Real-Time Network Function Virtualization with Timing Interfaces

Linh Thi Xuan Phan
University of Pennsylvania
linhphan@cis.upenn.edu

ABSTRACT

More and more infrastructure is becoming virtualized. Recently this trend has begun to include network functions – such as firewalls, WAN optimizers, and intrusion prevention systems – that have traditionally been implemented as middleboxes using dedicated hardware. This trend towards network function virtualization (NFV) offers a variety of potential benefits that resemble those of cloud computing, including consolidation, easier management, higher efficiency, and better scalability. However, current cloud technology is not a perfect match for NFV workloads: since the infrastructure is shared, the time it takes for a packet to pass through a particular function is no longer predictable, and can in fact vary considerably. This is causing headaches for operators, who can no longer treat network functions as "bumps in the wire" and must now consider a complex web of possible interactions and cross-talk when operating or diagnosing their systems.

In this position paper, we propose a compositional approach towards building a scalable NFV platform that can provide latency and throughput guarantees using timing interfaces. We discuss our preliminary results that leverage and extend recent advances on timing interfaces and compositional theory from the real-time systems domain to the NFV setting, and we highlight open challenges and potential directions towards real-time NFV.

1. INTRODUCTION

Modern network functions no longer restrict themselves to forwarding packets; they also perform a variety of other functions, such as firewalling, intrusion detection, proxying, load balancing, network address translation, or WAN optimization. Traditionally, these functions have been implemented as middleboxes on dedicated hardware. But increasingly this infrastructure is being virtualized, and the physical middleboxes are being replaced by virtual machines that run on a shared cloud infrastructure [24]. This trend towards *network function virtualization (NFV)* offers a variety of potential benefits that resemble those of cloud computing – including consolidation, easier management, higher efficiency, and better scalability.

Ideally, the virtualized network functions would offer the same properties as the middleboxes they have replaced. In particular, they would offer low, predictable latencies and guaranteed throughput. These properties are necessary for the network functions to remain transparent to the rest of the network: they are expected to behave as "bumps in the wire" that do not have any effect on the traffic that passes through them (other than the effects they were designed for). If network functions were allowed to interact with each other or to introduce bottlenecks, jitter, or latency variations of their own, this would create a massive headache for the network operators, who would face an exponential increase in the number of possible failure modes.

However, current virtualization technology can only support these properties to a very limited extent. The reasons are partly historical: most existing virtualization platforms were developed for cloud *computing*, where some latency and throughput variations can often be tolerated. Of course, there are scenarios in which such variations have problematic consequences even for cloud workloads – such as performance "cross-talk" between VMs [4, 10, 26] – and a number of countermeasures have been developed over the years, ranging from careful resource allocation [12, 17, 18, 26] to profiling and reactive reconfiguration [20, 28, 29]. However, most existing solutions take a best-effort approach and can correct only relatively large performance variations – far above the latency and jitter that packets typically experience in a network. A truly transparent NFV platform would require a far more fine-grained, proactive approach.

In principle, this problem could be avoided through careful resource management, e.g., using classical real-time scheduling techniques. However, classical real-time technology is designed for workloads that are very different from virtual network functions. For instance, the real-time literature often assumes that all the tasks run on a single machine, have few interdependencies, and are relatively static. In contrast, a shared NFV infrastructure would have a much larger number of tasks that would typically span several machines and might adapt at runtime to changes in the traffic.

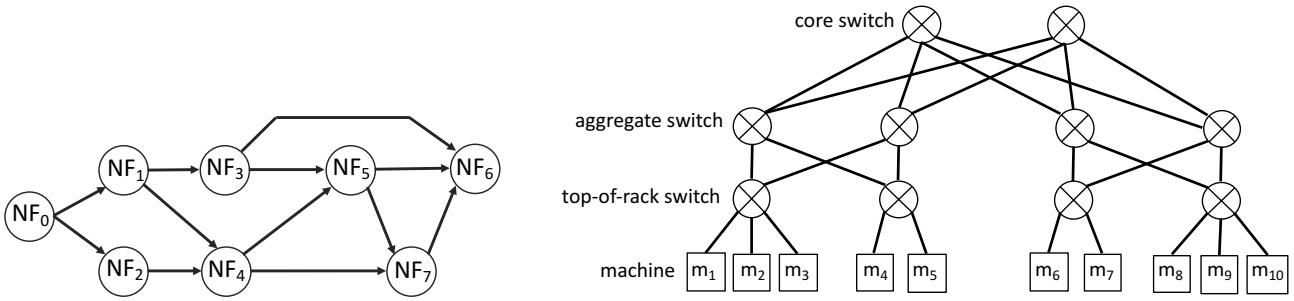
Fortunately, there are two recent developments in the real-time systems domain that have brought a solution within reach: 1) *Compositional scheduling* algorithms [2, 15, 21, 23, 25, 27], which scale much better than classical real-time schedulers and can handle the large workloads a shared NFV platform would likely encounter, and 2) *multi-mode scheduling* techniques [7, 21, 22], which enable systems to adapt at runtime without losing their real-time guarantees.

In this paper, we propose to leverage these techniques to construct a *scalable NFV platform that can provide latency and throughput guarantees*. We first present the system model and an overview of our approach. We then discuss the key challenges and highlight potential directions towards the envisioned platform. We conclude the paper with preliminary results that demonstrate the feasibility and potential benefits of our approach.

2. SYSTEM MODEL AND GOALS

We consider a cloud setting in which the provider offers NFV services to a wide variety of tenants, with each executing one or more NFV applications on behalf of its customers.

Platform model. The cloud platform is made of multiple racks of machines that are connected via switches that form a fat-tree network topology [13], as shown in Figure 3(b). We model the plat-



(a) NFV application model (b) Cloud platform model

Figure 1: System model.

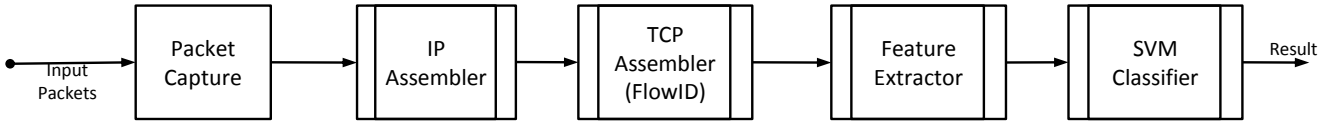


Figure 2: An NFV application that implements DDoS detection using machine learning.

form as a multi-rooted tree, where each node v represents a machine or a switch, and each edge (v, v') represents the network link that connects v and v', which is annotated with the link's total bandwidth. Each physical machine consists of a number of unit-speed cores that share a last-level cache and the main memory. Each machine runs a virtual machine monitor (VMM), such as Xen [3] or KVM [1], and it hosts multiple virtual machines (VMs) within which the network functions are executed.

Application model and tenants. An NFV application is modeled as a directed acyclic graph (DAG), whose nodes represent network functions (NFs). Each NF is modeled by the per-packet worst-case execution time (WCET), and the maximum required amounts of shared cache, memory, and storage with respect to the given WCET. Each edge from NF_i to NF_j is associated with a data size ratio r_{ij} and a selectivity s_{ij}: the former is the ratio of the input packet size of NF_j to that of NF_i, and the latter is the maximum number of output packets produced by an input packet of NF_i that are input packets to NF_j.

Each NFV application has an end-to-end relative deadline, i.e., the maximum end-to-end latency that any incoming packet can tolerate through the application, as well as a minimum throughput requirement. Each incoming packet of the application will be processed along one path in the DAG; the exact path depends on the packet's functional characteristics, which typically vary across packets.

Each tenant runs one or more NFV applications on the cloud, serving traffic on behalf of its customers. The request of a customer is modeled by a tuple that contains the ID of the NFV application it wants to use, a maximum packet rate of its (incoming) traffic. The request's SLA is specified in terms of meeting the packet-level NFV application end-to-end deadline and throughput requirements.

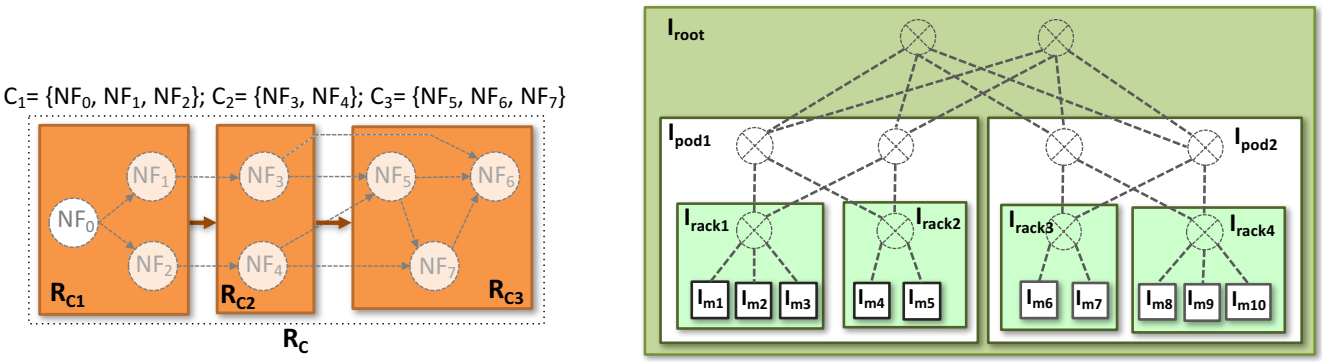
Example. Figure 2 shows an NFV application with a chain of NFs that implements Distributed Denial of Service (DDoS) attack detection [19] using machine learning. This application receives packets from the network (Packet Capture) and assembles packets into complete IP packets and TCP segments (IP Assembler and TCP Assembler). The Feature Extractor function extracts key features (e.g., connection duration, bytes/packets transferred, source/destination port number and addresses) for each assembled flow, which are then used by the SVM Classifier function to detect flows that exhibit DoS behavior. The SVM Classifier is first trained using existing traces, a subset of which are tagged as malicious, in order to learn which TCP flow features correlate with malice.

Objectives. Given the above setting, our goal is to develop a resource management and scheduling system for the cloud platform to maximize the number of requests that meet their SLAs at run time, while guaranteeing performance isolation among tenants. Towards this, our specific objectives are to develop (i) an NF assignment algorithm that assigns NFs to virtual machines (VMs) and to machines, (ii) a scheduling policy for the VMs and for the NFs within a VM on each node, and (iii) a routing strategy of the traffic in the network. The system should consider both the initial resource allocation and scheduling for an initial set of requests, as well as when new requests arrive and existing requests leave the system.

3. APPROACH: RESOURCE ALLOCATION VIA TIMING INTERFACES

To enable scalable resource allocation, we propose to use a *compositional* approach to the analysis and scheduling of the NFV applications. Our key insight is that, despite their complexity, both the cloud and NFV applications have well-defined structures that are amenable to component-based modeling and compositional reasoning. Specifically, we can group related NFs of an application into a component (to be executed within a VM) and describe their total resource needs by a – much simpler – *interface*. Intuitively, the interface specifies the amounts of resources (including CPU, memory, network bandwidth, etc.) that are needed for the component to meet its SLA constraints. Similarly, the hierarchical structure of the cloud enables us to decompose the platform into several pods, with each pod consisting of several racks, and each rack consisting of multiple machines that each consist of multiple cores. We can then describe the current available resources of each such component (machine, rack, pod, entire cloud) using an interface that captures its aggregated resource supply.

Resource allocation and scheduling can be done in a hierarchical manner based on these interfaces: at the top level, the resource manager performs admission control of new NFV requests based on



(a) NFV application abstraction

(b) Cloud platform resource abstraction

Figure 3: Interfaces of the NFV and platform components.

their applications' interfaces and the highest-level interface of the platform, and it distributes the NFV components to the pods based on their interfaces; the same then happens at each of the lower levels of the platform, until each NFV component is assigned to a VM on a machine. With this approach, performance isolation and SLAs can be guaranteed simply by ensuring that requests of different tenants are processed by different VMs, and that the resources requested by the interfaces of the NFV components can be satisfied by the interfaces of the platform components to which they are assigned.

Once an assignment is completed, the VMM at each machine can simply allocate resources to the VMs based on the interfaces of the NFV components they execute. Further, the feasibility of the resource distribution also guarantees a feasible routing solution for the requests' traffic through the machines that execute the corresponding NFV components. The interfaces of the platform components are recomputed after each successful assignment of new requests and whenever an existing request leaves the system, to reflect the current available resources. This computation can be done efficiently based on the current platform interfaces and the interfaces of the NFV applications of the requests.

Figure 3 illustrates our approach for the example shown in Figure 1. Here, the NFV application is decomposed into three components, $C = \{C_1, C_2, C_3\}$, with each C_i being encapsulated in an interface R_{Ci}. The entire application is encapsulated in an interface R_C, which is the composition of all R_{Ci}. The platform is abstracted as a hierarchical of interfaces, as shown in the figure. Admission control of a request of the application is performed at the root level, based on the top-level interface $\mathsf{I}_{\mathsf{root}}$ and the application's interface R_C. If it admits the request, the resource manager then assigns the corresponding instance(s) of the application to one of the pods, or distributes its components, C_i, between the two pods based on R_{Ci} (if neither $\mathsf{I}_{\mathsf{pod1}}$ nor $\mathsf{I}_{\mathsf{pod2}}$ can satisfy R_C). The assignment at the next level can be done similarly, until finally each component C_i is assigned to a VM on some machine. Once the assignment is completed, the VMM of the assigned machines can simply allocate resources to C_i's VM based on R_{Ci}.

4. CHALLENGES AND DIRECTIONS

There are several challenges to fully realizing our proposed compositional approach. In this section, we discuss some of the key challenges and potential directions towards addressing them.

4.1 Component modeling

Existing compositional theories do not address the component modeling for applications – instead, they always assume that the set of components and their composition are given a priori. Finding a good decomposition strategy for an NFV application is highly non-trivial, as different partitions of the DAG of NFs can result in very different scheduling decisions and resource demand patterns. In theory, one could simply view the entire application as a single basic component, but this approach does not always work because the application itself may not fit within a VM (or even a node). In contrast, simply viewing each NF as a basic component can lead to too many VMs and thus high overheads.

To facilitate compositional reasoning, besides the structural partitioning of the application into components, one also needs to decompose the end-to-end specifications (e.g., incoming arrival rates, end-to-end SLA requirements) into component-level specifications. We expect that existing deadline decomposition methods from the real-time scheduling literature can potentially be applied. However, they would need to be substantially enhanced to work for our setting, especially because – unlike in existing real-time work – the exact routing of the traffic through the NFs (when they span multiple nodes) is not known in advance but, instead, is driven by the decomposition itself.

4.2 Interface models and interface analysis

Existing timing interfaces are limited to independent tasks, and they consider only one type of resources (such as CPU, memory, or network) but not their combinations. While these assumptions are reasonable for many real-time systems, they do not hold for NFV applications: the NFs of an application require a much more diverse set of resources (e.g., CPU, memory, bandwidth), and they are highly dependent on each other, not only in the input-output data dependencies but also in the arrival rates and data sizes.

One direction is to adopt a multi-dimensional interface, with each dimension capturing the demand and supply of one resource type. At first, this seems like a straightforward extension of the existing interface models; however, since the different types of resources are intertwined, a component's demand for one type of resource (e.g., CPU) is dependent on how much resource of another type (e.g., memory) that is available to the component. This interdependence makes interface analysis highly challenging, but it too provides room for resource optimization based on various tradeoffs among the resource types.

Ideally, we would like to expose this interdependence on the interface to enable optimization; unfortunately, it is often difficult to characterize such a relationship in a closed form. To enable flexibility and online refinement, it would still be useful to consider an

approximation of this interdependence, e.g., using a multi-mode multidimensional interface, where each mode captures an optimization for a particular run-time operating condition. For example, the interface could minimize the memory demand in a mode where memory is a potential bottleneck, and it could minimize network usage in another mode where network is a potential bottleneck.

Another challenge lies in the modeling of input and output flows of a component. Here, we could exploit the property that each packet typically traverses only one path in the NFV graph, to simplify the interface. Instead of exposing all possible flows of a component, we could group the ones with similar arrival rates and data sizes into a single abstract flow that is characterized by the maximum arrival rate and data size per packet of such flows.

4.3 Scalable resource allocation

The interfaces of the NFV applications and the platform's available resources can be used to check for *feasibility* and *resource reservation*. Specifically, a component of an application can be feasibly assigned to a platform component if the former's interface can be guaranteed by the latter's interface. Further, once each NFV component is assigned to a node, the node's VMM can simply allocate to the VM that execute an NFV component the exact amounts of resources specified by the component's interface.

To arrive at an efficient resource allocation, the system must also be able to efficiently find a good assignment that optimizes resources at each level based on the feasibility conditions. This is not a one-time process: it must be able to add and remove the NFV components at runtime as new requests arrive at the system or existing customers leave, without disrupting too much the existing allocation while still achieving efficient resource utilizations. If the system were to generate an entirely new allocation for both the new and existing applications in such cases, many NFV components would likely end up on a different machine and would need to be moved, which would result in substantial overhead.

Dynamic bin-packing algorithms [8, 9] and efficient approximation algorithms for multidimensional bin-packing [5, 6, 14] should provide good starting points for this. Where multiple feasible assignments exist, one can optimize for other objectives, e.g., by balancing the load across servers, or by concentrating the workload on a subset of the servers so that the remaining ones can be powered down. The multiple modes of the interfaces also provide venues for optimization, e.g., by choosing the interface modes from several components that "fit together" the best, and thus yield the most efficient resource uses. For incremental resource allocation, it seems useful to develop an "interface decomposition" operator as the inverse of composition to enable efficient computation of the available resource interfaces of the platform components when an NFV component is removed from a platform component.

4.4 Accounting for virtualization overheads

The presence of virtualization adds two important complications to interface analysis. The first is that virtualization introduces several types of extra overheads; if these are not considered, the interfaces can underestimate the components' resource needs, and thus the components can violate their timing constraints even when their interfaces are satisfied [27]. The second challenge is that virtualization can cause unexpected interference, even between NFs in different VMs. For instance, two components in different VMs may be scheduled on cores in the same socket that share a L3 cache; thus, a memory-intensive service in one component can slow down the services in the other component. The magnitude of this effect can vary with the values in the interfaces as well as with the implementation of the interfaces by the VMM.

4.5 Time-aware traffic management

To achieve end-to-end timing guarantees, carefully scheduling the VMs on the individual hosts is not enough: we must also ensure that traffic is not delayed arbitrarily in the network. In principle, it is well known how this can be done – e.g., using circuit switching – but the network hardware that is commonly deployed today does not support this very well. The emergence of software-defined networking (SDN) provides a way out, namely by implementing a timing-aware traffic management scheme on the SDN controller. In our prior work, we have already explored OpenFlow-based protocols that implement path selection and performs dynamic rate reservation based on flow sizes and flow deadlines, as well as RTT estimates. We expect that implementation of such protocols should be feasible for the NFV setting as well.

5. PRELIMINARY RESULTS

We have done some exploratory work to verify the feasibility of the proposed approach. (Part of the results presented here has appeared in [16].) We consider the same cloud setting with a fat-tree topology as described in Section 2. Each machine runs RT-Xen, a real-time extension of Xen VMM that schedules VMs based on their assigned VCPUs' periods and budgets. For simplicity, we restrict each VM to use only a single VCPU. The component modeling ensures that each basic component does not require more than one core (i.e., their maximum utilization is at most 1). To enable efficient scheduling, we use partitioned EDF at both the VM and the VMM levels.

We study NFV applications with the general DAG topology, with each edge connecting two NFs having a different data size ratio, but the same selectivity of 1. We focus on only CPU and network resources, and the SLA is specified in terms of end-to-end deadlines. We assume that the timing specifications (e.g., WCETs) of the applications are never violated, and the scheduling and virtualization overheads are negligible.

Component modeling and interface analysis. As a first step, we focus on the abstraction of the NFV applications and use a simple approach for capturing the cloud resources. The interface of a cloud component is simply a fat-tree representing its network topology, where each edge is associated with the current available bandwidth of the associated network link, and each leaf node is associated with a vector of the current available CPU bandwidth for each core of the corresponding machine. We note that more efficient and effective representations are possible, but we leave the exploration of such interfaces for future work.

The NFV component is abstracted using an interface model

$$\langle \mathsf{Period}, \mathsf{Budget}, \mathsf{MinBW}, \mathsf{MaxPacketRate} \rangle,$$

where Budget is the minimum amount of CPU time that must be available to the component in each period of Period time units, MinBW is the minimum network bandwidth required for transmitting its output data to a subsequent component, and MaxPacketRate is the maximum traffic rate that can be sent through (an instance of) the component, to ensure that the component meets its SLA constraints. Intuitively, we can send some requests through an instance of a component – which will be executed in a VM – if their total packet rate is no more than MaxPacketRate, and we can assign the VM to a machine if the machine can guarantee Budget execution time units per Period time units for the VM. Note that, by definition, MinBW can be computed directly from MaxPacketRate, the incoming packet size, and the data size ratios of the NFs. The interface of the application is simply a vector of its components' interfaces.

To minimize the VM and transmission overheads, our component

modeling aims to minimize the number of components of an NFV application, such that each component can be feasibly scheduled by a single VM that uses at most one core. To simplify the analysis, we use the inverse of the maximum packet arrival rate of a component as its component-level deadline (i.e., $1/\mathsf{MaxPacketRate}$). In general, the maximum network delay depends on the specific routing, which is unknown during the interface computation. To minimize transmission delay, our resource allocation maximizes locality by always assigning an instance of an NFV application to the same pod. In addition, we limit the network bandwidth that can be reserved for any transmission between two components to be at most the smallest link bandwidth divided by a tunable parameter $\alpha \geq 1$ [1]. As a result, we can compute the maximum network delay of a packet between two components using the packet size and this link bandwidth limitation, assuming the components can be located in any two nodes of a pod.

The decomposition of the application into components and the generation of the components' interfaces can then be formulated together as an optimization problem that aims to minimize the number of components of an application and maximize $\mathsf{MaxPacketRate}$, subjected to three conditions: (i) the total WCET of all NFs along a path of a component is at most its deadline ($1/\mathsf{MaxPacketRate}$); this is necessary to ensure the VM executing a component is schedulable on at most one core, (ii) the network bandwidth requirement MinBW is at most the smallest link bandwidth divided by α, (ii) the sum of the local deadlines of all contiguous components plus the the sum of the maximum network delays between two consecutive components along a path of the applications is at most the application's end-to-end deadline.

The optimization problem can easily be solved using dynamic programming, and its solution gives both the set of components of an application and their interfaces.

Resource allocation and routing. The resource allocation and routing are performed based on the NFV interfaces and the current interfaces of the cloud resources as follows. For each request of an NFV application that arrives at the system, we first attempt to send its entire traffic through the VMs that execute an existing instance of the NFV application for the same tenant, if the sum of the maximum packet rate of the new request and that of all existing requests through this instance is at most $\mathsf{MaxPacketRate}$, where $\mathsf{MaxPacketRate}$ is the maximum packet rate specified by the application's interface. If no such instance exists, we create a new instance of the NFV application (or multiple instances, if the packet rate of the request exceeds $\mathsf{MaxPacketRate}$) and find a new assignment for its components based on their interfaces. An assignment of components to machines is feasible if (i) for each component C_i, the assigned machine contains a core with current available CPU bandwidth at least equal to C_i's interface bandwidth (i.e., $\mathsf{Budget_i}/\mathsf{Period_i}$), and (ii) if its successor component C_j is assigned to a different machine, then there exists a route from the machine of C_i to the machine of C_j such that all links along the route have the available bandwidth of at least $\mathsf{minBW_i}$.

We formulate the allocation as an optimization problem that aims to find a feasible assignment with the maximum number of admitted requests that most balances the total remaining network bandwidth between the pods and the core switches, as well as the remaining total utilization of the racks in a pod. The formulation can be solved efficiently using linear programming with integer rounding.

[1]Intuitively, the larger α is, the easier it is to find a feasible route during run time (as the network demand is small); however, the the transmission delay also becomes larger, and thus the components' local deadlines must be smaller and hence there will be more components per application.

Evaluation. To test scalability, we performed large-scale simulations for infrastructures with up to thousands of machines, using a greedy heuristic (based on an extension of the strategy from [11]) as a baseline. We found that a) for this very simple interface model, resource assignment can be done very quickly, and that b) on the same hardware, our approach can schedule many more requests than the baseline, and can thus achieve a far better utilization of the available resources. We also conducted emulation on a small local cloud testbed, with three racks of machines and 40 cores, using NFV applications with a simple chain that are made of firewall and network address translation services. The results show that almost all packets of the accepted requests met their deadlines, and the number of requests that meet their SLAs under our approach is three times that of the baseline. The remaining deadline misses of the accepted requests were due to overheads, which were not accounted for in this preliminary work.

Although the workload and setting in our experiments were relatively simple, we consider these initial results to be promising; they show that our approach works in principle, and they illustrate some of the potential benefits.

6. CONCLUSION

We have presented a compositional approach towards building a resource management system that can provide SLA guarantees for NFV applications on the cloud, through the use of component abstraction and interface analysis. We discussed several open challenges towards realizing this approach, which showcase the differences between the considered setting and traditional real-time systems, and we highlighted some potential solutions. We also presented premilinary results that validate the feasibility and demonstrate the potential benefits of our approach.

Acknowledgement

This work was supported in part by NSF CNS 1563873 and CNS 1329984, ONR N00014-16-1-2195, and the Defense Advanced Research Projects Agency (DARPA) under Contract No. HR0011-16-C-0056.

References

[1] Kvm. http://www.linux-kvm.org/.

[2] RT-Xen: Real-Time Virtualization Based on Compositional Scheduling. https://sites.google.com/site/realtimexen/.

[3] Xen. http://www.xenproject.org.

[4] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. A view of cloud computing. *Commun. ACM*, 53(4):50–58, Apr. 2010.

[5] N. Bansal, A. Caprara, and M. Sviridenko. A new approximation method for set covering problems, with applications to multidimensional bin packing. *SIAM Journal on Computing*, 39(4):1256–1278, 2009.

[6] C. Chekuri and S. Khanna. On multidimensional packing problems. *SIAM journal on computing*, 33(4):837–851, 2004.

[7] D. de Niz and L. T. X. Phan. Partitioned Scheduling of Multi-Modal Mixed-Criticality Real-Time Systems on Multiprocessor Platforms. In *Proceedings of the 20th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2014.

[8] L. Epstein and M. Levy. Dynamic multi-dimensional bin packing. *J. Discrete Algorithms*, 8(4):356–372, 2010.

[9] L. Epstein and R. van Stee. Optimal online algorithms for multidimensional packing problems. *SIAM Journal on Computing*, 35(2):431–448, 2005.

[10] S. L. Garfinkel. An evaluation of amazon's grid computing services: EC2, S3 and SQS. Technical Report TR-08-07, Computer Science Group, Harvard University, 2008.

[11] A. Gember, A. Krishnamurthy, S. S. John, R. Grandl, X. Gao, A. Anand, T. Benson, A. Akella, and V. Sekar. Stratos: A network-aware orchestration layer for middleboxes in the cloud. *CoRR*, 2013.

[12] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica. Dominant resource fairness: Fair allocation of multiple resource types. In *8th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2011.

[13] S. Kandula, S. Sengupta, A. G. Greenberg, P. Patel, and R. Chaiken. The nature of data center traffic: measurements & analysis. In *IMC*, 2009.

[14] D. Karger and K. Onak. Polynomial approximation schemes for smoothed and random instances of multidimensional packing problems. In *SODA*, volume 7, pages 1207–1216, 2007.

[15] J. Lee, S. Xi, S. Chen, L. T. X. Phan, C. Gill, I. Lee, C. Lu, and O. Sokolsky. Realizing compositional scheduling through virtualization. In *Proceedings of the 18th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2012.

[16] Y. Li, L. T. X. Phan, and B. T. Loo. Network functions virtualization with soft real-time guarantees. In *Proc. IEEE International Conference on Computer Communications (INFOCOM)*, 2016.

[17] C. Liu, L. Ren, B. T. Loo, Y. Mao, and P. Basu. Cologne: A Declarative Distributed Constraint Optimization Platform. In *Proceedings of VLDB Conference*, 2012.

[18] J. Mars, L. Tang, R. Hundt, K. Skadron, and M. L. Soffa. Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-44, 2011.

[19] S. Mukkamala and A. H. Sung. Detecting Denial of Service Attacks Using Support Vector Machines. In *IEEE International Conference on Fuzzy Systems (IEEE FUZZ)*, 2003.

[20] D. Novaković, N. Vasić, S. Novaković, D. Kostić, and R. Bianchini. Deepdive: Transparently identifying and managing performance interference in virtualized environments. In *Proceedings of the 2013 USENIX Conference on Annual Technical Conference (ATC)*, 2013.

[21] L. T. X. Phan, I. Lee, and O. Sokolsky. Compositional analysis of multi-mode systems. In *Proceedings of the 23th Euromicro Conference on Real-Time Systems (ECRTS)*, 2010. Available from http://repository.upenn.edu/cgi/viewcontent.cgi?article=1468&context=cis_papers.

[22] L. T. X. Phan, I. Lee, and O. Sokolsky. A semantic framework for multi-mode systems. In *Proceedings of the 17th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2011. Available from http://repository.upenn.edu/cgi/viewcontent.cgi?article=1495&context=cis_papers.

[23] L. T. X. Phan, J. Lee, A. Easwaran, V. Ramaswamy, I. Lee, and O. Sokolsky. Carts: A tool for compositional analysis of real-time systems. In *Proceedings of the 3rd Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (CRTS)*, 2010. Tool available from http://rtg.cis.upenn.edu/carts.

[24] 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 L. Eggert, J. Ott, V. N. Padmanabhan, and G. Varghese, editors, *SIGCOMM*, pages 13–24. ACM, 2012.

[25] I. Shin and I. Lee. Compositional real-time scheduling framework with periodic model. *ACM Transactions on Embedded Computing Systems*, 7(3):1–39, 2008.

[26] D. Shue, M. J. Freedman, and A. Shaikh. Performance isolation and fairness for multi-tenant cloud storage. In *10th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2012.

[27] M. Xu, L. T. X. Phan, I. Lee, O. Sokolsky, S. Xi, C. Lu, and C. D. Gill. Cache-Aware Compositional Analysis of Real-Time Multicore Virtualization Platforms. In *Proceedings of the 34th IEEE Real-Time Systems Symposium (RTSS)*, 2013.

[28] H. Yang, A. Breslow, J. Mars, and L. Tang. Bubble-flux: Precise online qos management for increased utilization in warehouse scale computers. In *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ISCA '13, 2013.

[29] X. Zhang, E. Tune, R. Hagmann, R. Jnagal, V. Gokhale, and J. Wilkes. Cpi2: Cpu performance isolation for shared compute clusters. In *Proceedings of the 8th ACM European Conference on Computer Systems (EuroSys)*, 2013.