VMC: Managing Virtual Middleboxes in a Data Center environment

Nodir Kodirov
University of British Columbia

***Abstract*—Middleboxes are an integral part of the data center. A recent trend, network function virtualization, promises to replace hardware-based data center middleboxes with software-based, virtualized appliances. Although the latest development in middlebox virtualization, ClickOS, shows virtualized middleboxes to have a high-performance packet processing capability, there is no unified management system to enable virtual middleboxes' wide deployment to a data center environment.**

To address this issue, we introduce VMC, virtual middlebox controller. VMC creates, terminates, monitors and migrates a wide range of virtual middleboxes, including, but not limited to firewalls, NATs, and load-balancers. We implemented VMC on top of ClickOS and evaluated its ability to enforce a policy reactively. For example, a policy that provisions an additional middlebox when network load of a particular middlebox exceeds a predefined threshold can be reactively enforced via VMC. In a controlled environment VMC is able to quickly (under 100 ms) enforce such policies and automatically delegate extra network load to the newly created virtual middlebox.

I. Introduction

Virtualized middleboxes are making their way to the market. Not only to a data center environment where number of middleboxes are reported to be comparable to a traditional networking devices (bridge and routers) [16], but also to a large-scale network providers' communication critical exchange points, too [8]. These initiatives are mostly inspired by a recent advances in commercial middlebox virtualization solutions such as Cisco [9], Vyatta [17], and open source solutions such as ClickOS [12].

Figure 1 describes an inevitable evolution path, or what we think to be an incremental deployment of the virtual middleboxes in a data center environment. From the left, it starts with a current state, where middleboxes are mostly deployed as a standalone physical appliance with dedicated hardware and manual network configuration. Advances in middlebox virtualization starts off virtual middleboxes' deployment to a data center, while virtual middlebox combined with software-defined networking (SDN) gives it a real momentum, thanks to easy network configuration. Further development will (almost) fully eliminate dedicated hardware-based middleboxes, while integration with the cloud would be the last major milestone towards full integration of the virtual middleboxes to the data center environment.

In this work, we make a big step forward to overcome challenges of the last major milestone. The first challenge is inherent in diverse functionality middleboxes provide - whether is it possible to build a middlebox functionality-agnostic management system? The second challenge is how to detect a right time for middlebox orchestration so that middlebox *deployment criteria* is satisfied. Deployment criteria describes data center operator's objective during middlebox deployment. It can have several forms, such as load-balancing middlebox deployment, maximizing data center resource utilization, or maintain fast packet processing, among others. These goals do not necessarily overlap with each other and virtual middlebox controller should be able to provide a support for various deployment objectives.

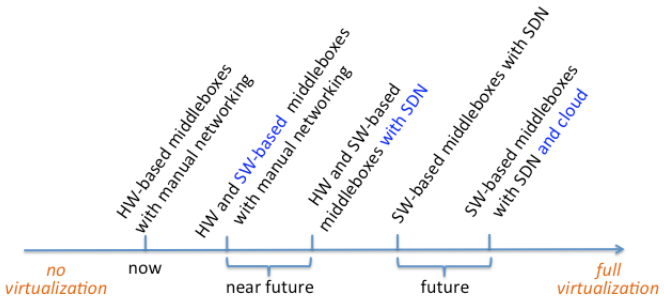


Fig. 1: Middlebox virtualization and integration scale.

To address these issues we introduce VMC, virtual middlebox controller. VMC uses ClickOS [12] as a virtualized middlebox building block and utilizes Libxenlight [4] API to control ClickOS instances. To further facilitate wide deployment of the virtualized middleboxes in a data center, we design VMC as a modular controller, which is able to interact with SDN and cloud controllers for easy networking and middlebox orchestration. Following section describes several motivational examples to illustrate VMC's management capabilities under different scenarios. Then we discuss VMC's implementation and evaluation. We conclude this report after summarizing related work and a huge body of future work.

II. Motivational scenarios

This section describes a number of cases where VMC could be very helpful to a data center operator. It is not an exhaustive list but merely highlights only few demonstrative cases.

A. CPU bound example

A NAT is configured to translate all network addresses from the internal subnet 10.1.0.0/16 to public address 203.1.2.11. Right now NAT box is serving 10K hosts and it is fully loaded. Then new policy is enforced to translate internal subnet 10.2.0.0/16 to public address 203.1.2.12. Enforcing additional policy into the same box will slow down its packet processing

speed, which will be detected by the system and new instance is spawn to take care of 10.2.0.0/16 subnet. This will ensure fast packet processing in both virtual middleboxes.

B. Memory bound example

A firewall is running with already established 1K rules. Enterprise network is experiencing DDoS attack by 100 different IP addresses, all belonging to Amazon EC2 West. There is a new policy to drop all incoming packets from those IPs before they hit servers. When administrator tries to enforce these policies to already running firewall, system will detect a memory saturation and provisions a new firewall. Additional 100 rules are pushed to a newly spawn instance. Then both firewalls will maintain a high packet processing rate.

FTP server can not handle large number of requests. WAN optimizer with 10 TB storage is deployed to cache .iso files being served to the clients. Increase in number of users diversify requested files and increase a cache-miss rate to 40% from original 10%. This is detected by the system and new optimizer is spawn with another 10 TB storage to serve additional requests. Now both middleboxes have $<$10% cache-miss rate.

C. Network I/O bound example

To reduce TCP connection overhead, a proxy is deployed between clients and a web server. It combines clients HTTP requests and sends to the server as one. Link between web server and proxy is already saturated. Increase in amount of client requests increases user experienced delay, since the link between middlebox and servers is 80% saturated. This overload is detected by the controller and a new proxy is instantiated to serve additional users.

III. VMC IMPLEMENTATION

We used ClickOS as virtual middlebox building block. It is open source, based on well-known Click library, has a small footprint (around 5 MB), and boots very quickly (around 30 ms) [12]. ClickOS does not separate between user and kernel execution space. Compiled image of the MiniOS and Click library are linked together to produce a final ClickOS image, which is runnable as Xen virtual machine (domU). When created, this domU starts at ”blocked” state waiting for the Click process to start. Click configuration is passed to the MiniOS via xenstore and xenstore can be used to manage ClickOS state [5]. Once Click configuration is written at predefined location of the xenstore, new thread is launched to read the configuration and execute respective Click process.

There are several challenges to make ClickOS work well with VMC. It has been reported [12] that ClickOS instances experience degraded performance as the number of rules increased. We think that this is a general problem for all virtualized middleboxes, since packet processing becomes a memory-bound rule-search operation. VMC monitors such performance degradation and provisions new middlebox to maintain high packet processing rate. Such monitoring module requires communication between VMC and each middlebox.

Generally, there are two options to establish such communication. The first is push-based, where virtual middlebox sends a frequent status update to the controller while controller continously listens to the incoming notifications. The other option is poll-based, where controller frequently requests status update from the virtual middleboxes. While both approaches work well for VMC, we opt in to use latter, poll-based approach since it makes our system fault-tolerant as well. The controller infers a middlebox has failed if it does not get response to the poll request within a reasonable time frame. Then, controller provisions a new middleboxes to take over the failed one.

In our current implementation VMC only supports basic bridge networking. It continuosly collects status information from each middlebox and enforces policies as instructed by an operator.

IV. EVALUATION

For our evaluation we chose middleboxes supported by ClickOS. Firewall and NAT are already supported by VMC, while support for a load-balancer is under active development. Here is a brief explanation with rationale behind each middlebox choice.

1) *Firewall*: sample Click configuration is provided in the tutorials, therefore easy to prototype. It is possible to start with a simple filtering rule and incrementally increase its complexity.
2) *NAT*: provides easy-to-understand functionality. ClickOS comes with carrier grade NAT and in [12] it is told to have a production-ready performance. By evaluating NAT in VMC we could perhaps test VMC's ability to support carrier-grade NAT, too.
3) *Load balancer*: does mostly stateful operation (except in a random flow assignment case) and therefore a good candidate to stress VMC's state management capability. Also, load-balancer has a close relationship with SDN-controller (since SDN-controller can also act as a load balancer via flow forwarding), therefore it is interesting to explore different combinations they could be deployed in a data center environment.

We evaluated VMC's ability to reactively enforce administrator's provisioning policy. In this case, VMC provisioned a virtual NAT between source and destination virtual machines as described in figure 2. Here NAT translates all internal network traffic originating from 10.0.1.0/24 subnet to a public IP subnet 198.162.52.0/24. We enforce a policy to provision a new NAT instance once outgoing traffic (Tx) of NAT reaches 10 MB. We keep increasing network traffic and when it reaches predefined threshold VMC creates one more NAT. Evaluation graphs are illustrated in figure 3, where initial NAT is shown as NAT1 and newly provisioned one as NAT2.

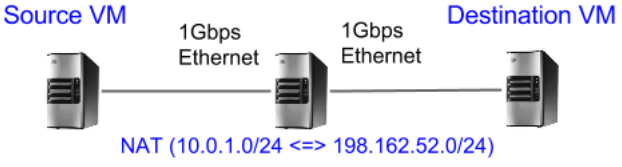
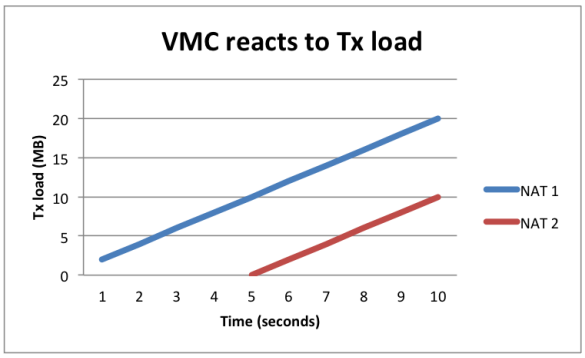
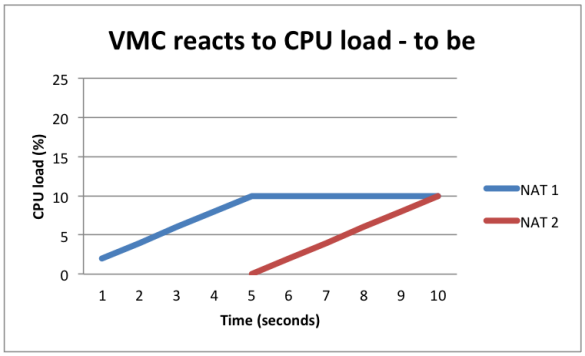


Fig. 2: NAT deployment.

Figure 3a shows current state of the VMC. There are two limitations of this figure. Firstly, we should not provision new middlebox when NAT Tx traffic exceeds predefined threshold, but when CPU, RAM or network load exceeds certain threshold (such as network has over 80% saturation). This was not possible to achieve (yet) due to a bug in ClickOS status reporting code. We could not get this issue resolved due to time constraints, but still actively working on it with ClickOS community [1]. We believe evaluating VMC's correctness does not heavily depend on the property being monitored, as long as VMC can reactively create a new virtual middlebox when certain threshold is reached, which is indeed achieved in VMC's current state. Once ClickOS bug gets fixed and it reports correct status information, we can quickly fix VMC logic by changing only few lines of the code.



(a) Current state of the VMC



(b) Expected behavior of VMC with Open vSwitch deployment

Fig. 3: Current and expected evaluation of the VMC with NAT.

Another limitation of figure 3a is system's state after provisioning the second NAT. As it is can be seen, Tx traffic of the NAT1 keeps increasing even after NAT2 instantiation. This is due to bridged network configuration, where network traffic from internal subnet is still broadcasted to both NATs. Solution to this problem is Open vSwitch (OVS)-based [3] deployment, where VMC automatically configures middlebox networking right after new middlebox instantiation. OVS network configuration is done by inserting appropriate rules to its forwarding table, such that extra traffic will be forwarded to NAT2 only.

In our initial VMC prototype used a bridge configuration due to its simplicity, but could not complete OVS deployment due to OpenStack networking bug. We spent several days to get OVS running with VMC and cloud controller. Our setup did not succeed, yet, and we are working with OpenStack community to get this technical issue resolved [2]. We conducted similar experiment with a firewall and VMC was able to seamlessly create an additional firewall when its Tx traffic exceeded a predefined threshold.

We believe above two limitations are not fundamental to the VMC. Given enough time and effort we can resolve technical problems and produce a graph similar to figure 3b. This figure shows VMC's reaction when NAT's CPU exceed predefined 10% threshold, which triggers instantiation of the second NAT and its automatic network configuration. Thus, extra network traffic will be forwarded to the second NAT and both NATs will experience CPU load under a predefined threshold.

V. Related Work

Closest to our work is the software-defined middlebox policy enforcement (SIMPLE) framework [13], where middlebox resource constraints are taken into account during policy assignment. SIMPLE enables network administrators to express middlebox policies in a generic language (called dataflow abstraction) and system takes care of enforcing rules into the physical topology. Central to this work is a load-balanced policy enforcement. Administrators have to express only "what" policy has to be enforced, the system decides "where" they have to be enforced. To do this, SIMPLE takes the policies as an input and collects several other information, such as network topology, middlebox locations, switch and middlebox resource constraints, and link capacity between switches to model the problem as a load-balanced optimization problem. This problem is solved via integer linear programming (ILP). Once a solution is found, the policies are expressed in a topology specific way and pushed to the middleboxes. While end result of our work would also contain load-balanced middleboxes deployment, VMC can deploy virtualized middleboxes with other objectives too, such as maximizing data center resource utilization, and maintaining fast packet processing rate. Moreover, SIMPLE uses middleboxes "as is", it does not dynamically scale in/out. VMC instantiates new middleboxes to handle dynamic load, while SIMPLE does not consider additional middlebox instantiation, it rather pushes policies to existing instances in a load-balanced way. Overall, SIMPLE's main contribution, ILP-based resource optimizer, is complementary to VMC.

Another work envisions a control plane for network middleboxes, called Slick [7]. Slick has two main objectives. One is to provide a support for policy enforcement in middleboxes running on heterogeneous platforms, such as CPU, GPUs, NetFPGA and etc. The other objective is optimal middlebox placement and steering of the matching traffic. Although not our primary target, our monitoring system can also be used in heterogeneous environment, as long as middleboxes are composed of Click elements. On addition to a traffic steering, we scale in/out middleboxes depending on the load, which Slick does not seem to handle.

A survey of 57 different sized data center enterprises hosting up to 100K servers show that the amount of middleboxes

deployed in the data center are comparable to a traditional L2/L3 appliances, hence cost and administration complexity [16]. These numbers are expected to see further growth once companies outsource middlebox processing into the dedicated cloud [10]. In an addition to easy administration, functionality update and reduced cost, middlebox cloud service providers are expected to guarantee a fast packet processing, too. Our work can aid providers with on-demand, functionality-agnostic middlebox control system.

A traditional hardware-based middlebox has a number of disadvantages, such as a vendor lock-in, slow update phase, lack of centralized administration and impossibility of computation reuse. Consolidated Middleboxes (CoMb) proposes a novel middlebox architecture with centralized controller on commodity (x86) hardware [6]. Consolidation overcomes most of the disadvantages mentioned above, and provides a centralized automated management with higher resource utilization. Controller reads network capacity, each CoMb box's capability and location, and policies to be enforced. Then, middlebox consolidation is modeled as an optimization problem which maximizes CoMb box's utilization in a load-balanced manner. Based on the solution, policies are then enforced to the middleboxes, which are run as a separate process in a CoMb box. While middlebox consolidation provides several advantages mentioned above, the authors do not show CoMb's applicability for fast packet processing. It also requires administrators to express expected load, which might not be known in advance. Since the CoMb framework does not have monitoring capabilities, it can not dynamically scale in/out depending on the load. Monitoring and on-demand provisioning are the main difference of our work.

Several works [15], [6] contributed design of modular, extensible middlebox architectures which are easy to build and manage. ClickOS is inspired by Click modular router. It builds middlebox from composition of the Click elements [11]. ClickOS is a Xen virtual machine instance with small memory footprint (about 5 MB) and fast packet processing capability (at wire speed, about 10 Gbps). Our work uses ClickOS as virtual middlebox building block.

The Split/Merge system [14] proposes on-demand scale in/out of middleboxes based on its load. Their main contribution is to carefully manage state associated with middlebox and flows going through it. To evaluate the system's state management capabilities, the authors enforce a policy to split middlebox into two (or merge it by moving flows into another middlebox) when the number of flows exceed (or drops below) an arbitrary threshold. In our work VMC uses Split/Merge as one of features to support middlebox management.

VI. Future Work

Our immediate future work is to resolve aforementioned bugs in ClickOS and OpenStack controller. They will enable us to experiment with wide range of additional middleboxes and further verify VMC's ability to achieve functionality-agnostic management.

Next, there are several other directions we plan to extend VMC. One direction is virtual middlebox state management, similar to Split/Merge [14]. Middlebox state management will further widen VMC applicability to another class of so called *stateful middleboxes*. For example, when middlebox performance drops below a threshold value imposed by administrator, controller provisions an additional ClickOS instance, copies over relevant middlebox state, enforces new policies and re-routes all incoming network traffic to the newly created middlebox. Conversely, VMC can merge two underutilized middleboxes if one is capable of taking over the other (e.g., when middleboxes have the same type, deployed along the same path and etc.)

We can also develop several add-on features for VMC's closer integration to a data center environment. For example, ability to interact with cluster-wide SDN controller will allow VMC manage virtual middleboxes across different clusters. Integration with cloud controller will be the last major milestone for virtualized middlebox management (see fig. 1). Cloud's image service helps us eliminate VMC's manual image transmission during each virtual middlebox instantiation and further reduce VMC's overhead. Cloud's compute and network controllers already collect hypervisor status information, network topology, link bandwidth and utilization information. VMC could take advantage of already collected data and make administrators job easy by showing resources available for virtual machines and virtual middleboxes.

Another forward-looking work is to deploy multiple middlebox controllers for fault-tolerance, such that VMC does not become single point of failure in middlebox deployment. We could also partition data center resources to different clusters and have separate instance of VMC controlling each cluster. All of these enhancements target VMC's closer integration with existing technologies and facilitate virtual middlebox deployment in data center environment.

VII. Conclusion

Middleboxes provide important and wide range of services to a data center environment. Transition to a software-based virtualized middleboxes is necessary to overcome vendor lock-in, fixed point deployment, manual configuration and operation of the hardware-based middleboxes. Controller with adequate virtual middlebox management support will facilitate virtual middleboxes' large-scale deployment and integration with existing data center technologies, such as cloud and SDN controllers. In this work we presented VMC, virtual middlebox controller to provide such management support. We designed and implemented VMC on top of ClickOS, minimalistic operating system able to run Click library. Our experiments with two middleboxes (third one ongoing) confirm VMC's ability to provide unified middlebox monitoring and orchestration in a middlebox functionality-agnostic way. VMC's monitoring capability enables us to continuously monitor hypervisor and virtual middlebox resources, and dynamically scale in/out depending on the current workload. Our initial prototype of VMC and preliminary evaluations show feasibility of such controller and we plan to develop VMC further to achieve its closer integration with SDN and cloud controllers.

REFERENCES

[1] Clickos mailing list bug report. https://listserv.netlab.nec.de/mailman/private/cnplab/2014-September/000015.html, 2014. [Online; accessed 15-Sep-2014].

[2] Openstack networking bug report. https://bugs.launchpad.net/neutron/+bug/1303998, 2014. [Online; accessed 15-Sep-2014].

[3] Production quality, multilayer open virtual switch. http://openvswitch.org/, 2014. [Online; accessed 15-Sep-2014].

[4] Xen manuals, choice of toolstacks: libxl. http://wiki.xen.org/wiki/Choice_of_Toolstacks#Libxenlight_.28libxl.29, 2014. [Online; accessed 12-Sep-2014].

[5] Xen manuals, xenstore. http://wiki.xen.org/wiki/XenStore, 2014. [Online; accessed 12-Sep-2014].

[6] J. W. Anderson, R. Braud, R. Kapoor, G. Porter, and A. Vahdat. xomb: Extensible open middleboxes with commodity servers. In *Proceedings of the Eighth ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ANCS '12, pages 49–60, New York, NY, USA, 2012. ACM.

[7] B. Anwer, T. Benson, N. Feamster, D. Levin, and J. Rexford. A slick control plane for network middleboxes. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pages 147–148, New York, NY, USA, 2013. ACM.

[8] AT&T. At&t domain 2.0 vision white paper. http://www.att.com/Common/about_us/pdf/AT&T%20Domain%202.0%20Vision%20White%20Paper.pdf, 2013. [Online; accessed 12-Sep-2014].

[9] Cisco. Cisco nexus 1000v switch for vmware vsphere: Data sheets. http://www.cisco.com/c/en/us/products/switches/nexus-1000v-switch-vmware-vsphere/literature.html, 2014. [Online; accessed 12-Sep-2014].

[10] G. Gibb, H. Zeng, and N. McKeown. Outsourcing network functionality. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN '12, pages 73–78, New York, NY, USA, 2012. ACM.

[11] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The click modular router. *ACM Trans. Comput. Syst.*, 18(3):263–297, Aug. 2000.

[12] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, M. Honda, R. Bifulco, and F. Huici. Clickos and the art of network function virtualization. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 459–473, Seattle, WA, Apr. 2014. USENIX Association.

[13] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu. Simplefying middlebox policy enforcement using sdn. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, pages 27–38, New York, NY, USA, 2013. ACM.

[14] S. Rajagopalan, D. Williams, H. Jamjoom, and A. Warfield. Split/merge: System support for elastic execution in virtual middleboxes. In *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, nsdi'13, pages 227–240, Berkeley, CA, USA, 2013. USENIX Association.

[15] V. Sekar, N. Egi, S. Ratnasamy, M. K. Reiter, and G. Shi. Design and implementation of a consolidated middlebox architecture. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, pages 24–24, Berkeley, CA, USA, 2012. USENIX Association.

[16] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar. Making middleboxes someone else's problem: Network processing as a cloud service. In *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '12, pages 13–24, New York, NY, USA, 2012. ACM.

[17] Vyatta. The open source networking community. http://www.brocade.com/launch/vyatta/, 2014. [Online; accessed 12-Sep-2014].