SynAPTIC: Secure And Persistent connecTIvity for Containers

Alireza Ranjbar*†, Miika Komu*, Patrik Salmela*, Tuomas Aura†
*Ericsson Research, Finland †Aalto University, Finland
{alireza.ranjbar, miika.komu, patrik.salmela}@ericsson.com, tuomas.aura@aalto.fi

Abstract—**Cloud virtualization technology is shifting towards light-weight containers, which provide isolated environments for running cloud-based services. The emerging trends such as container-based micro-service architectures and hybrid cloud deployments result in increased traffic volumes between the micro-services, mobility of the communication endpoints, and some of the communication taking place over untrusted networks. Yet, the services are typically designed with the assumption of scalable, persistent and secure connectivity. In this paper, we present the SynAPTIC architecture, which enables secure and persistent connectivity between mobile containers, especially in the hybrid cloud and in multi-tenant cloud networks. The solution is based on the standardized Host Identity Protocol (HIP) that tenants can deploy on top of existing cloud infrastructure independently of their cloud provider. Optional cloud-provider extensions based on Software-Defined Networking (SDN) further optimize the networking architecture. Our qualitative and quantitative evaluation shows that SynAPTIC performs better than some of the existing solutions.**

Index Terms—**containers, docker, security, HIP, SDN, mobility**

1. Introduction

Cloud virtualization has been dominated by hypervisor-based virtual machines (VMs) to offer secure and inexpensive services. However, during the recent years, container technology has been catching up by offering more agile and lightweight services with higher visibility to users and application developers. In general, containers are leaner than virtual machines, and more dense deployments of containers are possible per host machine. Since the containers share the system resources of the host, they introduce new communication and security challenges when deployed in large cloud networks. Despite all the attempts to boost the deployment of containers in cloud networks, the connectivity between containers is still at the early stage of development and several challenges remain to be solved.

Containers change the way applications are developed, and these changes highlight the importance of container networking. Applications are gradually moving from monolithic to microservice architectures in which a single application is split into several smaller services. This development goes hand in hand with the deployment of containers, which provide suitable isolated and light-weight execution environments. However, these changes in the application architecture can increase the traffic volumes due to the chatty communications between the microservices, which has the risk of reducing the overall service performance [8], [27].

A service implemented by a container can be accessed externally based on Network Address Translation (NAT) techniques which may lead to several networking issues such as port collisions and overlapping private address realms. These problems are aggravated by the fact that containers are sometimes deployed inside virtual machines and the hypervisor may introduce yet another NAT layer. The cloud provider can, of course, implement various overlay mechanisms to interconnect containers even in nested NAT topologies, and also provide container migration support across the network boundaries. However, solving these problems manually requires extra configuration and administrative overhead.

In hybrid cloud solutions, the tenant has its own private cloud infrastructure and it may employ the public cloud only to manage peak loads. Furthermore, the tenant may wish to move its workloads between different cloud vendors. Thus, containers need to move between different cloud infrastructures, and the mechanism for interconnecting containers should be independent of the cloud service provider. To avoid vendor lock-in, the solutions for such inter-cloud connectivity should be primarily managed by the tenant, and they should be based on standard protocols.

The inter-cloud connectivity highlights the need for NAT and firewall traversal support in the containers, and the need for secure end-to-end connectivity. Data plane security, such as domain isolation and access control, provided by the cloud vendor is also not sufficient when the containers communicate with each other across domain boundaries and over the Internet. Thus, the inter-container communication should be secured end-to-end to protect the data flows, e.g., between microservices [4].

Our Contribution. In this paper, we propose SynAPTIC, a solution to provide secure and persistent connectivity for containers. SynAPTIC uses the Host Identity Protocol (HIP) [24] to meet the challenges in connecting containers. HIP is a standardized mobility and security protocol that separates identity from location and provides unique identifiers for end-hosts. A host identifier in HIP is derived from a cryptographic public key that enables strong authentication. HIP supports NAT traversal and end-host mobility in both IPv4 and IPv6 networks, and HIP secures network connectivity with IPsec. As our first contribution, we integrate HIP in Docker containers and design an architecture that can in practice be deployed on containers without any changes in networking infrastructure. In addition, to ensure compatibility with the latest networking technologies, SynAPTIC supports the use of Software Defined Networking (SDN), which is being integrated in networking stacks of some of the well-known container solutions [11], [12]. In our SDN-based approach, the network infrastructure facilitates centralized access control based on the public-key identifiers of the containers. Also, the SDN controller assists in routing the data flows and updating the existing forwarding paths in case of container mobility.

The rest of the paper is structured as follows. Section 2 explains the state-of-the-art solutions. We introduce SynAPTIC architecture on traditional and SDN based cloud networks in Section 3. We evaluate our solution in Section 4. Section 5 compares SynAPTIC with some of the existing solutions and describes possible future improvements. Finally, Section 6 concludes the paper.

2. Related Work

In this section, at first, we review some of the cloud networking protocols and then we discuss the existing container solutions. Lastly, we explain the recent inter-container networking approaches based on SDN.

Networking protocols: To provide logical isolation between tenants in a cloud network, Virtual Extensible Local Area Network (VXLAN) [23] has been developed which creates a layer-2 overlay network on top of layer-3 network. A set of other protocols such as Virtual Private Network (VPN) [18] aim to enhance security of data flows by utilizing IPSec or SSL and by creating secure connectivity between private and public clouds routed through untrusted networks. Furthermore, a set of protocols such as Locator/ID Separation Protocol (LISP) [17] have been developed to separate the identity of end-hosts from their network location.

Host Identity Protocol (HIP) [25] collects a number of features from other protocols within a single protocol. HIP separates the end-host's identity from its location and provides cryptographic authentication and Denial of Service (DoS) protection and encryption for application payloads, and supports end-host mobility in communication networks. Also, HIP supports heterogeneous addressing, and it can traverse through NATted networks.

Container connectivity: By default, Docker [2] connects containers through a bridged interface. Also, tenants can use VXLAN and SSL/TLS to create logically isolated and secure networks and connect containers that are placed in different host machines. Weave [14] allows Docker containers to communicate with each other through a set of so called peer routers in a mesh-like topology. Weave is able to forward the data flows through the fast path based on VXLAN tunnels and it is able to encrypt the control and data flows between weave routers. The project Calico [1] does not employ any overlay network but instead, it provides a direct path by configuring the Linux IP tables. Also, Calico installs a BGP client on host machines to distribute routing information through a data center. Kubernetes from Google [6] places containers in so called Pods, each of which has a unique IP address. Kubernetes enables the containers in the same pod to communicate with each other through the "localhost".

Linux Containers (LXC) [7] is an alternative implementation for containers. Similarly as in docker, LXC containers running on the same host are connected via a shared bridge interface. FlockPort [5] allows two LXC containers behind a NATted network to establish secure communication channels based on IPSec VPN tunnels.

Software-Defined Networking: SDN and overlay technologies such as VXLAN have been widely used for networking between VMs [22] and recently some solutions have applied SDN to enhance network portability and provide seamless connectivity between containers. OpenContrail [11] and Pertino [12] are examples that benefit from centralized SDN architecture and overlay networks in order to provide network connectivity in a container-based cloud. Kim et al. [21] employ SDN as a solution for flexible address rewriting in container-based cloud networks. Nakagawa et al. [26] use a centralized controller to distribute logical end-point configurations to physical switches in order to connect Linux containers.

3. Connectivity Architecture for Containers

In this section, we explain our container connectivity solution in detail. First, we explain how HIP can be deployed on existing (non-SDN and non-HIP-aware) cloud infrastructure. Then, we describe a HIP-aware software-defined networking solution.

3.1. Solution for Today's Cloud Networks

To begin with, we review the structure of HIP protocol for inter-container communications. HIP adds a new name space between the transport and network layer and separates the identity of end-hosts from their location. While containers use the IP addresses as location identifiers, HIP assigns Host Identities (HI) based on public keys for identifying the end-hosts. This HI value is self-certifying by its nature, and it can be used to uniquely and securely identify an end-host in a cloud network. However, as public keys can be of varying size, they are difficult to employ as surrogate virtual addresses in existing network applications, and therefore, HIP specifications define fixed-sized identifiers. For IPv6-based applications, HIP generates a hash of the HI which is referred as the Host Identity Tag (HIT). A HIT is the same size as an IPv6 address (128 bits), so it can be utilized by IPv6-capable applications running in the containers, and for compact source and destination identification in HIP headers. For IPv4-based applications, HIP provides a Local Scope Identifier (LSI), which is essentially a private address that HIP translates locally into a HIT before transmission on the wire.

Figure 1 illustrates the base exchange process which is based on a 4-way handshake including I1, R1, I2 and

R2 messages between the initiator and responser. During the base exchange operation, the initiator and responder negotiate and agree on algorithms and keys for establishing end-to-end IPSec tunnels. The base exchange also authenticates the two end-hosts to each other because both end-hosts exchange their HIs during the key exchange and sign the packets using their private keys. In order to protect the responder from DDoS attacks, during the base exchange process, the responder sends a stateless, computational puzzle with a specific difficulty level to the initiator to solve in the beginning of the base exchange. After successful completion of the base exchange, the two end-hosts have to set-up a secure IPsec tunnel to encrypt the application data.

As it is depicted in Figure 1, when either the initiator or responder changes its location, it informs the other side of the communication about its new location by exchanging 3-way update messages. After the base exchange and possibly some update exchanges, the end-hosts can terminate the control and data plane using the closing procedure [24].

In SynAPTIC architecture, we employ HIP for Linux (HIPL) [3] implementation to integrate the HIP protocol in Docker containers. In the HIPL implementation architecture, a software component called HIP daemon is responsible of IPsec key and mobility management. In other words, it is responsible of the HIP control plane which set-ups IPsec tunnels between two end-hosts and updates the tunnel during end-host movement. The HIP daemon runs on each of the containers and, on the first boot-up, it generates private-public key pair, which represents the Host Identity (HI) of the container. To communicate over a HIP-based tunnel, applications have to use LSIs or HITs in their socket calls as local and remote addresses. It is worth noting that one end can utilize LSIs and the other HITs because the former is always locally translated into the latter. The daemon associates these two virtual addresses representing the HI of the local host with a virtual interface.

A HIP capable firewall [20] can be used to control and restrict communication between HIP-enabled containers. A migrating container can update its locators to the DNS, so that other containers can reach the migrated container. For rapidly moving hosts, extra immobile infrastructure, HIP

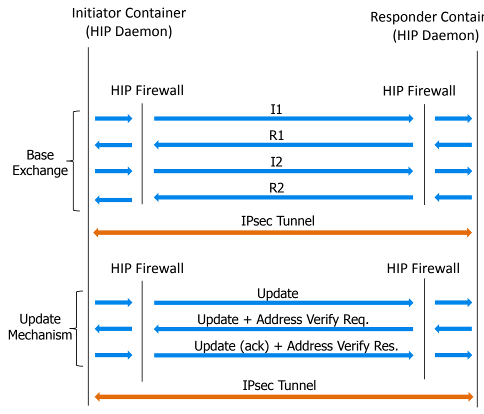


Figure 1: Flow diagram of HIP Base Exchange

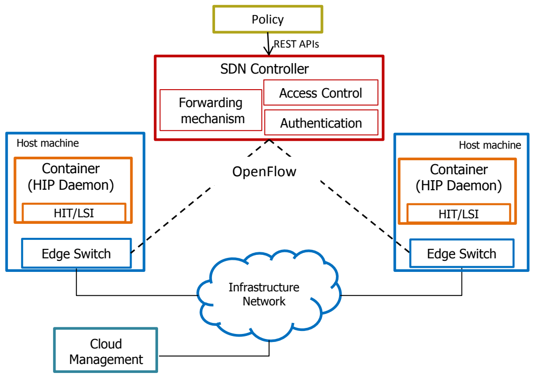


Figure 2: Container networking with SDN and HIP

rendezvous server [25], can be utilized. Also, it is possible to deploy HIP Relay [20] to connect containers behind NATted networks.

3.2. Supporting Container Connectivity with SDN

An increasing number of cloud providers are adopting SDN to provide higher level of control and flexibility in the infrastructure network. Therefore, in the remainder of this section, we extend our solution by proposing a SDN-based architecture for deploying SynAPTIC in cloud networks.

Authentication: An SDN controller is responsible for registering and authenticating new containers. While it is possible to verify containers by passively observing the signatures exchanged during the HIP base exchange, such a solution is vulnerable to replay attacks. We adopt the challenge-response approach from Heer et al. [19] where middle-boxes add a standard computational HIP puzzle into the base exchange messages in order to authenticate the end-points of the communication and to protect against replay attacks. However, instead of middle-boxes, we use the SDN controller for authenticating end-points from a centralized point by appending puzzles to the base exchange messages.

Access control: In our architecture, firewall and access control mechanisms are centralized with SDN. Unlike some of the existing solutions which mainly enforce policy based on network identifiers (i.e. IP address), access control in our solution is based on public keys (HIT values) which has an advantage of enforcing global policy on inter-container connectivity in a location-independent way.

Forwarding mechanism: The SDN controller forwards the traffic in the network by installing OpenFlow forwarding rules, and it reacts quickly in case of changes in the network topology, e.g., in case of relocation of containers.

Figure 2 illustrates the SDN-based architecture for deploying SynAPTIC. In this architecture, the controller must be able to control the (virtual) edge switches connected to the containers and the edge switches should support the OpenFlow protocol. When an application in a container initiates a HIP flow, the corresponding edge switch captures the base exchange packets and forwards them to the SDN controller (packet-in messages in the OpenFlow protocol). The SDN controller processes the HIP packets according to their type and instructs the edge switches to forward or drop the flow. Figure 3 shows the steps for handling a base exchange in the SDN controller which will be elaborated next.

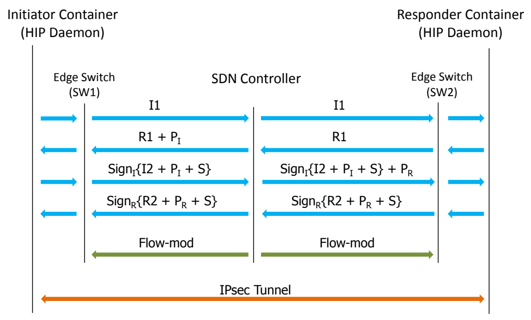


Figure 3: HIP base exchange through the SDN controller

I1 Packet: Upon receiving an I1 packet, the SDN controller verifies the source and destination HITs in the packet header based on the policies defined by the corresponding tenant. After policy verification, the controller stores the state of HIP tunnel and forwards the I1 packet to the responder.

R1 Packet: After receiving the R1 packet from the responder, the controller records the public key of the responder, appends the puzzle to the R1 packet and forwards the packet to the initiator. The puzzle is generated using one-way SHA-1 hash algorithm.

I2 Packet: The initiator solves the included puzzle using a brute-force process, appends the solution to the I2 packet, signs the packet with its own private key and sends the signed packet. The controller receives the I2 packet and verifies the identity of the initiator by checking the public key and signature. The controller also verifies the solution from the initiator. In case of successful verification, the controller adds a new puzzle to the I2 packet to authenticate the responder and forwards the I2 packet to the responder.

R2 Packet: The responder solves the puzzle from the controller, appends the solution to the R2 packet, and signs the packet and sends the packet to the initiator. The controller receives the packet and verifies the solution and signature of the responder and if they are valid, the controller forwards the R2 packet to the initiator and instructs the edge switches by installing OpenFlow rules to allow a direct IPSec tunnel between the initiator and responder.

Update Packets: If the controller receives HIP update messages for any container, it associates the update packets with the previous HIP base exchange and verifies the signatures. After the update procedure is successfully completed, the controller updates forwarding rules at the data plane to allow a direct IPSec tunnel from the new location.

Two connected containers may execute the HIP closing procedure in order to terminate an established HIP association. Since the controller cannot differentiate the encrypted IPSec packets and Close messages, we use idle_timeout and hard_timeout values in OpenFlow rules to remove inactive IPSec tunnels.

We have implemented a prototype on ONOS [15] which offers a distributed, multi-instance SDN controller. In our prototype, tenants are able to set policy through REST APIs and ONOS stores the policy in a shared data store from which all instances are able to independently match the received requests with the associated policy. Similarly, after verifying any new HIP base exchange, the information about the new HIP association is stored in a distributed store so that each ONOS instance has an updated view of all HIP associations in the network. The total lines of the prototype is about 3000 lines of JAVA code running on ONOS 1.5 (Falcon version), and it is based on OpenFlow 1.3 (but can also support other versions of OpenFlow).

4. Evaluation

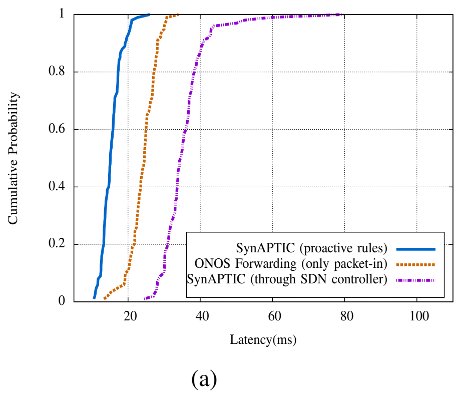
In this section, we evaluate the performance of SynAPTIC to establish inter-container connectivity and we compare it with Weave. We also test the functionality of SynAPTIC to preserve established sessions in case of container migration.

Our test network consists of two VirtualBox 4.3.36 [13] VMs running on a host machine with 16GB DDR2 RAM and 8 core CPU core i7@2.93GHz. Each VM has been allocated 4GB RAM and 2 CPU cores, and both VMs are directly connected through an internal network. Also, we run Docker 1.11 [2] on each VM with *NET_ADMIN* option enabled. Docker containers equipped with HIPL 1.0.8 trunk-6467 [3] and Netperf 1.11 [9]. Containers are able to connect to each other using Open vSwitch (OVS) 2.4.0 [10] or Weave 1.5 [14]. Furthermore, to evaluate the SDN-based approach, we run an ONOS instance on a separate machine with 8GB RAM and Core i7 Quad Core 2.80 GHz CPU. The controller machine can connect to OVS switches on host machine through a 1GB Ethernet switch.

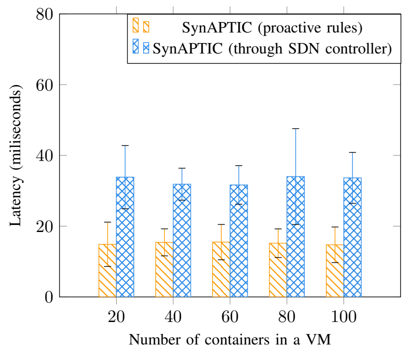
4.1. Latency

We measure the latency for establishing HIP tunnels and then we continue the evaluation in order to measure the network latency between two containers.

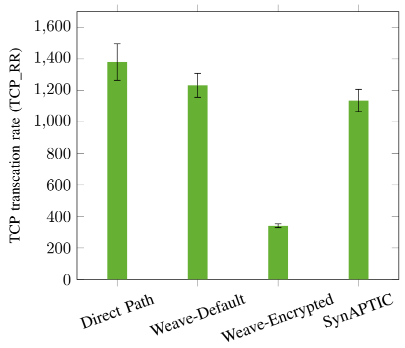
To begin with, we measure the latency only between two containers that are placed in separate VMs (one container in each VM). We consider three cases for comparing the latency of making HIP tunnels. Firstly, we connect containers through a direct path by installing proactive forwarding rules on OVS switches. In the next scenario, we connect the OVS switches to the ONOS instance and we run the default ONOS application for reactive forwarding. The ONOS application only receives OpenFlow packet-in messages from the switches and installs OpenFlow flow-mod rules for forwarding the data flows. Lastly, we run our SDN-based prototype on ONOS to verify all HIP base-exchange messages. We modify our prototype to allow connectivity between containers, and we also set the difficulty of the puzzles to the default value (difficulty 1).



(a)



(b)



(c)

Figure 4: (a) HIP base exchange latency (b) Base exchange latency in number of containers (c) TCP packets transaction rate

In each case, we measure the delay between sending I1 message and receiving R2 message in HIP base exchange. We repeat this process for 100 times to enhance the accuracy of the measurements. Figure 4a shows the resulting CDF graphs of our measurements. The latency to establish a HIP base-exchange through a direct path between containers is about 15 ms while it increases to 24 ms when forwarding HIP base exchange using default ONOS forwarding application. When we use SynAPTIC prototype to verify all HIP handshake messages, the latency rises to about 35 ms. Comparing with the default ONOS forwarding application, our prototype only adds 11 ms to verify all base exchange messages.

We continue our measurements to test the latency in large scale deployments. In one VM, we run a single container as a server and then in another VM, we increase the number of containers from 20 to 100 and we use a script to initiate the HIP base exchange from containers to the server. Figure 4b illustrates latency measurements with standard deviations for HIP base exchange process for different number of containers when the OVS switches are directly connected versus when the OVS switches are controlled by our ONOS prototype for SynAPTIC. The measurement results deviating 3 standard deviations from the average were considered outliers and omitted from the results. We measured the latency between I1 and R2 messages for all HIP connection requests received at the server. Based on the results, the measured latency is quite stable even with 100 containers in both cases.

Next, we measure and compare the latency of transmitting data flows through a direct path between containers using Weave networks with and without encryption versus using a HIP tunnel in SynAPTIC. For this purpose, we run one container in each VM and we use TCP_RR option in Netperf to measure the number of successful TCP request/response transactions between containers. We set the size of TCP packets to 1024 bytes and, we send and receive TCP packets in a period of 50 seconds. To enhance the accuracy of our measurement, we repeat the test for 100 times for each case. The bar chart in Figure 4c illustrates the average number of successful TCP transactions with standard deviations in a period of 50 seconds. As shown in the plot, SynAPTIC decreases the transaction rate for about 18% compared with the average transaction rate for the direct path between containers. Weave has a little better performance without encryption. However, when we enable encryption in Weave, the performance degrades considerably. One reason for poor encryption performance in Weave is because, by design, the encrypted packets are forwarded through the slow path between Weave routers.

4.2. Session Continuity

As we described earlier, SynAPTIC is able to provide connection sustainability based on the update mechanism in HIP. We run an experiment in the test network, where we connect two containers in different VMs using OVS switches and we use Netperf to stream TCP packets between the two containers. To test connection sustainability, we detach the container that initiates TCP stream (Netperf client) from the OVS switch and we connect it to a new switch port. Figure 5 shows the amount of traffic received at the other container (Netperf server). As we can see from the figure, the delay to resume the existing session between containers is less than a second. In our measurements, the actual latency for HIP update procedure is about 3.2 ms. However, the additional latency in Figure 5 occurs due to the switch port detach/attach delay and network configurations between containers.

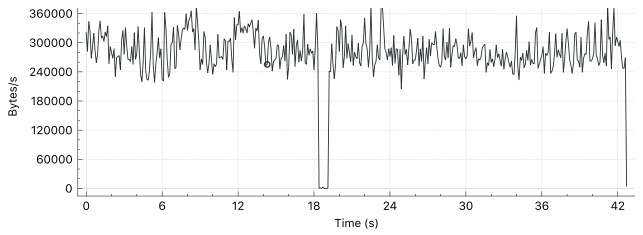


Figure 5: Testing session sustainability in SynAPTIC

5. Discussion

At the following, we discuss different aspects of SynAPTIC architecture.

Comparison with existing solutions: SynAPTIC assigns permanent and self-certifying identifiers (i.e. 128-bit HITs) to containers which is large enough to support a magnitude of containers. These identifiers can be used independently of the underlying cloud network topology. In comparison, Docker overlay and Weave employ tunneling techniques to support large number of containers and these solutions may not be effective to prevent IP address collisions (e.g. when two or more tenant networks are merged together in the same broadcast domain) and they may not be portable when switching between cloud providers.

SynAPTIC benefits from the strong security mechanisms in HIP that authenticate the containers with their public keys, encrypt application data flows using IPSec and provide countermeasures to protect against DoS and MitM attacks. Additionally, SynAPTIC enhances security by providing centralized access control and authentication, and protection against replay attacks based on SDN networks. In contrast, Docker recently adopted SSL/TLS protocol to enhance security, but it requires complex certificate and identity management. Weave can also supports encryption of the data flows. However, as we illustrated in our evaluation, Weave has poor performance with encryption. Also, encryption in Weave is based on simple passphrases shared between Weave routers which may not be secure method since the passphrases are shared and anyone with a passphrase can join the network.

Visibility vs Security: As SynAPTIC encrypts data packets, it may reduce the visibility of containers to cloud providers. However, we argue that, in container networking, this is not a major issue. Unlike with VMs, each container runs a certain service and thus, cloud providers should be aware of the expected type of traffic from containers. Therefore, even with encryption, providers can enforce access controls or QoS policies for certain type of services.

Lessons Learned: The deployment of SynAPTIC requires several considerations. There are still some features in Docker containers such as migration feature which are not fully supported. To achieve consistent view of the network in our prototype that is running on ONOS, our prototype needs to access the shared data store which is an expensive operation especially in large networks and may increase the latency of HIP base exchange. Since our solution stores security credentials (i.e. HIP private keys) on containers, one of the concerns is the trust between tenants and service provider networks which can be improved by deploying Virtualized Trust Platform Modules (vTPM) [16].

6. Conclusion

Virtualization based on Linux containers has become a popular way to deploy cloud services. Yet, the recent trends in cloud computing such as microservices and hybrid clouds impose new challenges particularly on container networking and connectivity. In this paper, we propose SynAPTIC architecture that provides secure and persistent connectivity between containers based on the standard HIP protocol. SynAPTIC can be deployed by the tenants in today's cloud networks, while the cloud providers can provide additional infrastructure support for tenants with software-defined networks. Our evaluation shows that SynAPTIC can outperform some of the existing solutions.

Acknowledgments

This work was supported by TEKES (the Finnish Funding Agency for Innovation) as part of the DIMECC Cyber Trust program.

References

[1] Calico Project. http://projectcalico.org.

[2] Docker Containers. http://docker.com.

[3] HIP for Linux. http://infrahip.hiit.fi.

[4] How to secure containers and microservices. www.infoworld.com/article/3029772/cloud-computing/how-to-secure-containers-and-microservices.html.

[5] IPSec connection between LXC containers. http://flockport.com/connect-lxc-containers-with-an-ipsec-vpn.

[6] Kubernetes. http://kubernetes.io.

[7] Linux Containers. http://linuxcontainers.org.

[8] Microservice architecture. http://microservices.io.

[9] Netperf. http://www.netperf.org/netperf.

[10] Open VSwitch. http://openvswitch.org.

[11] OpenContrail. http://opencontrail.org.

[12] Pertino. http://pertino.com.

[13] VirtualBox. https://www.virtualbox.org.

[14] Weave. http://weave.works.

[15] P. Berde et al. ONOS: Towards an open, distributed SDN OS. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, pages 1–6, New York, USA, 2014. ACM.

[16] S. Berger et al. vTPM: Virtualizing the trusted platform module. In *Proceedings of the 15th Conference on USENIX Security Symposium - Volume 15*, Berkeley, CA, USA, 2006. USENIX Association.

[17] D. Farinacci, V. Fuller, D. Meyer, and D. Lewis. The Locator/ID Separation Protocol (LISP). RFC 6830 (Experimental), Jan. 2013.

[18] B. Gleeson et al. A Framework for IP Based Virtual Private Networks. RFC 2764 (Informational), Feb. 2000.

[19] T. Heer, R. Hummen, M. Komu, S. Gotz, and K. Wehrle. End-host authentication and authorization for middleboxes based on a cryptographic namespace. In *2009 IEEE International Conference on Communications*, pages 1–6, June 2009.

[20] T. Henderson and A. Gurtov. The Host Identity Protocol (HIP) Experiment Report. RFC 6538 (Informational), Mar. 2012.

[21] K.-H. Kim et al. Flexible network address mapping for container-based clouds. In *Network Softwarization (NetSoft), 2015 1st IEEE Conference on*, pages 1–5, April 2015.

[22] T. Koponen et al. Network virtualization in multi-tenant datacenters. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 203–216, Apr. 2014.

[23] M. Mahalingam et al. Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks. RFC 7348 (Informational), Aug. 2014.

[24] R. Moskowitz et al. Host Identity Protocol Version 2 (HIPv2). RFC 7401 (Proposed Standard), Apr. 2015.

[25] R. Moskowitz and P. Nikander. Host Identity Protocol (HIP) Architecture. RFC 4423 (Informational), May 2006.

[26] Y. Nakagawa et al. Dynamic virtual network configuration between containers using physical switch functions for NFV infrastructure. In *IEEE Conference on NFV-SDN*, pages 156–162, Nov. 2015.

[27] D. Namiot and M. Sneps-Sneppe. On micro-services architecture. *International Journal of Open Information Technologies*, 2(9), 2014.