: Average connection setup times for varying load levels when using different data access and write modes.

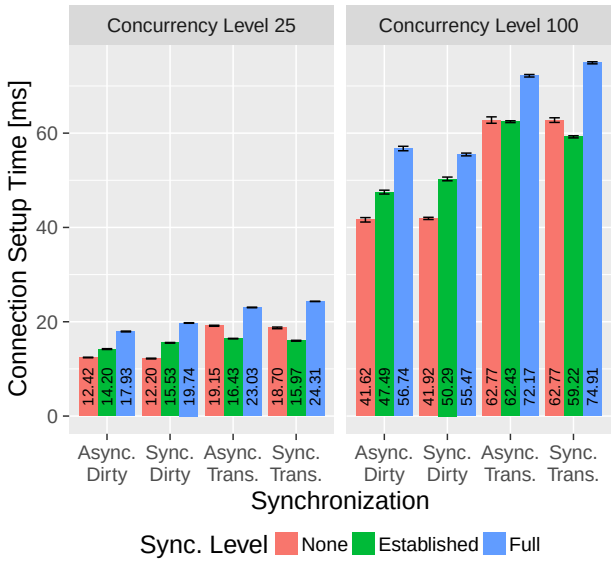


Fig. 6: Varying synchronization levels when using different synchronization and access strategies.

firewall instances and the x-axis groups the varying synchronization strategies. Within one group, the bar color reflects the synchronization level of different TCP states.

In most cases, higher connection setup times are observed for the more comprehensive synchronization levels *Established* and *Full*, as more state changes are synchronized with the entire cluster. Synchronizing only the `ESTABLISHED` state introduces the risk of connection resets during the initial handshake, while it reduces the average setup time by roughly 16% in the case of *async dirty*. The calculated risk of an inconsistency during failover situations seems worthy, as the client's operating system will automatically try another connection attempt without the user noticing.

As seen before, the exception is the transaction context regarding synchronous versus asynchronous writes, however, the differences are very minor with overlapping confidence intervals in many cases. Even more prevalent is the faster connection setup in the case of the *Established* synchronization level for the transactional data access experiments compared to no state sharing among the cluster at all. A possible explanation for this is the split into a local and a shared state table (cf. Section III-B) so that write operations can be parallelized by the Mnesia database. As the *None* level only uses the local table, the transactional write access requires serialized processing, resulting in higher connection setup times.

3) Influence of Cluster Size: Based on the evaluations described before, it can be seen that the amount, as well as the strategy of state synchronization can significantly influence VNF performance. One critical feature of VNFs has yet been neglected in the evaluations: The requirement to scale the network function horizontally across many more instances to

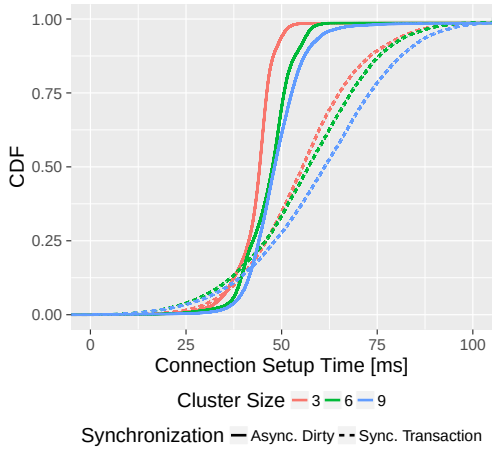


Fig. 7: Influence of cluster size for 100 concurrent connections.

distribute the load and allow to dynamically adapt to the resource demands. Therefore, the following evaluation expands the previously local cluster consisting of two firewall VNF and adds additional instances running on remote hosts in an OpenStack cloud within the same LAN.

Figure 7 depicts the cumulative distribution functions of the average connection setup times for 100 concurrent connections. For the sake of readability, only the strategies optimized for performance (*async dirty*) and consistency (*sync transaction*) are illustrated using solid and dashed lines respectively. The individual line colors represent the results for a varying cluster size of 3, 6, and 9 firewall VNFs, while all but one instance are only synchronizing without receiving any traffic. Still, the effect of a growing cluster is clearly visible and induces additional synchronization overhead for the instance processing the traffic resulting in augmented connection setup times. The results show that the performance-oriented configuration performs similar in 20% and significantly better in 80% of the cases than the consistent configuration.

VI. Conclusion and Future Work

Traditionally used hardware firewalls have drawbacks due to their physical deployment model and resulting scalability limitations. NFV address these challenges by implementing network functions in software that can be operated and scaled in a cloud-like environment. The feature set of software firewall implementations, however, lacks the ability to run in clustered environments, where multiple instances can be used for processing traffic. Based on our previous work [3], implementation options for VNF-based firewalling approach with a focus on state synchronization among the instances were discussed and evaluated.

In this work, different strategies regarding the amount of TCP states, as well as the write strategies to the shared state were investigated. The results indicate that a synchronization strategy aiming at performance allows up to 20% lower TCP connection setup times than using a synchronization strategy that enforces consistency across all instances within the cluster. In addition, the influence of the cluster size was investigated, which supported the previous results.

Future work should focus on evaluating an alternative in-memory store, like *memcached* or *Redis*, as well as the effects caused by sacrificing consistency and resulting issues during instance failover. Furthermore, [3] introduces a hybrid VNF approach, which relies on dynamically offloading of traffic-intense flows to the switching hardware. Building upon the implementation of the current work, novel research question regarding offloading strategies for effective use of the limited hardware resources should be investigated.

Acknowledgment

This work has been performed in the framework of the KMU-Innovativ project SarDiNe funded by the BMBF. The authors alone are responsible for the content of the paper.

References

[1] ETSI, "Network functions virtualisation - an introduction, benefits, enablers, challenges & call for action," Oct. 2012.

[2] S. Gebert, A. Müssig, S. Lange, T. Zinner, N. Gray, and P. Tran-Gia, "Processing time comparison of a hardware-based firewall and its virtualized counterpart," in *8th EAI International Conference on Mobile Networks and Management (MONAMI 2016)*, Abu Dhabi, United Arab Emirates, October 2016.

[3] C. Lorenz, D. Hock, J. Scherer, R. Durner, W. Kellerer, S. Gebert, N. Gray, T. Zinner, and P. Tran-Gia, "An SDN/NFV-enabled Enterprise Network Architecture Offering Fine-Grained Security Policy Enforcement," *IEEE Communications Magazine*, vol. (to be published), 2016. [Online]. Available: http://www.comnet.informatik.uni-wuerzburg.de/staff/members/steffen_gebert/

[4] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, M. Honda, R. Bifulco, and F. Huici, "ClickOS and the art of network function virtualization," in *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'14, Berkeley, CA, USA, 2014.

[5] L. Bondan, C. R. P. dos Santos, and L. Z. Granville, "Management requirements for ClickOS-based network function virtualization," in *10th International Conference on Network and Service Management (CNSM) and Workshop*. IEEE, 2014, pp. 447–450.

[6] S. Rajagopalan, D. Williams, and H. Jamjoom, "Pico replication: A high availability framework for middleboxes," in *Proceedings of the 4th annual Symposium on Cloud Computing*. ACM, 2013, p. 1.

[7] M. Kablan, B. Caldwell, R. Han, H. Jamjoom, and E. Keller, "Stateless network functions," in *Proceedings of the 2015 ACM SIGCOMM Workshop on Hot Topics in Middleboxes and Network Function Virtualization*. ACM, 2015, pp. 49–54.

[8] A. Gember-Jacobson, R. Viswanathan, C. Prakash, R. Grandl, J. Khalid, S. Das, and A. Akella, "Opennf: Enabling innovation in network function control," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 163–174, 2015.

[9] D. Suh, S. Jang, S. Han, S. Pack, T. Kim, and J. Kwak, "On performance of OpenDaylight clustering," in *2016 IEEE NetSoft Conference and Workshops (NetSoft)*, June 2016.

[10] "Network functions virtualisation (nfv); management and orchestration; report on architectural options," ETSI, ETSI GS NFV-IFA 009 V1.1.1, July 2016. [Online]. Available: http://www.etsi.org/deliver/etsi_gs/NFV-IFA/001_099/009/01.01.01_60/gs_NFV-IFA009v010101p.pdf

[11] "Erlang programming language." [Online]. Available: http://erlang.org/

[12] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference*, Berkeley, CA, USA, 2014.

[13] "Ryu 4.5 documentation - RESTful API," 2016. [Online]. Available: http://ryu.readthedocs.io/en/latest/app/ofctl_rest.html

[14] "ab - apache HTTP server benchmarking tool," Aug. 2016. [Online]. Available: https://httpd.apache.org/docs/2.4/programs/ab.html

[15] "Varnish HTTP cache." [Online]. Available: https://varnish-cache.org/