Research Statement

Timothy Wood, timwood@gwu.edu

Abstraction layers are a fundamental idea within computer science. A common thread in much of my research is exploring how abstraction layers should be designed to better support modern cloud applications: where should we have strong isolation and opaque interfaces that support loosely coupled components, and when should we puncture abstraction layers to explicitly share information across boundaries to improve performance? By reconsidering the interfaces and information flow in operating systems, virtualization layers, and communication networks, my work has provided techniques that substantially improve the performance, efficiency, reliability, and security of software systems in cloud data centers and programmable networks. Below I summarize some of the key projects that I have worked on since tenure to advance these goals.

1 High Performance, Reliable, Software-based Networking

Modern networks do far more than just transport bits. Middleboxes ranging from software switches and routers to complex intrusion prevention systems have proliferated in the network and provide important services between clients and servers. In the past, these middleboxes were deployed as custom hardware/software devices, but the shift towards virtualized networks has increased the demand for flexible, software-based solutions. By deploying network functions (NFs) as software on commodity hardware, network operators have far greater control over the services they can provide. However, running these network functions at line rates of 10-100Gbps becomes a major challenge since traditional operating systems and network stacks are not designed for this type of load.

Our work on network middleboxes began with our NetVM project [1, 2], which proposed a variety of techniques to support high performance network function chains running on commodity hardware. NetVM became the foundational platform for much of our work in this area. NetVM achieved its performance gain by rethinking the interfaces between the network card, the operating system, and the network functions running on a host. Our work leveraged kernel bypass techniques to make packets directly available to the NetVM Management framework, which in turn handled the routing of packets to one or more networks functions (NFs) running on the machine. To support chains of functions, NetVM used shared memory to provide zero copy data movement between functions running in separate VMs or containers. While this comes at a cost in terms of isolation, we demonstrated how it can improve performance by orders of magnitude. Since the time of our work, similar shared memory mechanisms have been incorporated into industry standard high performance I/O platforms such as DPDK and VPP.

Chain-Aware CPU Scheduling: While NetVM excelled at maximizing performance of middleboxes, it achieved this by dedicating CPU cores to NFs and using expensive polling operations. A seemingly small change—retaining polling for the Manager's dispatcher threads that receive packets, but changing network functions to sleep and be woken up when packets arrive—can make the system significantly more efficient by allowing multiple NFs to run on a shared CPU. However, this creates new challenges in the scheduling of network functions since existing linux scheduling policies are a poor fit for achieving both low latency and high throughput and are unaware of concepts like packet queue lengths. At its core, the challenge here was again related to abstraction layers—by bypassing the kernel's networking stack to achieve high performance user space networking, the operating system no longer had sufficient information to make good scheduling decisions.

Our work on NFVNice [3] explored the appropriate abstractions and interfaces between the user space networking framework and the kernel's scheduling mechanisms so NFs could share CPUs fairly, while still maximizing overall performance. We found that directly exposing information such as queue lengths to the kernel scheduler was too cumbersome, so we instead employed a user space driven scheduling model where the NF manager uses queue length and priority information to determine when an NF container should be woken up, and then NFs proactively yield the CPU if it appears another NF container would benefit more. This approach allows NFVNice to employ intelligent policies such as using backpressure to carefully determine which NFs to schedule based on the load on NFs in an entire chain. Backpressure allows NFVNice to avoid "wasted work" where NFs earlier in a chain are scheduled to process packets even if they might be dropped later in a chain due to lack of resources. We took the model of NFs sharing resources to the extreme with our Flurries system [4], which sought to allocate a new network function container for each network user's flows, and have also explored providing stronger resiliency for network functions through efficient state replication [5].

Pub/Sub Protocol Stacks: Our work on maximizing the performance of network function chains helped us realize that a key challenge was to improve the programming model for building functions that could work together effectively and efficiently. Due to the history of network functions being deployed as discrete, hardware-based middleboxes, functions deployed as chains were often built in isolation, each accepting a packet, processing it, and then releasing it to be analyzed by another function. While this packet in, packet out abstraction simplified deployment, it led to inefficiencies as multiple functions in a chain might need to perform redundant processing such as protocol stack analysis.

We sought to improve this with Microboxes [6], a framework for building modular, high performance protocol stacks and network functions that could be composed to efficiently support complex services. Microboxes used a publish/subscribe architecture so that packets could be selectively processed by just the portion of the protocol stack needed, and trigger events which would result in further processing based on the computed data. Microboxes uses asynchronous stack snapshots and parallelism to optimize performance while meeting consistency requirements. Our evaluation showed up to a 2x throughput improvement from stack consolidation, 20% lower latency, and 30-51% higher throughput compared to HAProxy by customizing the stack processing.

Our work on NetVM explored how bypassing traditional operating system and virtualization layer interfaces can provide substantial performance improvements, but it can come at the expense of security and resource efficiency. With NFVNice, we introduced new approaches for chain and queue-length aware user-level scheduling policies since the kernel lacked appropriate insights. Similarly, Microboxes showed how to flexibly bring modular protocol stacks out of the OS and into user space. *My work has contributed new resource management algorithms and user space managed mechanisms to improve the performance of network functions.*

2 Strong Isolation, High Density, and Low Latency Cloud Services

In the last few years my research has explored the shift away from large, centralized data centers towards much smaller, distributed "Edge Clouds". The benefit of an Edge Cloud is its close proximity to users – rather than five or six massive data centers spread around the country, an Edge deployment might have thousands of much smaller sites. Unlike a centralized data center that might be composed of 10s of thousands of servers, an Edge Cloud data center is expected to only have 10s of servers (1000X fewer resources). If we can overcome the challenge of resource scarcity in Edge clouds, then their use promises to open up a new range of real-time and latency sensitive applications such as autonomous vehicles and intelligent IoT devices, which cannot handle the high latency delay inherent in transmitting data across hundreds of miles to reach a traditional cloud data center.

In EdgeOS [7], we designed a microkernel based OS built from the ground up for running edge applications and network functions. EdgeOS incorporated what we learned from NetVM for creating high performance functions, while pairing it with OS support for "feather weight processes" (FWPs) which provided strong isolation and fast startup. This allows EdgeOS to provide strong security—the execution for each user or group of network flows is done by a unique FWP that is started and destroyed on demand. Since shared memory would violate EdgeOS's isolation properties, we instead rely on dedicated "memory movement accelerators" maintained by the kernel to efficiently copy packet data as needed between functions. EdgeOS FWPs reduce 90 percentile startup latency from 574 milliseconds with Docker containers to only 0.054 milliseconds. FWPs also allow far greater density, supporting sub millisecond latency for memcached requests even when running 600 simultaneous instances. These results demonstrate the power of carefully designed OS interfaces optimized for high performance network applications. While a custom OS may be impractical for deployments that require support for legacy code, our ongoing work tries to achieve similar goals of lightweight isolation through user level web assembly sandboxes [8].

Serverless Resource Management: Serverless computing is a software paradigm where the runtime management of applications is fully controlled by a platform provider, eliminating the need for developers to worry about deploying and scaling applications. At first glance, serverless seems like an ideal combination with Edge Clouds, since it promises the ability to "scale from zero" so that no resources are needed when workloads are light. Unfortunately, our work has found that serverless can incur high overheads both in terms of overall resource consumption and added latency. These costs arise due to the complexity of the serverless runtime and inefficient scaling and placement algorithms.

Our work on Mu [9] sought to enhance KNative, an industry standard serverless platform, to be better adapted for edge environments. Rather than assuming a cloud with practically infinite capacity for scaling,

Mu was designed for an Edge environment with limited resources and multiple tenants. Mu's autoscaler used machine learning models to predict the resource needs of each workload, integrated with a placement engine to assign resources fairly among competing functions, and enhanced the load balancing algorithms to coordinate information such as scaling events. While existing serverless platforms generally limit communication between the components responsible for autoscaling, load balancing, etc. for scalability reasons, our work demonstrated that we could provide valuable information between these management services efficiently by "piggybacking" useful data such as queue length information in request responses and other existing messages. As a result, Mu could improve fairness metrics by 2X, reduce tail latency by 62%, and reduce resource consumption by 15%.

Serverless Function Memory: Many serverless functions are written around the Extract Transform Load (ETL) pipeline approach where functions first read from a data store (E), modify the data in some way (T), and then write it back into the data store (L). This process can be repeated multiple times, for example when an image must go through a processing pipeline that rescales, converts format, and performs image recognition as a set of three functions, each of which must read and write the image from a shared data store. As a result, the read/write latency of the serverless storage system is an important factor in the overall pipeline processing speed.

Our Opportunistic FaaS Cache (OFC) [10] sought to accelerate functions by leveraging the fact that there is often spare memory allocated with functions which could potentially be used to cache commonly used data objects. This wasted memory can occur either because users overprovision function containers or because containers are generally kept alive by the platform for several minutes after a request finishes to avoid cold start delays. OFC accurately predicted how much memory a function invocation would require while taking into account request details, such as an image recognition function requiring more memory to handle a larger input image than a smaller one. It then used a model to predict whether the data is likely to be used again soon, e.g., because it is part of a chain of functions. If the data is deemed valuable, then it is written into a distributed cache maintained in the spare memory across the serverless worker nodes. OFC sought to use this cache as efficiently as possible by co-locating function executions that use data stored on a worker's cache and by proactively evicting data that won't be used again because it has reached the end of a function chain.

While NetVM achieved performance by completely bypassing the OS, our work on EdgeOS shows how an OS can provide the appropriate abstractions and interfaces for data transfer to ensure high performance and efficient network services without violating security goals. Our work in Mu and OFC emphasized the importance of using function-specific information to guide resource decision making. Rather than requiring application modifications to expose this information, we showed how to extract or embed useful information in serverless requests/responses. For OFC, this allowed us to build application models that accurately captured the memory behavior of diverse functions seeing different types of inputs. For Mu, our proxies could infer service costs and help communicate queue lengths by efficiently piggybacking this information into existing reply messages. *Overall, my work in this area has shown how redesigning interfaces—either at the OS level or in proxies between components in a distributed system—can help improve performance, isolation, and resource efficiency for edge cloud systems.*

Open Source Software, Student, and Industry Impact: While I have always made an effort to release code related to my lab's work, we took this to the next level with our OpenNetVM project, which rewrote the entire NetVM platform and adapted it to use docker containers instead of virtual machines. Funded as an NSF community research infrastructure project, I led a small army of undergraduate students to develop and maintain this platform for community use since 2015. We've had code contributions from forty different developers: about two thirds of them GW students who I mentored, and the rest external contributions from other universities and industry groups. I ran three outreach tutorials on NFV and how to use OpenNetVM at SIGCOMM, ICDCS, and Middleware. The code is available here: `https://github.com/sdnfv/openNetVM`

All of my work was accomplished by mentoring graduate and undergraduate students, and collaborating closely with industry partners at AT&T, HP Labs, VMware, Facebook, IBM, and Nokia Bell Labs. My undergraduate research students have gone on to top employers like Google, Meta, and Apple or entered Ph.D. programs. My own Ph.D. graduates have taken positions in industry at Intel and VMware, and become professors at University of Connecticut and NYU Shanghai.

References

[1] 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, Apr. 2014.

[2] W. Zhang, G. Liu, W. Zhang, N. Shah, P. Lopreiato, G. Todeschi, K. Ramakrishnan, and T. Wood, "OpenNetVM: A Platform for High Performance Network Service Chains," in *Proceedings of the 2016 ACM SIGCOMM Workshop on Hot Topics in Middleboxes and Network Function Virtualization*, HotMiddlebox 16, ACM, Aug. 2016.

[3] S. G. Kulkarni, W. Zhang, J. Hwang, S. Rajagopalan, K. K. Ramakrishnan, T. Wood, M. Arumaithurai, and X. Fu, "NFVnice: Dynamic Backpressure and Scheduling for NFV Service Chains," in *Proceedings of SIGCOMM*, SIGCOMM '17, (New York, NY, USA), pp. 71–84, ACM, 2017.

[4] W. Zhang, J. Hwang, S. Rajagopalan, K. K. Ramakrishnan, and T. Wood, "Flurries: Countless Fine-Grained NFs for Flexible Per-Flow Customization," in *ACM Co-NEXT*, Co-NEXT 16, Dec. 2016.

[5] S. G. Kulkarni, G. Liu, K. K. Ramakrishnan, M. Arumaithurai, T. Wood, and X. Fu, "REINFORCE: Achieving Efficient Failure Resiliency for Network Function Virtualization Based Services," in *Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '18, (New York, NY, USA), pp. 41–53, ACM, 2018. event-place: Heraklion, Greece.

[6] G. Liu, Y. Ren, M. Yurchenko, K. K. Ramakrishnan, and T. Wood, "Microboxes: High Performance NFV with Customizable, Asynchronous TCP Stacks and Dynamic Subscriptions," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, (New York, NY, USA), pp. 504–517, ACM, 2018. event-place: Budapest, Hungary.

[7] Y. Ren, G. Liu, V. Nitu, W. Shao, R. Kennedy, G. Parmer, T. Wood, and A. Tchana, "Fine-Grained Isolation for Scalable, Dynamic, Multi-tenant Edge Clouds," in *USENIX Annual Technical Conference*, ATC '20, pp. 927–942, 2020.

[8] X. Lyu, L. Cherkasova, R. Aitken, G. Parmer, and T. Wood, "Towards efficient processing of latency-sensitive serverless DAGs at the edge," in *Proceedings of the 5th International Workshop on Edge Systems, Analytics and Networking*, EdgeSys '22, (New York, NY, USA), pp. 49–54, Association for Computing Machinery, Apr. 2022.

[9] V. Mittal, S. Qi, R. Bhattacharya, X. Lyu, J. Li, S. G. Kulkarni, D. Li, J. Hwang, K. K. Ramakrishnan, and T. Wood, "Mu: An Efficient, Fair and Responsive Serverless Framework for Resource-Constrained Edge Clouds," in *Proceedings of the ACM Symposium on Cloud Computing*, SoCC '21, (New York, NY, USA), pp. 168–181, Association for Computing Machinery, Nov. 2021.

[10] D. Mvondo, M. Bacou, K. Nguetchouang, L. Ngale, S. Pouget, J. Kouam, R. Lachaize, J. Hwang, T. Wood, D. Hagimont, N. De Palma, B. Batchakui, and A. Tchana, "OFC: an opportunistic caching system for FaaS platforms," in *Proceedings of the Sixteenth European Conference on Computer Systems*, EuroSys '21, (New York, NY, USA), pp. 228–244, Association for Computing Machinery, Apr. 2021.