Research Statement

Joshua Reich

My research on networked systems brings together computation (end hosts) and communication (network switches), two traditionally separate areas. Hosts at the network's edge produce and consume packetized data, which network switches forward. Yet, recent trends call for reassessment of this separation between edge and network.

Cloud computing entwines computation and communication in novel ways. Cloud technologies shift data and processing seamlessly between local devices and remote datacenters, while cloud-based Virtual Machines (VMs) execute in-flight during networked migration between hosts.

In contrast, ***big data*** magnifies the coupling between computation and communication inherent in distributed data processing. To feasibly operate on petabytes of data, future systems may need to transfer similarly-sized intermediate results in tight coordination with ongoing computation.

My work integrates edge and network, enabling systems that leverage the combined strength of hosts and switches in tandem.

Technology trends have both driven, and been driven by, the growth of innovative communities such as USENIX NSDI and exciting new areas, like Software Defined Networking (SDN), for investigation. This is the space in which I work. My research explores the intellectual exchange between systems and networking in both directions, producing designs that:

- **make the edge network-aware**: more tightly coupling host OS and network stack to save host energy [3] or accelerate host storage [4];
- **use the edge to enhance the network**: utilizing hosts' local wireless interfaces to increase network capacity [2] or coverage area [5]; and harnessing host processing to programmatically control and manage the network [1].

I explore problem spaces by implementing and deploying prototypes. My systems are informed by mathematical modeling, data analysis, and programming language theory. This approach promotes a deep understanding of technology and serves as a concrete test of ideas—revealing unanticipated insights, hidden flaws, and opportunities for improvement.

My designs emphasize consideration of factors that are critical to practical adoptability. This may require contemplating how people use systems to work [3], play [2], or program [1]. It also entails accounting for hardware resource limitations [2], environmental constraints [5], and compatibility with pre-existing infrastructure [4, 2].

This combination of *exploration through implementation* and *emphasis on practicality in design* helps me generate work with wider impact. In addition to producing high-quality publications, my research has[1]:

- ***won awards*** for a running system (MobiCom/MobiHoc '07 Best Demo) [5] or codebase (NSDI '13 Community Award) [1].
- ***spun out a start-up*** (Infinio) [4].
- ***contributed to product distributed worldwide*** (Microsoft WakeProxy) [3].
- ***launched an OpenSource project*** [1].
- ***generated several technology patents*** [2, 4].

Making Edge Systems Network-Aware

For the past several decades, networked communication has played an increasingly central role in a steadily expanding universe of computerized devices—desktops, laptops, smartphones, tablets, and now, even glasses. Yet the development of network sub-system software has lagged behind that of hardware. How should we redesign and augment host operating systems to interface more cohesively with the network? Can we do things like save energy or accelerate storage and virtualization by making hosts more network-aware?

Storage Acceleration. In [4], we dramatically accelerated the rate at which new VMs could be

[1] Links at `http://www.cs.princeton.edu/~jreich`

launched from cloud storage. Existing hypervisor designs assumed each VM would be executed off a locally stored image. However, the transition to cloud-based execution of cloned VMs resulted in dozens, if not hundreds, of hypervisor machines concurrently accessing a single networked storage location. Employees arriving in the morning might idle while their virtual desktops load, or an entire e-commerce system might stall while launching new virtual servers to meet sudden demand.

We aimed to design a storage sub-system for hosts that could utilize network access to other hosts to quickly scale VM image access throughput, without degrading latency. The problem was that scalability demanded distributing content randomly, while smooth VM execution required sending blocks as they were accessed by the hypervisor.

To solve this problem, we collected VM image access patterns and used them to develop an approach that bridged the conceptual space between unstructured P2P bulk download and deterministic P2P video-streaming. This novel VMTorrent design balanced the collective's need for scalable throughput with each individual host's need for low latency access. In hardware tests of up to 100 servers, our VMTorrent prototype outperformed state-of-the-art solutions by orders of magnitude.

Energy Efficiency. In [3], we addressed widespread power waste by enterprise hosts. OS energy-saving mechanism designers assumed users would access computers through locally attached peripherals. The shift to remote interaction broke this assumption. As a result, enterprise users disabled OS power-saving features so their machines would remain remotely accessible. In response, the Wake-on-LAN (WoL) standard defined special packets whose receipt would wake the host: a mechanism broken shortly thereafter by new standards which blocked WoL packets from traveling across Local Area Network boundaries.

What we needed was a backwards-compatible energy-saving architecture that accommodated both the needs of human users and the economic realities of the enterprise. Our solution had to be inexpensive to install and maintain; ensure users remote access with little or no perceptible delay; and still produce significant energy savings.

We collected user data to determine how much energy might be saved at various points in the design space. What we discovered was that a relatively lightweight design, using edge and network-based mechanisms in tandem, should provide significant savings. On the OS side, we tied together energy-saving and network subsystems, so hosts could transmit a handoff message immediately before transitioning to energy-saving states. On the network side, we leveraged knowledge of Ethernet switching logic and the Address Resolution Protocol to redirect future traffic towards the host taking the handoff (who would wake the sleeping host using WoL as needed) in a completely backwards compatible manner. Thus network hosts could transition to energy-saving mode, while remaining continuously available for network access. We deployed our prototype on over 50 user workstations, for half-a-year at Microsoft Research Redmond. Our measurements demonstrated that this approach did, in fact, realize the majority of theoretically attainable power-savings.

Edge Systems that Enhance the Network

Network infrastructure, while far more optimized than edge hosts for forwarding packetized data, is relatively limited in many respects. Limitations include capacity and range in wireless networks, and computational power in both wired and wireless networks. How might we exploit hosts with local wireless interfaces to serve a *dual-role*, as both edge consumers and network forwarders? How can we leverage edge computation power to provide platforms that empower human programmers to effectively control and manage the network?

Augmenting Network Capacity. In [2], we designed protocols to increase network capacity by disseminating high-bandwidth content (e.g., videos) during opportunistic meetings between mobile hosts (e.g., smartphones).

What made this particularly difficult was that a practical solution must consider both host storage limitations and how human users grow impatient while waiting for requested content.

To this end, we defined an objective function that accounted for the effects of *user impatience*: a monotonically decreasing function representing the loss of utility as users wait for requested content. The value taken by this objective function depended on the allocation of content to each device's finite local cache. We demonstrated that the optimal allocation maximizing this objective function can be computed efficiently. Further, under simplified assumptions, we showed that the corresponding optimal cache allocation is known in closed form for a general class of delay-utility functions. Finally, we developed completely distributed protocols that provably converge to this optimal allocation.

Extending Network Range. In [5], we developed protocols that enabled mobile robots to form a self-spreading wireless mesh network. Such networks could extend the area covered beyond that of fixed wireless infrastructure.

To realize this potential, we needed control-plane protocols that would enable the robots to spread, while ensuring the physical layer network remained connected. Further, we wanted our protocol to work even in challenging environments, containing obstacles to movement, wireless channel propagation, and/or robot localization. This provided a serious challenge. Unlike previous work, our connectivity algorithm could not rely on deterministic wireless connectivity models or knowledge of spatial coordinates.

Instead, we showed it was possible to use knowledge of the current network connectivity topology to predict the likelihood of partition in the near future. Our protocol was scalable, requiring each robot to learn the local (2-hop) network topology, and independently decide whether to continue movement or freeze in place. We implemented our protocol on our mobile robotic testbed—a first in this research area, to the best of our knowledge. Testbed evaluation demonstrated our protocol maintained connectivity over 99% of the time in realistic environments, while still enabling significant coverage extension.

Modular SDN Programming Platforms. In [1], we developed programmer-facing abstractions needed to build truly modular SDN applications, and efficient techniques for implementing these abstractions. Before SDN, network packet processing was determined by an ad hoc process which cobbled together various decentralized routing protocols. The emergence of OpenFlow allowed the network data-plane to be programmed by hosts. This enabled direct, albeit low level, control of the network, tightly tied to hardware primitives.

The key challenges we faced were (a) designing SDN programming abstractions that programmers would find both intuitive and powerful, and (b) implementing a runtime that realized those programs using low-level hardware-centric primitives, applied asynchronously across the network.

Informed by programming language theory, our Pyretic SDN controller platform was the first that enabled programmers to generate almost any packet processing behavior realizable via OpenFlow, using only a small and intuitive set of basic primitives and combinators. Further, Pyretic approached network virtualization in a new way, providing a proof-of-concept network hypervisor that was simply an ordinary Pyretic application.

To provide maximum accessibility to the systems and networking communities, we both embedded the Pyretic language and implemented the Pyretic runtime in a familiar general purpose language, Python. We released Pyretic to the research community under an open source license, and its subsequent development has been informed by a growing community userbase. Pyretic has been used by roughly 1000 students taking Coursera's open online SDN course, and dozens more in University courses and academic research projects. With each new release, Pyretic has reduced controller load, processing an ever-higher proportion of packets through the data-plane. We are refining the Pyretic language, building a robust engine for querying network traffic, adding new features to the runtime, and using Pyretic to explore a variety of open questions in controller design.

Future Work

Going forward, I intend to continue unifying edge and network. One branch of my future work will continue the examination, begun with Pyretic, of *network controller platform design*. This study will include exploring incorporation of dual-roled hosts to provide network data-plane functionality. The other branch of my research will focus on *coupling programmatic network control and host-based applications* to produce better distributed systems.

Network Controller Platform Design

The Pyretic platform provides a rich laboratory for research on a wide cross-section of general SDN programming platform challenges. Among the questions that I intend to investigate are:

Distributed Controllers. The current Pyretic runtime executes on a single host. What happens when that host fails or becomes a bottleneck? These are the classical distributed systems problems of *high-availability* and *scale-out*, encountered in a new setting; a setting in which classical solutions may not apply. I intend to extend integration of edge and network, by adapting techniques from distributed systems to build fully-distributed network control-planes.

Packet Processing at the Edge. The functionality offered by SDN programming platforms, including Pyretic, primarily targets what OpenFlow's switch-bound programming interface can provide. This restricts how packets can be matched, limits what can be done to them, and fails to provide access to stateful packet-processing mechanisms such as rate-limiters. While I am currently exploring abstractions for utilizing switch hardware features beyond what OpenFlow provides, even full utilization of switch features will support only relatively restricted data-plane packet-processing functionality. Might we be able to provide a significantly more flexible, feature-rich, and efficient data-plane by incorporating edge-based packet-processing? Can we do so without compromising on latency, throughput, or cost? What abstractions might make such systems usable?

Dynamic Policy. While we have developed robust abstractions for specifying the data-plane forwarding behavior desired at any given instant (*static policy*), our constructs for specifying how we want data-plane forwarding behavior to change over time (*dynamic policy*) are less mature. What are the right abstractions for specifying dynamic policy? How do we integrate additional sources of information into a dynamic policy? What guarantees can we give about dynamic policies? Can we implement these policies on switches in a way that is both correct and efficient? And what do "correctness" and "efficiency" even mean in this context?

Coupling Network Control & Edge Systems

I intend to study unification of edge and network by investigating systems such as **network-native key-value stores** that tightly couple network control with host-based systems.

The key-value store provides a powerful paradigm for organizing and distributing storage capacity in the datacenter. However, traditional approaches implement such stores entirely at the network edge. This has at least two drawbacks:

1. Host servers are entirely responsible for maintaining an overlay mapping from data item keys to network addresses (IP or MAC) and for forwarding requests across that overlay (which may require traversing multiple overlay hops). Thus, each key-value store must implement its own routing and forwarding logic—burdening servers with additional load.
2. The underlying network may be used inefficiently by the overlay, resulting in (a) duplicate packets, (b) sent across unnecessarily long paths, (c) traversing unevenly utilized links.

Can we do better, building a key-value store capable of serving as datacenter-scale distributed shared memory? I am intrigued by the possibility that both of these drawbacks might be addressed by combining content-centric networking concepts, like those used in my VMTorrent work, with data-plane forwarding programmability. By embedding data names in routable packet header fields (e.g., source IP) and programming switch forwarding tables intelligently, might we implement a network-native key-value store that reduces server load while concurrently improving network resource utilization?

References

[1] C. Monsanto, J. Reich, N. Foster, J. Rexford, and D. Walker. Composing Software-Defined Networks. In *USENIX NSDI*, pages 1–13, April 2013.

[2] J. Reich and A. Chaintreau. The Age of Impatience: Optimal Replication Schemes for Opportunistic Networks. In *ACM CoNEXT*, pages 85–96, December 2009.

[3] J. Reich, A. Kansal, M. Gorackzo, and J. Padhye. Sleepless in Seattle No Longer. In *USENIX ATC*, June 2010.

[4] J. Reich, O. Laadan, E. Brosh, A. Sherman, V. Misra, J. Nieh, and D. Rubenstein. VMTorrent: Scalable P2P Virtual Machine Streaming. In *ACM CoNEXT*, pages 289–300, December 2012.

[5] J. Reich, V. Misra, D. Rubenstein, and G. Zussman. Connectivity Maintenance in Mobile Wireless Networks via Constrained Mobility. In *IEEE Infocom*, pages 927–935, April 2011.