A Policy-Based Architecture for Container Migration in Software Defined Infrastructures

A Policy-Based Architecture for Container Migration in Software Defined Infrastructures

Xu Tao*
Politecnico di Torino, Italy
xu.tao@studenti.polito.it

Flavio Esposito
Saint Louis University, USA
flavio.esposito@slu.edu

Alessio Sacco
Politecnico di Torino, Italy
alessio_sacco@polito.it

Guido Marchetto
Politecnico di Torino, Italy
guido.marchetto@polito.it

***Abstract*—Software-Defined Networking (SDN) is a paradigm that enables easier network programmability based on separation between network control plane and data plane. Network Function Virtualization (NFV) is another recent technology that has enabled design, deploy, and management of softwarized networking services. The vast majority of SDN and NFV based architectures, whether they use Virtual machines (VMs) or Lightweight Virtual Machines (LVMs), are designed to program forwarding, probably the most fundamental among all network mechanisms.**

In this paper instead we demonstrated that there are other (as important) networking mechanisms that need programmability. In particular, we designed, implemented and extensively tested an architecture that enables policy-programmability of (live) migration of LVMs. Migration is used for maintenance, load balancing, or as a security mechanism in what is called Moving Target Defence (a virtual host migrates to hide from an attacker). Our architecture is based on Docker and it is implemented within a Software-Defined Infrastructure. Migration mechanism can be set easily by means of configuration file, to make a novel policy-based architecture. We evaluated the performance of our system in several scenarios, over a local Mininet-based testbed. We analyzed the tradeoff between several Load Balancing policies as well as several Moving Target Defense solutions inspired by network coding.

***Index Terms*—software defined networking, container migration, moving target defense.**

I. Introduction

The recent surge in popularity of Cloud Computing and Internet of Things (IoT) has resulted in a number of IoT networks, widely deployed. As new technologies showing up, today's network is much harder and more complex to manage and monitor. Thus, new network solutions come up. For instance, Software Defined Networking (SDN) is the latest network paradigm to solve the complexity of networking. It provides the benefits by detaching networking control layer and data layer, providing the possibility to use powerful central commands to meet the requirements of underlying demand data planes. Instead, Network Functions Virtualization (NFV) is a new method to design, deploy, and manage networking services. Virtual Machines (VMs) are widely used to implement NFV. Despite VMs, Lightweight Virtual Machines (LVM), such as Dockers, are a more efficient solution. The Docker technology allows the true independence between application, infrastructure, developers, and IT Ops. It enables creating a model for better collaboration and innovation.

This work was done in the Computer Science Department at Saint Louis University, USA.

Why a policy-based programmable migration mechanism is needed? Based on these new network solutions, migration is a new solution widely used in cloud network structure and data center. Migration is the movement of a virtual machine from one physical host to another, it happens without the awareness of end users. It can achieve networking maintainance, load balancing, network failure repair for providing an always available system. Apart from these, it can also be used as a security moving target defense strategy.

Nowadays migration solutions mostly focus on VMs [1], and Virtual Routers (VR) [2]. Besides, they are usually in ad-hoc environment, concerning a specific policy of the migration mechanisms; for instance, loading balancing [3] or energy optimization [4]. There is less concern on container migration. The container is known as the lightweight virtual machine. It does not virtualize only the hardware, but also the operating system. Comparing with the virtual machine, it is much lighter. If there is a high requirement with respect to the speed for migration, container migration could be a good solution.

Virtualization is a way that enables network programmability, and software defined networking is a good example. Above control plane, it is flexible and easy to develop applications such as routing, access control, etc. But it is only good for forwarding mechanism. In addition, network protocols are usually designed in the ad-hoc fashion. Different versions of TCP or routing exist, some of them are suitable for bandwidth sensitive applications, some are for delay sensitive applications, some aim to achieve security, some aim to provide better performance. There is no one-size-fits-all, a policy-based programmable migration mechanism is needed.

Our contribution. We designed a policy-programmable container migration architecture based on Docker. The policy-based architecture allows us to change policies with a simple configuration file, so programming the migration mechanism is easy. Second, we test security and load balancing policies within our SDN-based prototype over Mininet. Third, we designed and evaluated novel Moving Target Defense (MTD) solutions inspired by network coding.

The policy-based migration system can do software defined measurement based on the network traffic statistics obtained through SDN controller. We developed our algorithms to make migration decision and applied it on two use cases. The first is Load Balancing that we feature with 3 policies: bandwidth-based, shortest path, random. The second is Moving Target

Defense, where novel solutions are inspired by network coding, that we feature also with three policies: Shamir, Digital Fountain, and Pseudo Random function.

The paper is organized as follows. In section II, we discuss related migration solutions. Section III describes our migration system architecture. In section IV we present two use cases: load balancing, moving target defense. Section V illustrates the experimental validation results we obtained. In the end, the work is concluded in Section VI.

II. Related Work

Several network migration solutions exist nowadays, and a considerable work has been done concerning live VM migration [5]–[7]. In addition, there are a set of papers in which the authors compare and analyze the possible factors that could affect the migration performance. In [8]–[10], the authors examined the major issues of virtual machine live migration with some metrics, e.g downtime, total migration time, also classifying the techniques and comparing the different solutions. However, containers (lightweight virtual machine (LVMs)) are showing up as recent virtualization technique, they don't virtualize only the hardware infrastructure but also the operating system. Recently, new attempts to use containers instead of VMs have been proposed [11]. They focus on reducing migration time, with no concern about the network traffic situation. In our work, we concentrate on container migration, because compared to VM it is lighter and the migration can be faster than migrating a virtual machine. Our policy-based system performs migration adapting different application needs by just changing a configuration file. This is the first attempt, to the best of our knowledge, to build a architecture to enable programmable migration mechanism.

Moving Target Defense (MTD) is a new security paradigm. Instead of defending unchanging infrastructure by detecting, preventing, monitoring, tracking and remedying threats, moving target defense makes the attack surface dynamic. Many attempts have been proposed to achieve security through MTD. For instance, U-TRI adopts a randomly changing identifier to replace the original static data link layer address [12]. They defend traffic privacy by obfuscating the identifiers in network and transport layer. A different approach is used in WebMTD, that randomizes certain attributes of web elements to differentiate the application code from injected code and disallow its execution [13]. Besides, a more general solution is Mutated Policies [14]. It is an attribute-based defense strategy for access control that carefully selects the attributes that uniquely identify the entities involved. Then it randomly mutates the original access policies over time by adding additional policy rules constructed from the newly-identified attributes.

In our migration system, we move the container from one host to another one, to guarantee that the hosted machine IP address keeps changing. Then, we improve different algorithms existing with information of the network, integrating polynomial concept with a novel algorithm such as digital fountain mechanism.

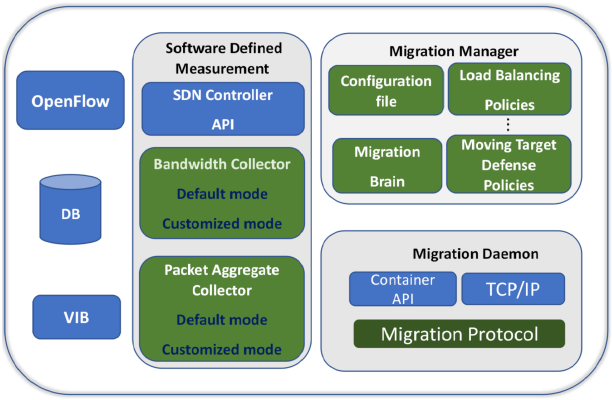

Fig. 1: System Architecture and Components, the green blocks are our contribution.

III. Architecture Design

In this section, we focus on the system architecture design and the function of each component. We built a programmable policy-based migration system, to provide flexibility for adapting different application needs by just changing one parameter in the configuration file. Besides, the system is designed to collect network traffic statistics leveraging an SDN controller, which applies software-defined measurements on the basis of these statistics. Thus, a more accurate migration decision is made by adding information about the network. As a consequence, a container can be migrated from a source host to a destination host within a cloud-edge network with a programmable policy-based mechanism. A well-designed migration system should be able to answer three questions: (i) which container should be migrated, (ii) when migration should happen, (iii) where to migrate. Following those questions, we designed our system architecture.

A. System Architecture Overview

Figure 1 shows the general architecture and key components of the system. There are 4 main component blocks designed for migration system: Database & Virtual Information Base (VIB), Software defined measurement, Migration Manager, and Migration Daemon.

(1) *Database & VIB* is designed to store the network traffic statistics. (2) *Software Defined Measurement* collects network traffic statistics through an SDN controller (Floodlight) and stores the data in an SQL-based database. We use two data collectors, one for bandwidth and one for packet aggregate. The *Bandwidth collector* is used to measure the bandwidth consumption per switch port. On the other hand, the *Aggregate collector* is used to get the number of packets per switch. Both measurements are collected at a customizable standard frequency. We use these measurements as input of our controller to detect traffic or switch overloads and start a migration process. We also use the information collected in the database to decide what is the destination for the migrating LVM, according to a programmable policy. Software defined

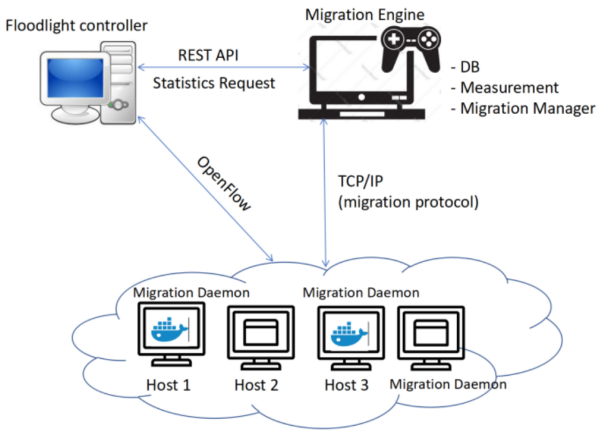

Fig. 2: System design and protocol used to exchange information. Migration Engine asks for network information and decide the migration parameters.

measurement system enables the simplicity and flexibility in collecting network traffic statistics. (3) *Migration Manager* monitors the process and makes migration decisions. In a configuration file, we specify a set of threshold parameters and the policy name. In our prototype we implemented two sets of policies for two use cases: Load Balancing and Moving Target Defense. Users can, however, easily implement their own policies. This component includes a *Migration Brain*, which executes the policy specified in the configuration file. (4) *Migration Daemon* is the process running on hosts, and handles the migration process. We use the Docker API to create, start, stop, take the memory and storage snapshot of the current container status. A schema of our prototype implementation is shown in 2.

B. Migration Model and Protocol

Migration manager makes the migration decision and communicates the destination host to the source host. When the source receives the command and the migration destination IP address, it starts the migration process. We defined a *Migration Protocol* used to execute such migration. First, Migration Manager makes migration decision and communicates it to the Source Host with a "MIGRATE" command. At this point, Migration Source Host takes the snapshots and stores the image files of the current running container (docker checkpoint). After that, it transfers the container image files to the Destination Host. During this communication, the source host does not stop providing the service. The communication between Source and Destination Host starts with a "RESTART" command sent by the source. This message is followed by the information about container image files. Once Destination has received all the required details, it restart the container. After the service starting, Destination sends "SUCCESS" command to the migration source host. Then, the TCP connection will be closed between all the parties involved. Also, source host stops the container providing the service. In the end, the routing is redirected to the migration destination host.

Our programmable migration framework enables to chose the destination host according to different criteria. In such a way an administrator is able to choose different policies for different use cases.

IV. Migration Policy Tradeoff and Use Cases

In this section, we explain our migration system on two use cases: Load Balancing and Moving Target Defense. The policies used in each use case will be listed and compared.

A. Use Case 1: Load Balancing

This application allows migration by monitoring the network traffic. The destination host is selected according to different criteria, and we focused on three policies to select the destination:

Random: destination host is selected at random.

Bandwidth-based: destination host is the host with the maximum available outgoing bandwidth. We define this value as the minimum link capacity of the links in the path.

Shortest Path: leveraging Floodlight controller we are able to get the network topology and compute the shortest path for each couple of nodes.

B. Use Case 2: Moving Target Defense

Moving Target Defense is a paradigm whose idea is to make the attack surface more dynamic. During the setup phase, (private) key(x,y) and a lookup table are distributed to each host. The lookup table is encrypted with a master secret for protecting the migration destination host. This table is an hash table associating to each index the destination host IP address. At the migration stage, our system provides to the source host a random number R to combine with the key as the input of a hash function:

$$hash(x) = Hash(R + X * Y)\%(N + 1), \tag{1}$$

where N is the number of hosts, R is the random number, (X, Y) is a key represented as a point and $\%$ is the modulo function . The value obtained from the hash function is the index of the lookup table. Here, three policies are used to share a secret:

Shamir: This policy is inspired by Shamir's method [15]: a secret is divided into K parts, and each participant has its own unique part. To get the secret key, a host needs to authenticate with some or all other hosts. The migration source host has to ask K disjoint hosts for K different keys to reconstruct the key and decrypt the lookup table. K is specified in the configuration file.

Digital Fountain: The migration source host needs to ask to K hosts for K keys, not necessarily disjoint. In our implementation we pick these K hosts probabilistically, using the following formula:

$$P(i,k) = \frac{\frac{1}{latency(i,k)}}{\sum_{\substack{j=1 \\ j \neq i}}^{n} \frac{1}{latency(i,j)}}, \tag{2}$$

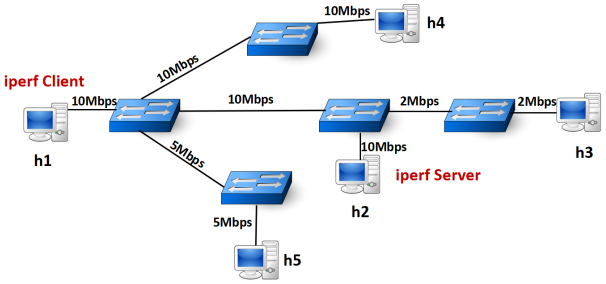

Fig. 3: Network topology with heterogeneous link capacity

where i is the source host, k is the random host, and $P(i,k)$ is the probability that host k is selected for asking the key. The host which has a smaller latency has a higher probability of being selected. This means that closer hosts may be contacted multiple times for the key.

Random: The destination host is selected by using a pseudo random function. We use this policy as a benchmark.

At the beginning, Migration Manager distributes different encrypted lookup table with the information required for the algorithms to each host. The manager generates also a set of key(x,y) for each host. Hence each host has a part of the information to decrypt the lookup table. Then, Migration manager sends a random number to the source host, it applies hash(x), and the result is the migration destination host index i. According to the policy specified in the configuration file, different strategy are used for decrypting the table. In case of *Digital Fountain* the same host can be contacted many times, since the one which has the shorter path will have the higher probability to be chosen. On the other hand, in *Shamir* the host asks to k disjoint hosts the key pair in order to decrypt the lookup table. After getting the k keys, the migration source host applies the algorithm (Digital Fountain or Shamir) to get the master secret S. Hence, the source host decrypt the lookup table using S, get the migration destination host IP, and start the migration process.

V. Experimental Validation

In this section, we test our system in a Mininet testbed, evaluating the two use cases and all the policies. The results are obtained using a Ubuntu Intel i7-6500U @ 2.50GHZ, 8.00 GB RAM, 64-bits.

A. Use Case1: 3 policies evaluation for Load Balancing

Scenario 1: Link capacity is heterogeneous. The topology we used as testbed is shown in Figure 3, where the link capacity varies among the links. H1 executes a docker container running iperf client, while H2 will be the source host and executes a docker container running the iperf server. The migration decision is different according to the chosen policy.

1) **Bandwidth-based policy:** H4 is selected as the destination host with a minimum bandwidth on its path of 10 Mbps.
2) **Shortest path policy:** H3 is selected as the destination host because of just 2 switches in between.

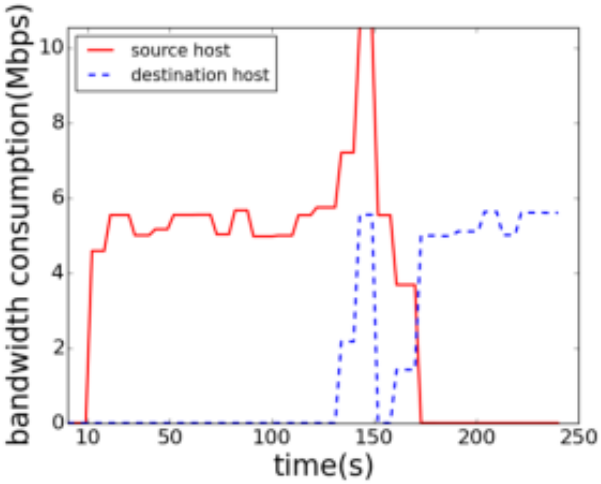

Fig. 4: The graphs display migration source and destination hosts bandwidth consumption collected for bandwidth-based policy.

	Bandwidth (s)	Shortest Path (s)	Random (s)
Heterogeneous	**9.1 ± 0.15**	40.1 ± 0.15	23.6 ± 7.12

TABLE I: Migration time of 3 policies in Load Balancing task for Scenario 1 on Mininet.

3) **Random Policy:** the destination host is randomly selected among the free hosts set: (H3, H4, H5).

Figure 4 shows the bandwidth consumption during the migration process. *Bandwidth consumption* value is the sum of sent and received bandwidth for the migration source and destination host. In the first period (up to 125s) the migration procession is not started yet, so on the source host (red line) the traffic is related to the docker container running the iperf server. After 125s the traffic on the switch is detected as too high and the migration process starts. During this period, the source host generates not only traffic data for the iperf client, but also the traffic data for the container migration. As a consequence, the destination host (blue line) starts to receive the migration files, so bandwidth consumption starts increasing. Then after migration process is done (150s), the source host (red line), does not run the iperf server anymore, so there is no more traffic. On the other hand, the destination host (blue line) starts to run the iperf server after migration.

In order to evaluate the time necessary for the migration process, we run 20 times the procedure for 3 the policies as shown in Table I. The time is the sum of time to make the decision and to make the migration.

Table I shows that bandwidth policy provides the best trade-off between network load balancing and the migration time. Bandwidth policy takes the advantage of the bigger link bandwidth, so the migration time is much smaller than shortest path policy, random policy. The confidence interval for bandwidth and shortest path is very small. This happens because the migration decision made for both use cases is determinate, H4 for bandwidth policy, H3 for shortest path. On

	Bandwidth (s)	Shortest Path (s)	Random (s)
Homogeneous	18.6 ± 0.91	**17.1 ± 0.16**	18.7 ± 0.91

TABLE II: Migration time of 3 policies in Load Balancing task for Scenario 2 on Mininet.

	Digital Fountain (s)	Shamir (s)	Random (s)
Homogeneous	36.6 ± 5.20	40.1 ± 6.60	**27.2 ± 3.70**

TABLE III: Migration time of 3 policies for Moving Target Defense.

the other hand, for Random, the migration destination is not determinate, so each run, it may choose different destination.

Scenario 2: Link capacity is homogeneous. In addition to topology with heterogeneous capacity, we tested the same topology where for all the link the capacity is homogeneous, and set to 5Mbps. In this context the decisions of three policies are as follows:

1) **Bandwidth-based policy:** the destination host is randomly selected among the free host set: (H3, H4, H5).
2) **Shortest path policy:** H3 is selected as the destination host because of just 2 switches in between.
3) **Random Policy** the destination host is randomly selected among the free host set: (H3, H4, H5).

In this case, the bandwidth policy has multiple choices, so the migration destination may vary every run. Table II highlights how in this case, shortest path performs better than bandwidth and random.

B. Use Case 2: Three policies evaluation for Moving Target Defense

In addition to the Load Balancing use case, we evaluated the cost of the system security by the application of Moving Target Defense. We tested the migration time for the three policies aforementioned. Looking at Table III, it is possible to observe how Random policy is the fastest one while for Shamir the migration time is the highest. This happens because in Shamir policy source host asks to k disjoint hosts for k different keys, hence far hosts can be selected. In Digital Fountain policy the source host asks to k non-disjoint hosts for k keys. It is likely to ask the host with small latency more times, leading to a smaller migration time.

In essence, random policy is the fastest one, but it does not apply any secure mechanisms, while Digital Fountain provides the better speed-security trade-off.

VI. Conclusion and Future Plan

In this paper we presented a policy-programmable container migration architecture based on Docker within an SDN prototype. It allows to change strategy and algorithm with a simple configuration file. Moreover, we tested two uses *i.e.*, Load Balancing and Moving Target Defense, and we applied three different policies for each use case. Based on the results obtained we found that in different scenarios different algorithms provide the best performance. Hence, our policy-programmable LVM migration system guarantees the appropriate flexibility, as such it can adapt to different application needs by just modifying the configuration.

As a plan for the future, we want to improve the system in several aspects. For the software defined measurement, we could integrate the SDN controller with big data and machine learning algorithms. In this case, the migration destination host can be predicted. By doing this we can improve the network management service. In addition, we could scale further the testbed and explore the policies trade-off in different topologies, such as tree, linear, star, fully connected.

Acknowledgments

This work has been partially supported by NSF CNS-1647084 and CNS-1836906.

References

[1] D. Kapil, E. S. Pilli, and R. C. Joshi, "Live virtual machine migration techniques: Survey and research challenges," in *Advance Computing Conference (IACC), 2013 IEEE 3rd International*. IEEE, 2013, pp. 963–969.

[2] Y. Wang, J. E. van der Merwe, and J. Rexford, "Vroom: Virtual routers on the move." in *HotNets*, 2007.

[3] P. Lu, A. Barbalace, R. Palmieri, and B. Ravindran, "Adaptive live migration to improve load balancing in virtual machine environment," in *European Conference on Parallel Processing*. Springer, 2013, pp. 116–125.

[4] I. S. Dhanoa and S. S. Khurmi, "Energy-efficient virtual machine live migration in cloud data centers," *International Journal of Computer Science and Technology (IJCST)*, vol. 5, no. 1, pp. 43–47, 2014.

[5] P. Kokkinos, D. Kalogeras, A. Levin, and E. Varvarigos, "Survey: Live migration and disaster recovery over long-distance networks," *ACM Computing Surveys (CSUR)*, vol. 49, no. 2, p. 26, 2016.

[6] S. Q. Zhang, P. Yasrebi, A. Tizghadam, H. Bannazadeh, and A. Leon-Garcia, "Fast network flow resumption for live virtual machine migration on sdn," in *2015 IEEE 23rd International Conference on Network Protocols (ICNP)*. IEEE, 2015, pp. 446–452.

[7] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation-Volume 2*. USENIX Association, 2005, pp. 273–286.

[8] D. Kapil, E. S. Pilli, and R. C. Joshi, "Live virtual machine migration techniques: Survey and research challenges," in *Advance Computing Conference (IACC), 2013 IEEE 3rd International*. IEEE, 2013, pp. 963–969.

[9] W. Voorsluys, J. Broberg, S. Venugopal, and R. Buyya, "Cost of virtual machine live migration in clouds: A performance evaluation," in *IEEE International Conference on Cloud Computing*. Springer, 2009, pp. 254–265.

[10] G. Soni and M. Kalra, "Comparative study of live virtual machine migration techniques in cloud," *International Journal of Computer Applications*, vol. 84, no. 14, 2013.

[11] "Criu for process ad container migration," http://criu.org/Main_Page, accessed: 2010-09-30.

[12] Y. Wang, Q. Chen, J. Yi, and J. Guo, "U-tri: Unlinkability through random identifier for sdn network," in *Proceedings of the 2017 Workshop on Moving Target Defense*. ACM, 2017, pp. 3–15.

[13] A. Niakanlahiji and J. H. Jafarian, "Webmtd: Defeating web code injection attacks using web element attribute mutation," in *Proceedings of the 2017 Workshop on Moving Target Defense*. ACM, 2017, pp. 17–26.

[14] C. E. Rubio-Medrano, J. Lamp, A. Doupé, Z. Zhao, and G.-J. Ahn, "Mutated policies: Towards proactive a ribute-based defenses for access control," 2017.

[15] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, Nov. 1979. [Online]. Available: http://doi.acm.org/10.1145/359168.359176