Cloud-Scale Application Performance Monitoring with SDN and NFV

Guyue Liu and Timothy Wood
The George Washington University

Abstract— **In cloud data centers, more and more services are deployed across multiple tiers to increase flexibility and scalability. However, this makes it difficult for the cloud provider to identify which tier of the application is the bottleneck and how to resolve performance problems. Existing solutions approach this problem by constantly monitoring either in end-hosts or physical switches. Host based monitoring usually needs instrumentation of application code, making it less practical, while network hardware based monitoring is expensive and requires special features in each physical switch. Instead, we believe network wide monitoring should be flexible and easy to deploy in a non-intrusive way by exploiting recent advances in software-based network services. Towards this end we are developing a distributed software-based network monitoring framework for cloud data centers. Our system leverages knowledge of topology and routing information to build relationships between each tier of the application, and detect and locate performance bottlenecks by monitoring the network from software switches.**

I. Introduction

The applications running inside public clouds and private data centers are growing in size and complexity. Even a relatively straightforward web application is likely to be composed of multiple interacting service components: a front-end web server, a caching tier, an application server, and a database tier, each of which may be replicated to support higher demand. The result is a complicated distributed application that may exhibit performance bottlenecks and unpredictable behavior depending on server and network load.

Understanding the behavior of these applications is further complicated by the shared nature of modern data centers that rely on server multiplexing for cost and energy efficiency. Thus a data center will host many different applications, sharing a mixture of server, storage, and network resources. Further, the data center or cloud operator is unlikely to have expert knowledge (or in some cases *any knowledge*) about the application components being run across its servers. This makes diagnosing performance problems particularly difficult, since in many cases the servers running the applications cannot be easily modified to provide information about application-level performance.

The use of virtual machines running on a hypervisor provides one opportunity for observing the behavior of individual application components, but this still typically provides only coarse grained resource consumption information, and it must be aggregated across hosts to understand overall application performance.

Instead, we are developing a network monitoring and performance debugging system that takes advantage of recent advances in Network Function Virtualization (NFV), which allows high speed packet processing software to be built into the network itself. These virtual machine-based network services can be instantiated on demand, and Software Defined Networking (SDN) can be used to route flows to them for monitoring when needed. This allows the software-based network infrastructure to efficiently observe packet flows and infer performance data from many hosts, while treating applications as black-boxes.

To achieve this, we are developing NetAlytics, a platform for high speed cloud network analysis. NetAlytics can efficiently monitor packet flows at rates of 10Gbps or more, sending a sample of packets or aggregated flow statistics to queuing and processing engines for further analysis. In this paper we discuss the challenges of cloud-scale network monitoring and our progress building NetAlytics, which includes:

- NFV-based network monitoring components that can be deployed precisely where they are needed to efficiently monitor packet flows.
- A control framework that uses SDNs to split traffic and send targeted flows to the monitoring system.
- A processing engine that can perform real-time analytics on the aggregated data, providing insights into application performance and network behavior.

We describe our initial prototype and demonstrate how it can be used to have the network monitor response time statistics for a multi-tier web application, allowing cloud administrators or an automated resource management system to detect bottlenecks and understand per-tier performance. We also illustrate how more sophosticated statistics such as "top-k" content popularity analysis can be performed in real time on network data.

II. Network and Performance Monitoring

This section provides background on approaches to monitoring application performance in cloud environments and recent developments in software defined networking and network function virtualization.

A. Cloud Application Monitoring

Infrastructure as a Service cloud platforms and private data centers provide elastic scalability by allowing virtual machines to be replicated or resized to meet changing workloads. The decisions to perform these changes can either come manually from system administrators or through an automated resource controller. In either case, some kind of monitoring agent is typically what triggers the alerts that

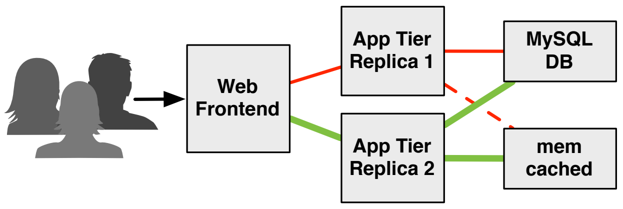

Fig. 1. This multi-tier web application has been misconfigured, causing App Tier 1 to send requests only to the database instead of making use of the cache; this kind of performance issue can be very difficult to diagnose without detailed information of per-tier application performance.

suggest an application may not be meeting its Service Level Agreements (SLAs).

Many performance management systems simply rely on resource consumption as a proxy for understanding performance [17], [11]. However, this is not always effective, particularly for dealing with performance issues in multi-tier applications where it may be difficult to determine which tier is the source of a performance problem. Consider the multi-tier web application illustrated in Figure 1. The application has a web front end that load balances requests to a replicated application tier. The application tiers retrieve data either from a MySQL database or a memcached in-memory cache. In this example, App Replica 1 has been misconfigured so that its connection to the memcached node is not being used, causing signficantly higher response times for requests since they need to be processed by the database instead of the much faster cache. This kind of performance anomaly is commonly seen, where due to the complex nature of modern distributed web applications, only a subset of requests may see a performance issue.

If resource monitoring were used on this system, it would find the CPU load on both App tier replicas to be very similar–each is simply receiving an input request and then making a network call to either the database or the memcached server, causing a similar load on CPU, memory, and the network. This hides the fact that the performance issue is really a misconfiguration on one of the nodes—simply using resource monitoring will be unable to reveal the source of this issue (and may not even indicate that there is a problem, unless the database's resources become overloaded). Only by understanding actual application-level performance broken down by tier can a system administrator determine the source of this kind of performance anomaly.

Amazon EC2's CloudWatch monitoring service can transparently observe the resource utilization levels on individual VMs, or VMs can be modified to include a monitoring agent that provides additional details such as application response times. However, instrumenting applications in this way can be difficult and requires actions by the owner of each virtual machine. This prevents the monitoring system (and thus any automated management systems that would rely on its data) from being completely controlled by the cloud provider or data center operator—ideally, clouds should be able to offer services such as dynamic performance resource management in a completely transpartent way.

One approach for transparently determining application performance is to intercept network packets and infer application behavior from this information. Aguilera et al. [1] treat each componet of a distributed system as a blackbox and perform performance debugging by observing message traces. NetCheck [19] performs diagnosis by using a blackbox tracing mechnism to collect a set of system call invocation traces from the relevant end-hosts. We have the same end-goals as these works but in this paper we focus on how to efficiently deploy network monitors in software-based networks and use a scalable processing engine to analyze network data in real time.

B. SDN and NFV Background

Software-Defined Networking (SDN) [4] is changing how networks are managed by providing a logically centralized controller capable of having a network-wide view and applying carefully crafted rules that target individual flows. At the same time, Network Function Virtualization (NFV) has emerged as a technique to run high performance network services as softwares running in virtual machines (VMs) on low cost commodity servers. [10], [5]

Recent projects [13], [6], [7], [2] propose to do network monitoring and troubleshooting under the SDN architecture. For example, NetSight [7] records the full packet histories and provides an API for network analysis programs to diagnose problems. OpenSample [2] leverages sFlow packet sampling functionality in physical switches to measure network load and individual flows. EMC2 [13] proposes monitoring network flows inside virtual swiches in the host to achieve network wide monitoring and improve scalability. Planck [15] improves network-wide monitoring performance by employing oversubscribed port mirroring to extract network information in commodity switches.

Compared to traditional network, SDN provides network-wide visibility and allows for constantly monitors network conditions and reacts rapidly to problems [15].

Existing SDN-based networking monitoring apporaches mainly focus on individual packets or flows, and simply query switches to get packet counts that match different flow table entries. This is sufficient for raw packet counts/bandwidth monitoring, but doesn't give sufficient detail for real performance monitoring or network debugging. Thus, we propose to intergate NFV-based monitors into SDN architecture so that monitors can interpret packet headers or even packet bodies to gather much more detailed monitoring information while leveraging controller's global topology information.

III. NetAlytics Design

NetAlytics will provide a platform for understanding application and network behavior in real time. The system is composed of components deployed into the network to analyze and process data, and a control plane framework that instantiates the data plane monitoring and analysis components and routes flows to them on demand. This allows

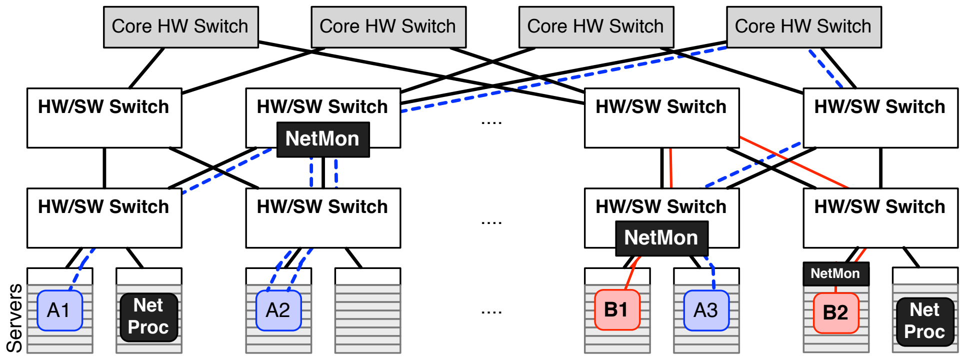

Fig. 2. Future data centers will be built from mixed hardware and software networking components, allowing network monitors to be strategically placed to observe applications and send their aggregated data to processing elements.

monitoring to be seamlessly integrated into the network precisely when and where it is needed, and ensures that the processing capacity will scale up and down with changing workloads.

A. Network Architecture

With the advent of fast software-based packet processing engines based on netmap [16], [14] and DPDK [3], [9], we believe that future data center networks will be composed of a mix of hardware and software switches. This architecture, illustrated in Figure 2, uses hardware packet switches in the core of the network where speed is of utmost importance, while intermediate and rack-level switches have more flexible software switching capabilities (or potentially a mix of hardware and software switching within a single box). A distributed application within the data center is likely to be spread across multiple network hops, so a single switch may not have complete visibility into the behavior of an application's components.

The goal of NetAlytics is to provide detailed real-time analysis of network and application behavior throughout this type of cloud network. The NetAlytics platform is composed of three main component types:

Monitors: are sources of data, typically residing on a software-based switch or an end-host server. They may perform some minimal processing of the data stream to reduce the volume of data they produce. For example, a web application performance monitor might sample a subset of flows and filter packets by the SYN and FIN flags to detect the start and end of each connection, it would then emit measurements that indicate the time to receive and process a web request over an HTTP connection.

Aggregators: are servers that take input from one or more monitors and hold it in a queue until it can be processed. Aggregators may also be configured to perform simple processing such as calculating basic statistics across several monitor streams. An example Aggregator might partition incoming HTTP request response time measurements into different queues based on the server or content being requested.

Processors: implement the desired analysis on the data retrieved from aggregators. We use an existing streaming data analytics engine to facilitate development of applications that can perform real-time analysis of the monitored data. For example, a Processor could pull data from specific queues on an Aggregator to provide live analysis of response times for different portions of a website.

In NetAlytics, we are building network data stream Monitors using DPDK and our NetVM platform. We use Apache Kafka, a high throughput distributed messaging system, for our Aggregators, and implement our Processors on the Apache Storm real-time distributed computation engine. We provide further details of our implementation in Section IV.

B. Network Analytic Queries

NetAlytics must deploy these network components in order to answer some analytic query issued by a user. For example, a user might wish to answer one of these questions:

- What is the response time breakdown per component for a mult-tier web application?
- Which servers are consuming the largest fraction of bandwidth over the most congested links?

Depending on the question, NetAlytics will have to deploy a different set of Monitors, Aggregators, and Processors.

NetAlytics assumes access to the topology of each application that must be monitored within the cloud. This includes both the set of components that form an application and their physical location within the network (i.e., which switches they are connected to and the network topology). This information can either be provided by the system administrators deploying applications into the cloud, or through automated topology detection methods [8], [18].

Given this topology, NetAlytics can determine which network links need to be monitored, and what data needs to be gathered about each flow. In our current implementation, the data to be gathered and the processing to be performed on each data tuple needs to be specified manually, but our goal is to generalize this to provide a more expressive framework.

The first step is to determine which data must be gathered. Ideally, users should be able to use SQL-like syntax to specify which applications and data fields they are interested in analyzing. Note that NetAlytics only enables monitoring when it is explicitly needed, so the "select statements" would be a trigger to decide what new data needs to be collected.

Given a data stream defined by the above query, NetAlytics will seek to provide a convenient way to perform common data analysis operations on the stream. This will be achieved through a collection of Processor elements implemented by a library of common functions. These library components would provide statistical tools ranging from simple running averages to more sophisticated "top-k" analysis.

With these two building blocks, a system administrator will be able to easily specify what type of data they are interested in and connect together processing elements that can perform the desired analytics. NetAlytics will then deploy the necessary services into the network to gather, aggregate, and process the data.

C. Control Plane

NetAlytics's Monitors, Aggregators, and Processors should be deployed into the network only *when* they are needed, and only *where* they can observe the specific applications being studied. A 10Gbps link can generate over 14 million packets per second, so continuously running monitors that gather fine grained statistics can be extremely wasteful. NetAlytics takes advantage of NFV to deploy monitors at the locations where they can most effectively analyze traffic between components of the targeted applications, and uses SDN controllers to direct packet flows to these monitoring services.

Instantiating NFV Services: our NetVM platform [9] facilitates deploying network functions such as packet monitoring into virtual machines. NetVM's management framework on each host can start a VM running a desired application and then ensure that incoming flows matching certain patterns are directed through the appropriate set of network functions. In our current setup, we are able to instantiate a new NFV service in less than four seconds. For NetAlytics, we are extending the NetVM platform to integrate with the NetAlytics controller, which will inform specific hosts that they should start Monitors, and provide them with the flow matching rules so that only the desired applications or users are monitored.

Mirroring Flows to Monitors: The NetAlytics Controller runs as an application integrated with an SDN Controller, allowing it to easily manipulate how packets are being forwarded throughout the network. In our current approach, we do not modify the forwarding paths of existing flows in the network—NetAlytics simply places the Monitor network functions within the existing paths being taken by the application under study. The SDN controller then issues commands to the relevant switches to enable port mirroring. This duplicates the traffic and sends it to the host running the monitoring service. By mirroring the traffic, we avoid having the monitoring infrastructure manipulate the existing networks, which could inadvertently impact the behavior being monitored.

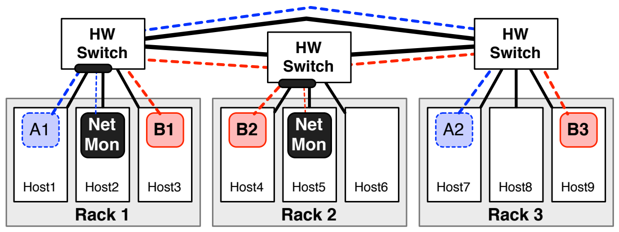

Fig. 3. NetAlytics uses port mirroring to duplicate the desired flows and send them to the network monitors. A monitor must be attached to at least one switch on the path between each communicating pair.

For example, consider the network illustrated in Figure 3. The NetAlytics user has requested performance monitoring for both applications A and B. The NetAlytics controller will cause monitoring services to be started in VMs on Host 2 and Host 5. The controller then uses OpenFlow to send a rule to the switches they are connected to to mirror traffic to those hosts. It would be possible to redirect traffic for application A through the middle switch so that a single monitor could be used on Host 5 to observe both applications. This could provide a more efficient monitoring infrastructure, but it also may negatively impact performance of other applications. In our future work we will explore the trade-offs of using different placement algorithms to deploy monitoring services.

IV. NetAlytics Implementation

We have been developing a prototype that includes some of NetAlytics's key components.

A. NFV Monitors

We have built a DPDK-based network monitor that can monitor packet flows at line-speed and gather statistics about web application performance such as response time. This can be installed inside virtual machines using SR-IOV to allow DPDK to directly access the NIC. We will be porting this code to our NetVM platform which provides for both faster packet processing in VMs and allows chains of monitoring services to be easily connected together on a single host, and managed by a SDN controller.

The network monitor receives a copy of every packet on the specified link. It can inspect the packet headers to determine the source/destination or view the other flags. We use this capability in our response time monitor by checking TCP ACK/SYN/FIN flags to determine the start and end of each connection and estimate response time. In some cases, such as monitoring performance of specific web queries, it can be necessary to inspect the packet body in order to examine data such as HTTP request contents. Our monitor is capable of performing this form of deep-packet inspection while maintaining line rate throughputs.

B. Realtime Processing with Storm

We are using Apache Storm as our processing engine to analyze network data in real time. Storm provides a distributed computation engine similar to the Map Reduce framework, but it focuses on processing streams of data in real-time. This provides a convenient platform to write programs that analyze the data gathered by the monitors.

Our current prototype uses Storm to process monitoring logs that have been saved to disk by an NFV monitor. In the future, we will use a queuing service such as Kafka [12] for our Aggregators, and will enhance our NFV monitors so that they can send records to the queuing service. We can also use a database to store output from Storm so that our system can support both short-term and long-term queries.

V. NetAlytics Prototype Use Cases

We have developed two applications that use our NetAlytics framework to study the performance of multi-tier applications and the popularity of content for a video streaming cluster.

A. *Multi-Tier Web Application Performance Debugging*

Our first use case shows how NetAlytics can detect bottlenecks and debug network performance issues by monitoring response time statistics. For these experiments we have the NetAlytics Monitor directly process the incoming streams, without using a processing engine.

We first configure our servers to act like the misconfigured web application from Figure 1, where one replica is sending requests to a fast memcached node while the other is using a slower database. We program our monitor to gather connection timing information by detecting the length of each TCP connection between the different application tiers by filtering for SYN and FIN packets. From the table below we can see that the CPU utilization on both application servers is identical, but because App Server 1 is misconfigured, it sees a significantly higher average response time. NetAlytics is able to help diagnose this problem by providing the response time breakdown for each component in the multi-tier application.

	Response Time	CPU Utilization
App Server 1	372 (ms)	%12
App Server 2	15 (ms)	%12

Full connection times can sometimes be too coarse grained since a client may issue multiple requests over a connection so we estimate the response time by continuously measuring the intervals between each ACK sent from server side, this information is an important metric for determining network behavior.

We next use one client and two servers to emulate a multi-tier application, with the goal of determining the response time breakdown per-tier at fine granularity. We run a configurable PHP application to create different load on the two web tiers and chain them together. Our NetAlytics monitor observes the ACK packet exchanges to determine the request cost breakdown within and across tiers.

Figure 4 illustrates how the NetAlytics monitor can use the network trace to measure the time for data exchanges between tiers. NetAlytics measures the time for the client's entire HTTP connection as 763 ms, with a breakdown of

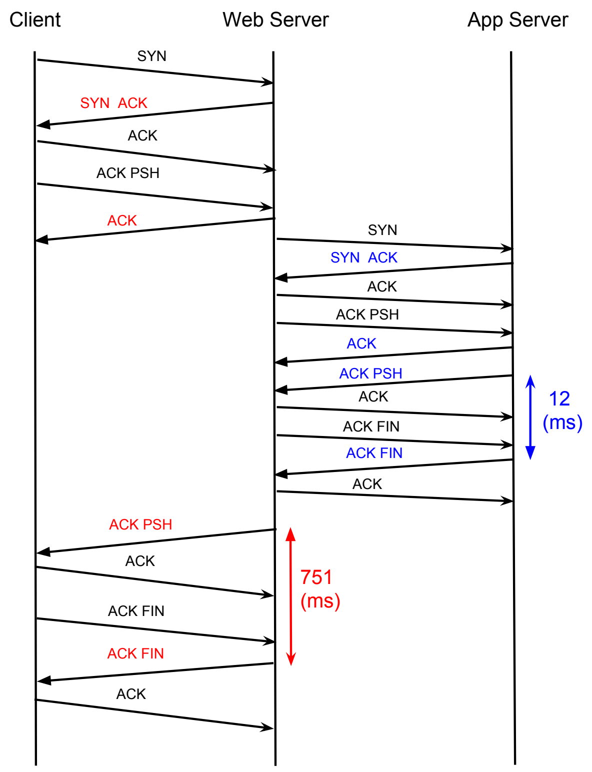

Fig. 4. Response time for each tier of web application

12 ms at the App tier, followed by 751 ms at the Web tier. This helps system administrators to find which nodes are the bottleneck, and helps application developers track down the sections of their code that have high cost—in this case, the performance overhead occurs in the code run after a connection to the App server is completed.

B. *Real-Time Content Popularity Monitoring*

Next we show how to use Apache Storm as a processing engine for network data. In particular, we show Storm can keep track of top-k content popularity in real time. We use traces released by Zink et al. [20] which contain information about client requests for YouTube video clips recorded at a network gateway. We analyze trace T6 which is composed of seven consecutive 24-h traces and includes a unique VideoID for each Youtube video and the content server IP address for each request. In a live deployment, this information could be easily gathered in a network monitor by processing the body of client request packets to determine the content being requested and the accessed video server.

We use Storm to process the data and find the Top-100 popular videos within a measurement interval. The table below shows the most popular videos and content servers measured over the entire trace. Currently, this kind of information is usually gathered by using daily or weekly log data analysis tools such as Hadoop. With NetAlytics, we can provide these types of insights at much finer grained time intervals, and *with no instrumentation of the video server applications themselves*. Figure 5 shows the popularity of

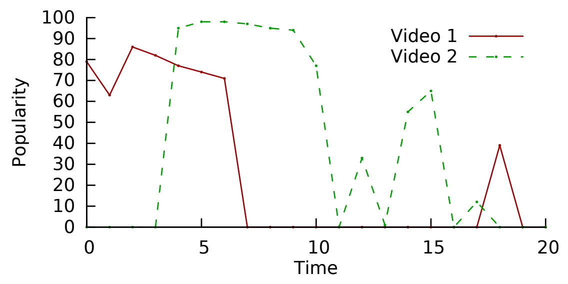

Fig. 5. NetAlytics uses Storm to measure video content popularity over time (100 is most popular video).

the 2nd and 3rd most popular videos over time as reported by Storm. In this case we can see that even this top content sees fluctuations over time. This real time information could be useful for optimizing resource allocations and caching behavior.

Rank	Video ID	#hits	Content Server	#hits
1	x9QNKB34cJo&origin	223	105.136.3.123	20522
2	7sei-eEjy4g&origin	215	105.136.3.63	14863
3	tyKtQ0216bo&origin	162	105.136.78.115	13408
4	4rb8aOzy9t4&origin	159	105.136.78.93	10136
5	edhpwSMqc2I&origin	155	105.136.78.16	9891

TABLE I
TOP 5 VIDEOS AND CONTENT SERVERS OF YOUTUBE TRACE

VI. CONCLUSIONS AND FUTURE WORK

Public clouds and private data centers are filled with a diverse range of applications, most of which are now distributed across multiple networked servers. This makes understanding the performance of these applications challenging, particularly when it is not feasible to directly instrument the applications or the virtual machines they run in. We have described NetAlytics, a platform for large-scale performance analysis by processing network data. Our system takes advantage of Network Function Virtualization to deploy software-based packet monitors deep into the network, and uses Software Defined Networking to steer packet flows to the monitors. Once data has been captured, it is aggregated and sent to a processing engine based on the Apache Storm real-time data analytics engine. We have illustrated how our platform can help diagnose performance issues by measuring per-tier response times and by providing insight into content popularity.

In our ongoing and future work we will further enhance NetAlytics to support a convenient language for defining the flows and packet data to be monitored, and a set of processing building blocks for common real-time analytic tasks. We are also exploring the placement problem to decide how network monitors and processing engines should be positioned within the network to minimize both bandwidth costs and resource consumption.

Acknowledgments: We thank the reviewers for their feedback; this work was supported in part by NSF grants CNS-1422362 and CNS-1253575.

REFERENCES

[1] M. K. Aguilera, J. C. Mogul, J. L. Wiener, P. Reynolds, and A. Muthitacharoen. Performance debugging for distributed systems of black boxes. In *ACM SIGOPS Operating Systems Review*, volume 37, pages 74–89. ACM.

[2] C. D. W. F. J. Carter. OpenSample: A low-latency, sampling-based measurement platform for SDN.

[3] I. Cerrato, M. Annarumma, and F. Risso. Supporting fine-grained network functions through intel DPDK.

[4] N. Feamster, J. Rexford, and E. Zegura. The road to SDN. 11(12):20:20–20:40.

[5] X. Ge, Y. Liu, D. H. C. Du, L. Zhang, H. Guan, J. Chen, Y. Zhao, and X. Hu. OpenNFV: Accelerating network function virtualization with a consolidated framework in OpenStack. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14. ACM.

[6] N. Handigol, B. Heller, V. Jeyakumar, D. Mazires, and N. McKeown. I know what your packet did last hop: Using packet histories to troubleshoot networks. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 71–85. USENIX Association.

[7] B. Heller, C. Scott, N. McKeown, S. Shenker, A. Wundsam, H. Zeng, S. Whitlock, V. Jeyakumar, N. Handigol, J. McCauley, and others. Leveraging SDN layering to systematically troubleshoot networks. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 37–42. ACM.

[8] J. Hwang, G. Liu, S. Zeng, F. Wu, and T. Wood. Topology discovery and service classification for distributed-aware clouds. In *2014 IEEE International Conference on Cloud Engineering (IC2E)*, pages 385–390.

[9] J. Hwang, K. Ramakrishnan, and T. Wood. NetVM: High performance and flexible networking using virtualization on commodity platforms. In *Symposium on Networked System Design and Implementation*, NSDI 14.

[10] E. T. S. Institute. Network functions virtualization (NFV): An introduction, benefits, enablers, challenges & call for action.

[11] G. Khanna, K. Beaty, G. Kar, and A. Kochut. Application performance management in virtualized server environments. In *Network Operations and Management Symposium, 2006. NOMS 2006. 10th IEEE/IFIP*, pages 373–381. IEEE.

[12] J. Kreps, N. Narkhede, J. Rao, and others. Kafka: A distributed messaging system for log processing. In *Proceedings of the NetDB*.

[13] V. Mann, A. Vishnoi, and S. Bidkar. Living on the edge: Monitoring network flows at the edge in cloud data centers. In *COMSNETS'13*, pages 1–9.

[14] 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. USENIX Association.

[15] J. Rasley, B. Stephens, C. Dixon, E. Rozner, W. Felter, K. Agarwal, J. Carter, and R. Fonseca. Planck: millisecond-scale monitoring and control for commodity networks. pages 407–418. ACM Press.

[16] L. Rizzo. netmap: A novel framework for fast packet i/o. In *USENIX Annual Technical Conference*, pages 101–112. USENIX.

[17] VMware. Resource management with VMware DRS.

[18] X. Zhang, Y. Zhang, X. Zhao, G. Huang, and Q. Lin. SmartRelationship: a VM relationship detection framework for cloud management. pages 72–75. ACM Press.

[19] Y. Zhuang, E. Gessiou, S. Portzer, F. Fund, M. Muhammad, I. Beschastnikh, and J. Cappos. NetCheck: Network diagnoses from blackbox traces. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 115–128. USENIX Association.

[20] M. Zink, K. Suh, Y. Gu, and J. Kurose. Characteristics of YouTube network traffic at a campus network measurements, models, and implications. 53(4):501 – 514. Content Distribution Infrastructures for Community Networks.