# Toward a Software-Based Network: Integrating Software Defined Networking and Network Function Virtualization

**Timothy Wood, K. K. Ramakrishnan, Jinho Hwang, Grace Liu, and Wei Zhang**

**Abstract**

Communication networks are changing. They are becoming more and more "software-based." Two trends reflect this: the use of software defined networking and the use of virtualization to exploit common off-the-shelf hardware to provide a wide array of network-resident functions. To truly achieve the vision shared by many service providers of a high-performance software-based network that is flexible, lower-cost, and agile, a fast and carefully designed network function virtualization platform along with a comprehensive SDN control plane is needed. The shift toward software-based network services broadens the type of networking capabilities offered in provider networks and cloud platforms by allowing network services to be dynamically deployed across shared hosts. Combining this with an SDN control plane that recognizes the power of a dynamically changing network infrastructure allows network functions to be placed when they are needed and where they are most appropriate in the network. Our system, SDNFV harmoniously combines the two fast moving technological directions of SDN and virtualization to further the goal of achieving a true software-based network.

Networks have traditionally been composed of interconnected hardware such as routers, switches, and firewalls. Employing purpose-built hardware appliances, managed through distributed protocols, has allowed networks to achieve high performance and reliability, but it comes at the cost of limited flexibility. This tradition has mostly continued, with such purpose-specific hardware systems being deployed even for typical software functions such as proxies, firewalls, and caches, because of the desire to have a high-performance data plane. This limits the flexibility of network functions, has high cost, and makes it difficult to deploy services dynamically.

Cloud data centers have increased their efficiency and flexibility by employing virtualization techniques that allow convenient (often centralized) management of dynamically created server instances. With the adoption of software defined Nnetworking (SDN) and network function virtualization (NFV), a similar revolution is happening in both wide area networks and data center networks. SDN provides a logically centralized control plane that can flexibly direct packet flows between network devices based on programmable policies [1–3]. NFV transforms networks from hardware appliances with customized application-specific integrated circuits (ASICs) into software running in VMs (VMs) on common off-the-shelf (COTS) hardware to increase flexibility and lower cost [4–6]. Together, SDN and NFV offer the potential to radically alter how networks are deployed and managed.

By moving to a software-based environment, NFV makes network services easy to deploy, and allows them to be more powerful and flexible, enabling more complex topologies and feature-rich network resident functions compared to hardware-based implementations. Unfortunately, the performance limitations of commodity hardware and the overhead of server virtualization platforms have prevented high-performance network processing to fully transition away from hardware-based routers and middleboxes. In this article we describe how a carefully designed NFV architecture can overcome these virtualization-layer overheads through zero-copy packet data transfer, non-uniform memory access (NUMA)-aware scheduling, and lockless data structures. Our approach exploits advances in multi-core CPUs and modern network interface cards (NICs) to enable a truly flexible, VM-based networking platform that can process packets at line rates.

SDN has already impacted how networks are deployed and managed, but current approaches do not fully exploit the benefits of an NFV-based infrastructure. SDN controllers still assume they are interacting with simple hardware devices that are incapable of making decisions on their own. The reality is that increasing deployments of middleboxes and NFV-based services means that not only will flow management become more complex, but also that data plane elements will want to make dynamic decisions about how packets are directed. The ability to dynamically steer flows through selected service functions is a key capability for network service providers.

*Timothy Wood, Grace Liu, and Wei Zhang are with George Washington University.*

*K. K. Ramakrishnan is with the University of California, Riverside.*

*Jinho Hwang is with IBM T. J. Watson Research Center and George Washington University.*


0890-8044/15/$25.00 © 2015 IEEE

Having some of these decisions performed locally by a middlebox in the network, rather than having to go to the centralized controller for orchestrating this capability can result in substantial performance improvement (in terms of both throughput and latency). Here we argue that the existence of a "smart data plane" capable of knowing its local state and basing decisions on such state is key to supporting networks with both high performance and flexibility.

## Evolving the Network Control and Data Planes

Networks have traditionally been constructed from hardware switches and routers that make decisions about forwarding based on the results of a distributed protocol such as Open Shortest Path First (OSPF) and Border Gateway Protocol (BGP). While this decentralized approach provides resilience and performance, it also limits forwarding decisions to simple criteria such as the number of hops along a path. Some approaches such as setting the DS bits in an IP header provide a coarse-grained mechanism for classifying network traffic; however, this places control at the end host, not at the network forwarding elements. In general, classes of packets destined for a particular destination end up being managed in the same way.

As the traffic flowing through networks becomes increasingly heterogeneous, network operators now desire to route traffic at the flow level, allowing differentiation between flows based on more selective criteria such as the individual source and/or destination, the protocol being used, or even packet content. SDNs have enabled this form of traffic management by dividing the network into a smart control plane along with a simple data plane. Rather than have switches use distributed route selection algorithms, they are only responsible for quickly forwarding packets. The data plane routers and switches learn how to route packets by contacting a logically centralized control plane that can be queried to determine the next hop for any packet flow. In essence, SDN develops algorithms that use network-wide data structures rather than distributed algorithms implemented in network elements. SDN-aware network hardware forwards the first packet for a new flow to the SDN controller, allowing it to inspect the packet's headers and apply a complex forwarding policy if necessary. The interaction between the control plane and the data plane is to share packet flow information. The control plane (SDN controller) uses the first packet and periodic statistics reported from network elements to generate rules that are incorporated into flow tables with actions on the packet flows. The data plane refers to these flow tables and sends packets according to actions for matched rules [1, 3].

SDN applications make intelligent decisions to perform traffic engineering, quality of service (QoS) differentiation, monitoring, and routing. In particular, service providers have exploited SDN principles for policy-based routing [7] and policy-based access control [8]. The logical centralization of such policy has been used to significantly simplify the incorporation of policy with BGP routing, while not impacting performance. Another area that has seen use of SDN capabilities is in network virtualization, where different sets of endpoints communicate over different virtual networks, while exploiting a common underlying physical network [4]. Each of these approaches has simplified network operations, especially from the point of view of configuration and provisioning.

Within the data plane, NFV has been developed as a way to run network services with greater flexibility and lower cost by using commodity servers instead of specialized hardware. While software routers such as Click have been around since the early 2000s, it has only recently become possible to process packets in software at line rates of 10 Gb/s or higher. Some approaches such as PacketShader leverage GPUs to provide fast, highly parallelizable processing of packets [9]. Purely software approaches such as netmap and Intel's Data Plane Development Kit (DPDK) exploit multi-core CPUs and zero-copy memory transfer approaches to efficiently move packet data directly to applications, eliminating operating system (OS)-level overheads [10–12]. By applying these techniques within VMs, NFV has further increased the flexibility in how packets are routed [4–6].

Despite these advances within the data plane, SDN assumes that the network elements are simple forwarding entities, such as Ethernet switches, along with queueing and scheduling mechanisms for achieving differentiation and meeting service-level assurances with which flows may be provided. However, with the advent of software-based network entities, it is only natural to examine whether it is appropriate to have the data plane make local decisions based on local state, while the logically centralized controller continues to use algorithms that use network-wide data structures and state that are more feasible to make available to the controller. We believe that this can enable a more flexible, higher-performance network environment that is also more responsive to dynamic changes in the placement of software-based network elements in the network.

## NetVM: An Efficient, Software-Based Data Plane

We have been developing NetVM, a virtual server platform optimized for running network services [13]. Our initial work on NetVM shows the considerable promise of the use of virtualization in a software-based platform. NetVM enables high-bandwidth network functions to operate at near line speed, while taking advantage of the flexibility and customization of low-cost commodity servers. The NetVM platform is the base for our SDN NFV(SDNFV) vision of flexible and dynamic network services.

### NetVM Platform Overview

There are three main challenges that prevent COTS servers from being able to process network flows at line speed. First, network packets arrive at unpredictable times, so interrupts are generally used to notify an OS that data is ready for processing. The overhead of handling millions of interrupts per second when receiving packets at 10 Gb/s or more quickly exceeds the time spent (or even available for) doing useful packet processing. Second, existing OSs typically read incoming packets into kernel space and then copy the data to user space for the application interested in it; these extra packet copies incur a high cost, especially if a packet is being transmitted through a chain of network services running on one host. Finally, network I/O in virtualized settings can have even greater overheads due to additional copies between the hypervisor and guest, and the need to multiplex a device across multiple VMs.

The Intel DPDK platform tries to reduce the first two overheads by allowing user space applications to directly poll the NIC for data. This avoids both interrupt processing and unnecessary data copies. Our NetVM platform extends DPDK for a virtual platform so that packet data can be efficiently transferred to and between VMs, a crucial requirement for the flexible network services envisioned by SDNFV.

Figure 1 illustrates the architecture used by NetVM to

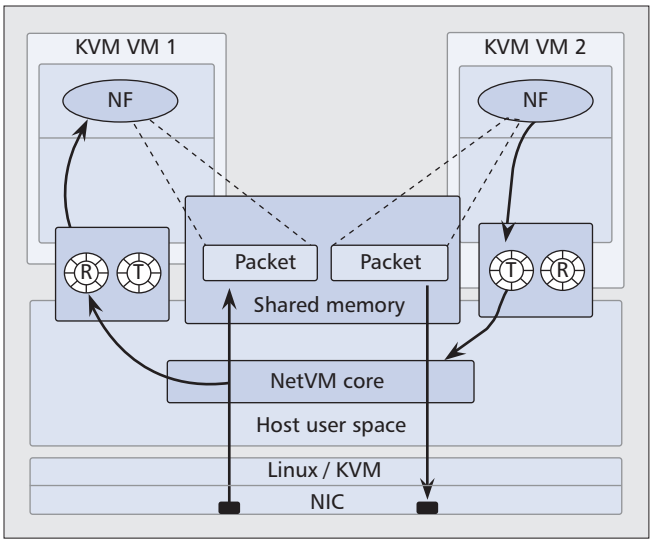


Figure 1. NetVM uses DPDK to direct DMA packet data into a region of memory shared with multiple VMs. Packet descriptors are then exchanged with the VMs, allowing them to examine and modify packet contents before sending them back out the NIC.

overcome these challenges. Our implementation builds on the KVM hypervisor, and contains components running in the host and each guest VM. The NetVM Core component manages VMs and routes flows to them, while individual VMs process packets for the desired network functions.

### Zero-Copy Communication

NetVM's focus is on allowing zero-copy packet delivery from a NIC to a network service composed of one or more VMs. NetVM uses shared huge pages to store all incoming packet data. This memory area is shared between the host OS and all VMs running network services. References to packets are sent between the hypervisor and VMs using a set of per-VM ring buffers for receiving and transmitting packets between domains (denoted as R and T in the figure). This approach is used for all communication in the system; using a shared memory area for large pieces of data combined with a simple queue for message passing avoids expensive copy operations while allowing high scalability across cores.

### Lockless, NUMA-Aware Optimizations

Locks are typically used to ensure consistency of shared data, but even acquiring an uncontested lock can take tens of nanoseconds, which is too large a cost when trying to process packets at line rates. Fortunately, the communication architecture used by NetVM can be implemented safely without any locks since only a single thread writes to each message ring. Since a packet descriptor will be held by only the host or a single VM at a time, packet data in the huge page region will never see concurrent accesses. This approach allows NetVM to avoid the high latency and performance variability inherent in using locks.

Modern servers may have multiple CPU sockets, each connected to different banks of memory. This can result in NUMA costs if a thread is accessing data spread across several dual in-line memory modules (DIMMs). To avoid this, NetVM uses one huge page region per CPU socket, and ensures that a packet stored in one region is only processed by cores on that socket. This prevents NUMA costs and increases cache locality.

### Software-Based Services

NetVM provides a user space library inside each VM that allows services such as routers, firewalls, and deep-packet inspection engines to receive and forward packets between VMs and other hosts. The library greatly simplifies the development of these services, and since they run as user space applications, programmers can more easily reuse and debug their code. The fact that services are run within VMs also greatly facilitates deploying these services: each can be treated as an isolated appliance that can easily be swapped in or out without worrying about dependencies or interference between components.

### NetVM Performance

Currently, the most common way to deploy network services inside VMs is to use single root IO virtualization (SR-IOV), which allows a physical NIC to be divided into multiple virtual interfaces, which are then assigned to each VM. This removes hypervisor-related overheads, and allows libraries like DPDK to be run directly inside each VM. This approach of using hardware virtualization to bypass OS or hypervisor overheads has seen growing popularity for optimizing the critical path for networked applications [12]. However, SR-IOV incurs a

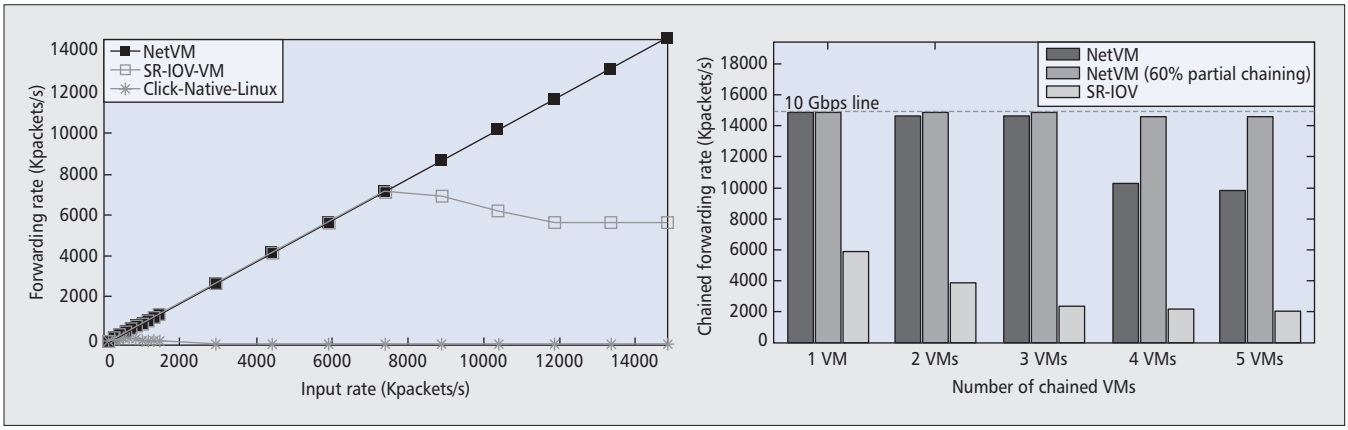


Figure 2. a) The throughput achieved by NetVM is orders of magnitude higher than running a software router in standard Linux, and gets double the performance of SR-IOV-based VM networking; b) the performance of SR-IOV drops even further when steering packets through multiple VMs on a host. NetVM only sees a performance drop after three VMs because the system does not have sufficient CPU cores; if only 60 percent of packets pass through the chain with the remainder only visiting the first VM, NetVM can maintain the line rate of 10 Gb/s.

cost, particularly if packets must be sent through a chain of several VMs on the same host. This is because moving packets between VMs requires transmitting the packet from the source VM to the NIC, where it will be switched to a different virtual interface. Even worse, if the destination VM is connected to a different physical port on the same NIC, the packet must be sent outside of the host to an Ethernet switch that will return the packet to the other port. In contrast, NetVM's optimizations let it achieve scalability across cores and high throughputs of 10 Gb/s, even when transmitting 64-byte packets through long chains of VMs, as shown in Fig. 2.

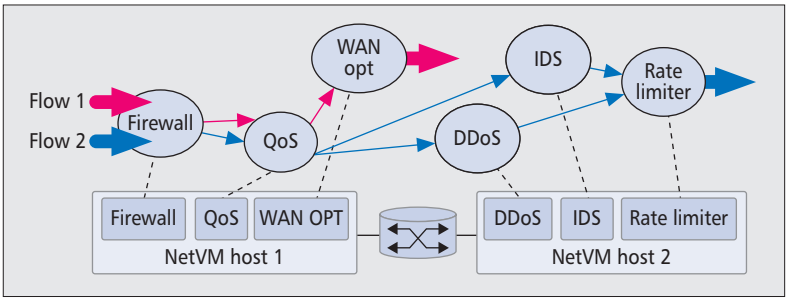


Figure 3. SDNFV uses a service graph to represent the abstract function names required for different flows. The graph is then implemented by functions deployed in VMs across multiple hosts.

## SDNFV: Controlling a Smarter Data Plane

An efficient NFV platform still must be managed by a controller with network-wide view and an understanding of the desired policies that must be applied to different flows. Our ongoing work on SDNFV is exploring how NetVM hosts should be combined with SDN controllers to provide coordinated network management while still taking advantage of the power and flexibility of smart data processing elements.

### Service Naming

We believe that introducing the idea of "names" for services, rather than continuing to retain the traditional location-based identification of service functions in the network, will further improve the performance and flexibility of SDN in an NFV-based network environment. Software-based network resident functions, such as functions to modify packet headers (e.g., NAT, proxy), examine and allow or block packet flows (e.g., firewalls and deep packet inspection devices), or even modify the packet payload (e.g., optimization and compression of video) will allow networks to be more dynamic. But these functions only need to be applied to certain packets. Implementing them in software can provide powerful flexibility to a service provider to place such services at optimal points in the packet flow, rather than in a particular location in the network topology and routing packets to them. Having functions dynamically move around implies that it is necessary to name these services rather than refer to them based on their current location. This has been demonstrated to provide significant performance improvement, particularly in reducing end-to-end latency [14].

Existing approaches unnecessarily couple routing with the policy: that is, when an SDN controller decides the functions a flow needs, it also decides the path the flow has to go through and sets up state on the intermediate switches. These solutions limit scalability and flexibility, making it difficult to adapt to the requirements of a large-scale, dynamically changing environment supported by NFV. Having service names enables an approach that separates the functions a flow needs from the location of network function instances. Such a decoupling facilitates the dynamic modification of the functions needed by a flow.

### Function Composition with Service Graphs

A network operator will have a desired set of functions that should be applied to different types of flows. In SDNFV, this is represented with a service graph that indicates which types of network functions a packet flow should be processed by, and in what order. An example is shown in Fig. 3, where two different types of flows are routed through a different set of network functions. These network functions then must be mapped to individual NetVM hosts, which can provide the desired packet processing capabilities. Using abstract service names allows the mapping of service to virtual instance to be performed far more dynamically than existing approaches.

When a packet arrives at a NetVM host, a flow table lookup is performed to see if the service graph for the packet's flow has already been determined. If not, the packet is sent to the SDN controller, which will find the appropriate service graph and then forward flow table rules to all NetVM hosts related to the flow. We repurpose the OpenFlow OFPT_FLOW_MOD message, which specifies an output port for a matched flow. In our context, the port id is used to represent the service, not an Ethernet port. The NetVM host maintains the local mapping between service names and VM instances. This architecture allows the SDN application to compose complex network services by populating rules across multiple hosts.

### Smart Data Planes

The current SDN approaches explore the space of having a somewhat strict separation between the control plane's decision making and the relatively simple data plane. While the current model can effectively support simple applications like a firewall, more complex applications such as intrusion detection require continuous analysis of state only available to the data plane. In this case, the controller will quickly become overloaded if it must analyze data plane information. In our experiments, we find that OpenvSwitch's throughput drops from 10 Gb/s to less than 4 Gb/s even when 10 percent of packets must be sent to the controller. Clearly, frequently sending traffic and doing expensive processing on the controller side is not a scalable solution for stateful middleboxes. We emphasize that there is a role to be played by the data plane in helping the control plane make better decisions, by providing state information that is critical in making policy decisions as well as making choices for routing of traffic flows, such as with network traffic engineering. With a "smart data plane," we can also make simple decisions based on a priori guidelines from the controller to immediately guide some network flows that would otherwise require communication with the controller resulting in both overhead and latency.

In the past there have been efforts to have a much more elaborate set of functions within the network, such as with Active Networks [15]. With the Active Network philosophy, network elements were more deeply involved in packet processing than the simple action of forwarding a packet over an appropriate port. This meant compromises had to be made on the performance achieved at these entities, as resources had to be allocated for the active network functionality. Alternatively, additional cost was incurred to have such functions resident and available at network resident entities, to be used as needed. However, with the evolution of middleboxes, this trend shifted to software-based distinct engines that were located at strategic points in the network or at best were appendages to the network forwarding elements. As technolo-

gy has progressed further, with the ability of software-based entities to perform both forwarding functions as well as policy and other middlebox functionality, our approach of a smart data plane in fact seeks to put capabilities exactly when and where they are needed, dynamically. Thus, where the functionality is primarily forwarding, the entire capacity of the engine is available. When a need arises for middlebox functionality, even for a subset of the flows, virtualization allows a new, isolated software middlebox to easily be deployed to perform this function. The performance impact therefore would be tuned to the demands of the flows and the traffic that is actually encountered at each network entity.

## Remaining Challenges and Opportunities

The combination of SDN and NFV offers new opportunities to increase the flexibility and power of the network, but there are many challenges to be overcome before this can be achieved.

**How Can Service Graphs Be Encoded by Applications that Speak to the Controller?** How can an SDN application speak of logical topologies, especially for each virtual network? We believe that this means the application has to base the description on service names and logical connections between forwarding entities as well as service functions and controller functions required for each virtual network. The controller will then have to translate that abstract specification to a physical topology. The question then arises as to whether the translation from service name to physical entity happens at the controller or at the network entity. If the network entities only understand addresses (e.g., loopback addresses), it will have to be performed at the controller. If the network entities understand names, we could go one step further and have the controller speak of service function names that are dynamically instantiated at a given network element.

**How Can SDNs Retain Control of the Network, While Exploiting the Programmability and Visibility into Packet Data of NFV Elements?** Currently, the OpenFlow protocol assumes that network elements are simple hardware switches only capable of matching a packet header to a pattern and then performing a basic action such as forwarding it out a specific port. As we move toward software-based network functions, this paradigm may be unnecessarily restrictive. A smart data plane will require a more expressive interchange with the control plane, perhaps allowing the SDN controller to provide snippets of code to software-based network elements and middleboxes. This would allow network elements to perform more complex algorithms on each flow, while still granting the SDN controller the necessary control over network behavior.

**How Does a Software-Based Network Affect the Security of Data Center and Wide Area Networks?** A more powerful and flexible network may become a ripe target for attackers since network elements have access to view and manipulate packet data, and the controller can redirect packets in arbitrary ways without input from the flow's endpoints. Ensuring the integrity of a software-based network will be an important concern so that both the controller and data plane elements can trust one another. A related challenge is that many applications encrypt traffic before sending it over the network; how will encrypted payloads limit the capabilities and benefits that a smart data plane can provide?

## Conclusion

Software-based elements are coming to both the network's control plane and data plane. While this brings the potential to build more dynamic networks at lower cost, it also introduces new challenges. We believe that the strict separation of an intelligent control plane and a simple data plane will constrain us in achieving the vision of network flexibility desired by service providers and data center operators. Instead, both SDN and NFV will have to evolve to maximize the benefits of coordinated, software-based control and data planes. Part of that evolution will be in moving a limited amount of autonomy for decision making to a "smarter data plane."

## Acknowledgments

This work was supported in part by NSF grants CNS-1422362 and CNS-1522546.

## References

[1] I. F. Akyildiza *et al.*, "A Roadmap for Traffic Engineering in SDN-OpenFlow networks," *Computer Networks*, Elsevier, 2014.
[2] N. Feamster, J. Rexford, and E. Zegura, "The Road to SDN," *Queue*, vol. 11, no. 12, Dec. 2013, pp. 20:20–40.
[3] H. Kim and N. Feamster, "Improving Network Management with Software Defined Networking," *IEEE Commun. Mag.*, vol. 51, no. 2, Feb. 2013, pp. 114–19.
[4] M. Casado *et al.*, "Virtualizing the Network Forwarding Plane," *Proc. Wksp. Programmable Routers for Extensible Services of Tomorrow*, New York, NY, 2010, pp. 8:1–6.
[5] ETSI, "Network Functions Virtualisation (NFV)," White Paper, 2014.
[6] J. Martins *et al.*, "ClickOS and the Art of Network Function Virtualization," *11th USENIX Symp. Networked Systems Design and Implementation*, Seattle, WA, 2014, pp. 459–73.
[7] V. Kotronis, X. Dimitropoulos, and B. Ager, "Outsourcing the Routing Control Logic: Better Internet Routing Based on SDN Principles," *Proc. 11th ACM Wksp. Hot Topics in Networks*, Redmond, WA, 2012, pp. 55–60.
[8] T. Koponen *et al.*, "Onix: A Distributed Control Platform for Large-Scale Production Networks," *11th USENIX Symp. Operating Systems Design and Implementation*, Vancouver, BC, Canada, 2010, pp. 1–6.
[9] S. Han *et al.*, "PacketShader: a GPU-Accelerated Software Router," *Proc. vACM SIGCOMM 2010 Conf.*, New Delhi, India, pp. 195–206.
[10] L. Rizzo, "netmap: A Novel Framework for Fast Packet I/O," *USENIX Annual Tech. Conf.*, Berkeley, CA, 2012, pp. 101–12.
[11] Intel, "Intel Data Plane Development Kit: Getting Started Guide," 2014.
[12] A. Belay *et al.*, "IX: A Protected Dataplane Operating System for High Throughput and Low Latency," *11th USENIX Symp. Operating Systems Design and Implementation*, Broomfield, CO, 2014, pp. 49–65.
[13] J. Hwang, K. K. Ramakrishnan, and T. Wood, "NetVM: High Performance and Flexible Networking Using Virtualization on Commodity Platforms," *Proc. 11th USENIX Conf. Networked Systems Design and Implementation*, Seattle, WA, 2014, pp. 445–58.
[14] M. Arumaithurai *et al.*, "Exploiting ICN for Flexible Management of Software-Defined Networks," 2014, pp. 107–16.
[15] D. L. Tennenhouse and D. J. Wetherall, "Towards an Active Network Architecture," *SIGCOMM Comp. Commun. Rev.*, vol. 37, no. 5, Oct. 2007, pp. 81–94.

## Biographies

TIMOTHY WOOD [M] (timwood@gwu.edu) is an assistant professor in the Computer Science Department at George Washington University. He received his B.S. from Rutgers University in 2005, and his M.S. and Ph.D. from the University of Massachusetts Amherst in 2009 and 2011. His Ph.D. thesis received the UMass CS Outstanding Dissertation Award, his students have voted him CS Professor of the Year, and he has won two best paper awards, a Google Faculty Research Award, and an NSF Career award. His current research focuses on improving systems software support for data centers and cloud networks.

K. K. RAMAKRISHNAN [F] (kk@cs.ucr.edu) is a professor in the Computer Science and Engineering Department of the University of California, Riverside. From 1994 until 2013, he was with AT&T, most recently as a Distinguished Member of Technical Staff at AT&T Labs-Research, Florham Park, New Jersey. Prior to 1994, he was a technical director and consulting engineer in networking at Digital Equipment Corporation. Between 2000 and 2002, he was at TeraOptic Networks, Inc., as founder and vice president. He is also an an AT&T Fellow, recognized in 2006 for his work on congestion control, traffic management, and VPN services, and for fundamental contributions on communication networks with a lasting impact on AT&T and the industry. He received an AT&T Technology Medal in 2013 for his work on Mobile Video Delivery Strategy and Optimization. His work on the "DECbit" congestion avoidance protocol received the ACM Sigcomm Test of Time Paper Award in 2006. He has published more than 200 papers and has 145 patents issued in his name. He has been on the Editorial Boards of several journals, and has served as a TPC Chair and General Chair for several networking conferences.

He received his M.S. from the Indian Institute of Science in1978, and his M.S. and Ph.D. degrees in 1981 and 1983, respectively, in computer science from the University of Maryland, College Park.

JINHO HWANG [M] (jinho@us.ibm.com) is a research staff member at IBM T. J. Watson Research Center, and works on cloud transformation centered on software-defined clouds to enable customers to adapt quickly to the heterogeneous cloud environments. He received his Ph.D. from George Washington University, Washington, DC, for his work on innovating cloud virtualization technologies regarding computing resources. He was with George Washington University from 2005 to 2006 as a visiting scholar, and with POSCO ICT R&D Center in South Korea from 2007 to 2010. He has published 30+ papers at top international conferences and journals (two best paper awards), and filed 10+ patents.

GRACE LIU (guyue@gwu.edu) received her B.S. degree in electrical engineering from Beijing University of Posts and Telecommunications, China, in 2012, and is currently pursuing her Ph.D. degree in computer science at George Washington University. Her research interests include software-defined networking and network function virtualization.

WEI ZHANG (zhangwei1984@gwu.edu) received her B.S degree in human resource management from Hebei University of Economics and Business in 2006 and her M.S. degree in computer science from Yanshan University, Hebei, China, in 2008, and is currently pursuing a Ph.D. degree in computer science at George Washington University. Her research interests include cloud computing, virtualization, and networking.