Research Statement

Radhika Mittal

1 Overview

The vast majority of our day-to-day applications run over computer networks, both across wide-area Internet and within datacenters, leading to an intense emphasis on network performance. This performance not only affects user satisfaction, but also has a significant impact on the revenues generated by online service providers such as Google, Microsoft and Facebook [6]. As a result, there have been many proposals for adding sophisticated features in network switches, including various scheduling policies [8, 11, 12, 25, 27, 31, 34], explicit rate signalling [13, 16], mechanisms to achieve losslessness [7], among others. While these features promise improved performance, they often come at a high cost, either requiring specialized network switching hardware or greatly complicating network management. Therefore, rather than continuing to add new features to the network in an ad-hoc manner, I believe we should adopt a more principled approach: First of all, echoing the classical end-to-end principle [28], complexity should be added into the network only when the required goals cannot be met solely from the end-points. Secondly, in such cases where it is necessary to add network support, we should look for *universal* solutions, where a single mechanism implemented in network switches can handle a wide range of requirements.[1] In many cases, the networking community has not explored the former before adding new features to the network, while the notion of universality had not even been formally defined. My work, therefore, focuses on the following two questions:

(1) To what extent can we use the commonly deployed network infrastructure to meet various performance requirements? Here, I have looked at congestion control algorithms (that determine the rate at which data is injected into the network), both in the context of wide-area [21] and datacenter networks [18, 23], showing that we can achieve near-optimal performance using commodity switches. More recently, I have worked on redesigning RDMA NICs to eliminate their reliance on complicated in-network mechanisms for loss avoidance [19]. Mellanox, one of the leading RDMA NIC vendors, will be adopting a version of my design in their next release.

(2) For cases where additional support is required in network switches, can we devise a single universal mechanism? Here, I have focused on packet scheduling algorithms (that determine the order in which packets are transmitted in the network), examining whether we can have a *universal* packet scheduling algorithm that can mimic all others. I showed, both theoretically and practically, that there is an almost-universal packet scheduling algorithm that is relatively easy to implement [24].

I begin with some relevant background (§2) for readers unfamiliar with computer networking, then describe my past work (§3 and §4), and end with a discussion of my future agenda (§5).

2 Relevant Background

The following is a very simplified view of how data is transferred across a computer network: Computers, phones, datacenter-servers, and other such *endhost* devices are connected to one another by a *network* of *switches*. The endhost's operating system typically implements a software network stack which transmits/receives data over a hardware device (called a *network interface card* or NIC) that interfaces with the physical network links. The data to be transferred is split into multiple *packets*. Packets belonging to the same connection make up a *flow*. The data payload in the packet is encapsulated within *packet headers* that contain the meta-data needed by the network to process and forward the packet. Switches in the network forward the packets along an appropriate route to the destination. Switches can only forward the packets at a finite rate determined by the *bandwidth* of their outgoing links. Therefore, the packets that arrive at a switch while it is busy forwarding another packet are *queued* in a switch *buffer*. When the buffer becomes full a packet is *dropped*. Most network communication happens over *reliable* connections, where the receiver acknowledges the data it receives and the sender needs to retransmit the dropped (or *lost*) packets. The time interval between a sender sending a packet and then receiving the corresponding acknowledgement is called a *round-trip time* or an RTT.

[1] Note that the quest for universality is complementary to recent developments in programmable switching hardware [9, 10], which I discuss in more detail in §5.

Network performance primarily depends on two main classes of algorithms: (i) *Congestion control algorithms* that run at the endhosts and control the rate at which data packets are injected into the network. Typical congestion control algorithms use a feedback loop based on congestion signals such as packet loss, round trip times, or explicit signals set by the switches. (ii) *Scheduling algorithms* that run inside the switches and determine the order in which the queued up packets in the buffer are to be transmitted on the outgoing link. While many scheduling algorithms have been proposed in the past [8, 11, 12, 25, 27, 31, 34], low-end commodity switches today only support a fixed number (typically eight) of priority queues, with simple first-in-first-out (FIFO) scheduling within each queue.

3 Improving Network Performance Using Existing Infrastructure

I will now elaborate on my past work that uses commonly deployed network infrastructure to improve performance.

RC3: Recursively Cautious Congestion Control for wide-area networks. A good congestion control algorithm needs to satisfy two conflicting goals (i) efficient use of network bandwidth, and (ii) not harming other flows that are sharing the network. The former requires a more aggressive behavior, while the latter requires a more cautious one. TCP, the most widely deployed congestion control algorithm, starts cautiously, sending a very small amount of data at first and exponentially increasing its sending rate after every round-trip, until the flow starts experiencing packet drops. This cautious ramp-up leads to significant wastage of network capacity, especially in wide-area networks with large RTTs and increasing bandwidths. However, because it uses only a single mechanism to achieve the two conflicting goals mentioned above, TCP must use cautious probing. RC3 decouples these two goals by *sending additional data aggressively* (to use all of the available network capacity), but *at a lower priority* than the regular traffic (so as not to harm the other flows in the network). For the latter, it leverages the priority scheduling support present in almost all currently deployed switches. This reduces the average flow completion by 40% when compared to regular TCP, with strongest gains (of over 70%) seen in medium to large sized flows. It also performs better than prior proposals that aim to eliminate TCP's cautious probing phase by using complex switch support [13, 16]. RC3 appeared in ACM HotNets 2013 [20] and USENIX NSDI 2014 [21].

TIMELY: RTT-based congestion control for the datacenter. The use of packet drops as the sole congestion signal results in high queueing delay, since the sending rate is reduced only after the switch buffer gets full. This negatively impacts end-to-end performance, particularly for short interactive flows in datacenters with stringent latency requirements ($\ll$msec). It is, therefore, desirable to react quickly to congestion as soon as switch buffer queues begin to grow. This makes RTT a natural choice for a congestion signal, which provides fine-grained information about the queuing within the network without requiring any switch support. However, it was considered to be too noisy to be used effectively in datacenters (where RTTs can be of the order of a few microseconds). As a result, the datacenter community moved towards using explicit congestion notification (ECN), which requires the switches to set a bit in the packet headers when the queuing exceeds a certain (carefully tuned) threshold. While on an internship at Google, I learned about recent advancements in the NIC hardware that enable accurate RTT measurements at microsecond granularity; leveraging this technology, we developed *TIMELY, the first RTT-based congestion control algorithm for datacenters*. TIMELY updates the sending rate based on the RTT gradient, to achieve ultra-low latency while maintaining near-optimal throughput. TIMELY was published in ACM SIGCOMM 2015 [23].

IRN: Revisiting Network Support for RDMA. A particularly relevant use-case for TIMELY was for data transfer over RDMA (Remote Direct Memory Access). Current datacenter requirements of low latency, high throughput and negligible CPU utilization can no longer be met by the software packet processing stack in the operating systems. Therefore, leading enterprises are moving towards using RDMA, where packet processing is offloaded to specialized NICs. RoCE NICs, used for deploying RDMA over the Ethernet fabric in datacenters, require a lossless network. This losslessness is achieved by enabling Priority Flow Control (PFC) [7], a push-back mechanism that avoids congestion drops. However, PFC requires very careful switch configuration and leads to myriad performance issues including unfairness, head-of-the-line blocking, congestion spreading and deadlocks [14, 15, 30]. I experienced these PFC issues first-hand during my Google internship, which made me question the need for losslessness in the first place. I contacted Mellanox, the leading RDMA NIC vendor, and, working in collaboration with them, showed that by making two relatively straight-forward changes to the RoCE NIC design, we can *eliminate the need for losslessness*, while improving performance and increasing robustness. Mellanox will be implementing a version of this new *improved RoCE NIC* (IRN) design in their next release. I also discussed the IRN design with Intel, who are now evaluating it for

implementation. More details about this work can be found here [19].

Factor Analysis for Datacenter Congestion Control. Recent years have seen a continuous stream of papers on datacenter congestion control mechanisms. While these solutions differ along many dimensions – such as how to set the endhost sending rates, and how to schedule packets in the switches – there was no systematic understanding of what is the most essential factor in achieving good performance. This motivated me to perform a *factor analysis* for datacenter congestion control. In collaboration with Aisha Mushtaq, a new PhD student in my lab, I showed that SRPT scheduling (where packets are scheduled in the order of shortest remaining flow size) achieves near-optimal average flow completion times (as also shown in [8]), while being most robust to different endhost behaviors (which was not shown before). However, SRPT cannot be deployed using today's commodity switches. Nonetheless, as shown by us and others, its behavior can be approximated using the small number of priority queues available in most switches. Our results have drawn interest from commercial vendors such as Intel, Huawei and Mellanox, who were completely unaware that one could use priority scheduling to approximate SRPT (and thus achieve significantly better performance than FIFO). More details about this work can be found here [18].

4 Universal Packet Scheduling

Scheduling algorithms play a key role in achieving various performance goals. As a result, there is a large and active research literature on novel packet scheduling algorithms. These include mechanisms for achieving fairness [12, 25, 31, 34], reducing tail latency [11], meeting deadlines [29], minimizing flow completion times [8, 18, 21] (as discussed in §3), among many others. Each of these scheduling algorithms must be implemented in the switch hardware, making it difficult to support *different scheduling algorithms* for *different requirements* [32]. This led me to ask the following question: do we really need different scheduling algorithms for different requirements, or can we instead have a *universal packet scheduling* algorithm (hereafter, UPS)? I explored this question using two perspectives:

Theoretical Perspective. The first step in this research was to introduce and formally define the notion of universality, which had not been previously considered in the packet scheduling literature: a packet scheduling algorithm is universal if it can *replay* the schedule (given by the set of times at which packets arrive to and exit from the network) produced by any other scheduling algorithm, where replay means that all packets achieve exit times no worse than in the given original schedule. I proved that while we can never have a universal packet scheduling algorithm, the classical Least Slack Time First (LSTF) [17] algorithm comes as close as any scheduling algorithm to achieving universality (in a sense made precise in [24]). With LSTF, each packet carries a slack value in its header, which is updated and used by the switch for scheduling. This slack value can be appropriately initialized at the network edge (or ingress) to achieve different outcomes.

Practical Perspective. The theoretical definition of UPS assumed some knowledge of the original schedule that it is trying to replay, which is not available in practice. However, in practice one would want a UPS to match the performance of the best known scheduling algorithm for a given performance objective. I showed that LSTF achieves performance comparable to the state-of-the-art for three different performance objectives (minimizing average flow completion times, minimizing tail latency, and achieving per-flow fairness). I also showed how LSTF can be used to implement Active Queue Management (AQM) from the network edge.

This work, which appeared in ACM HotNets 2015 [22] and USENIX NSDI 2016 [24], was a first step towards a paradigm shift in how we think about the well-established problem of packet scheduling, implying that we might not need many different scheduling and queue management algorithms implemented in the core of the network, and can instead just use LSTF in the core with varying header initializations at the network edge.

5 Future Research

Extending UPS beyond imitating packet scheduling algorithms. My work on UPS opens up several interesting research questions, some of which are discussed below:
(1) LSTF can imitate different scheduling algorithms that individually perform well on a given performance metric (such as average FCT, fairness and tail packet delay). However, most networks are shared by many different applications, each with their own performance goals. *The question then is whether we can use LSTF to meet this combination of goals, and is there any other scheduling algorithm that would perform better?*

(2) The notion of UPS only deals with how packets are scheduled on point-to-point links. *Can we extend the notion of universality to how packets are scheduled across resources in other entities, such as middleboxes and wireless or cellular channels?*
(3) UPS only looked at packet-level policies. *Can we extend the notion of universal scheduling to task-level policies?* I am particularly interested in exploring this in the context of serverless applications [1, 2, 4, 5], whose performance requirements, such as fairness across users or small aggregate completion times, can be considered analogous to those of packets and flows, but would lead to their own set of interesting challenges such as scheduling across multiple resources (compute, memory and network) and across multiple dimensions (both *when* and *where* to launch a task).

Clean-slate network stack offload. While RDMA is an obvious improvement over the more traditional network processing stack, I would like to explore whether RDMA, with all its bells and whistles, was the *right* choice for network processing offload, or was it only the most *convenient* one (since RDMA NICs were available)? My experience with IRN suggests that it might be possible to achieve the primary benefits provided by RDMA with significantly simpler mechanisms. I, therefore, plan on devising a clean-slate approach for offloading network processing (which can encompass both fixed function and programmable NIC designs [3, 26]).

Intelligence in the edge versus in the core: There is a long-standing battle about where to place the intelligence needed for a network functionality: should it go only in the network edge or should it be deployed more widely in the network core? However, this debate has mostly been informal (though often heated). My work on UPS has led to a clean theoretical formulation whose fundamental question can be more generally stated as: for which kinds of functionality can intelligence in the edge *effectively simulate* a set of specialized mechanisms deployed in the core? UPS results showed that we can effectively simulate packet scheduling and AQM policies. But there are many other functionalities where we do not yet have a clear answer, such as network monitoring, load-balancing, and support for information-centric networking, among others.

Packet processing as a service: The advent of programmable switching hardware [9, 10] will give network operators greater flexibility in what functionality is implemented in network switches (whether at the edge or in the core). However, little thought has been given to how this flexibility might be exposed to large customers who may want specialized processing for their own traffic as it is carried over the public Internet. I hope to investigate what kinds of interfaces can ISPs expose that would allow customers to monitor their own traffic (e.g., detecting where delays are occurring), or implement a scheduling policy within their own traffic (e.g., Netflix may want to prioritize certain downloads over others in the network, while still only getting its fair share of network service), or otherwise improve their performance through custom packet processing. This question is somewhat reminiscent of *active networking* [33], but is now framed within the context of a specific programming model whose performance is well-understood, making the idea far more feasible.

Overall, I enjoy questioning the conventional wisdom about network design. This has led me to learn and apply a variety of techniques for my past work, ranging from simulations, to systems implementation, to theoretical proofs and hardware design. I believe that the insights I gained from my work so far and the useful collaborations I built along the way, combined with my passion to learn new things, will help shape my future agenda.

References

[1] Amazon AWS Lambda. `http://aws.amazon.com/lambda/`.
[2] Microsoft Azure Functions. `http://azure.microsoft.com/en-us/services/functions/`.
[3] Cavium Intelligent NICs. `http://www.cavium.com/Intelligent_Network_Adapters_NIC4E.html`.
[4] Google Cloud Functions. `http://cloud.google.com/functions/`.
[5] Apache OpenWhisk. `http://openwhisk.incubator.apache.org/`.
[6] The Cost of Latency. `http://perspectives.mvdirona.com/2009/10/the-cost-of-latency/`.
[7] IEEE. 802.11Qbb. Priority based flow control, 2011.
[8] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker. pFabric: Minimal Near-optimal Datacenter Transport. In *Proc. ACM SIGCOMM*, 2013.
[9] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz. Forwarding Metamorphosis: Fast Programmable Match-action Processing in Hardware for SDN. In *Proc. ACM SIGCOMM*, 2013.

[10] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker. P4: Programming Protocol-independent Packet Processors. *ACM SIGCOMM Computer Communication Review*, 2014.

[11] D. D. Clark, S. Shenker, and L. Zhang. Supporting Real-time Applications in an Integrated Services Packet Network: Architecture and Mechanism. In *Proc. ACM SIGCOMM*, 1992.

[12] A. Demers, S. Keshav, and S. Shenker. Analysis and Simulation of a Fair Queueing Algorithm. In *Proc. ACM SIGCOMM*, 1989.

[13] N. Dukkipati and N. McKeown. Why Flow-Completion Time is the Right Metric for Congestion Control. *ACM SIGCOMM Computer Communication Review*, 2006.

[14] C. Guo, H. Wu, Z. Deng, G. Soni, J. Ye, J. Padhye, and M. Lipshteyn. RDMA over Commodity Ethernet at Scale. In *Proc. ACM SIGCOMM*, 2016.

[15] S. Hu, Y. Zhu, P. Cheng, C. Guo, K. Tan, J. Padhye, and K. Chen. Deadlocks in Datacenter Networks: Why Do They Form, and How to Avoid Them. In *Proc. ACM HotNets*, 2016.

[16] D. Katabi, M. Handley, and C. Rohrs. Congestion Control for High Bandwidth-Delay Product Networks. In *Proc. ACM SIGCOMM*, 2002.

[17] J. Y.-T. Leung. A new algorithm for scheduling periodic, real-time tasks. *Algorithmica*, 1989.

[18] A. Mushtaq, **R. Mittal**, J. McCauley, M. Alizadeh, S. Ratnasamy, and S. Shenker. Datacenter Congestion Control: Identifying what is essential and making it practical. *In submission*. http://people.eecs.berkeley.edu/ radhika/adsrpt.pdf . [pdf].

[19] **R. Mittal**, A. Shpiner, A. Panda, E. Zahavi, A. Krishnamurthy, S. Ratnasamy, and S. Shenker. Revisiting Network Support for RDMA. *In submission*. http://people.eecs.berkeley.edu/ radhika/irn.pdf . [pdf].

[20] **R. Mittal**, J. Sherry, S. Ratnasamy, and S. Shenker. How to Improve your Network Performance by Asking your Provider for Worse Service. In *Proc. ACM HotNets*, 2013. [pdf].

[21] **R. Mittal**, J. Sherry, S. Ratnasamy, and S. Shenker. Recursively Cautious Congestion Control. In *Proc. USENIX NSDI*, 2014. [pdf].

[22] **R. Mittal**, R. Agarwal, S. Ratnasamy, and S. Shenker. Universal Packet Scheduling. In *Proc. ACM HotNets*, 2015. [pdf].

[23] **R. Mittal**, V. Lam, N. Dukkipati, E. Blem, H. Wassel, M. Ghobadi, A. Vahdat, Y. Wang, D. Wetherall, and D. Zats. TIMELY: RTT-based Congestion Control for the Datacenter. In *Proc. ACM SIGCOMM*, 2015. [pdf].

[24] **R. Mittal**, R. Agarwal, S. Ratnasamy, and S. Shenker. Universal Packet Scheduling. In *Proc. USENIX NSDI*, 2016. [pdf].

[25] A. K. Parekh and R. G. Gallager. A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single-node Case. *IEEE/ACM Trans. Netw.*, 1993.

[26] A. Putnam, A. M. Caulfield, E. S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. P. Gopal, J. Gray, M. Haselman, S. Hauck, S. Heil, A. Hormati, J.-Y. Kim, S. Lanka, J. Larus, E. Peterson, S. Pope, A. Smith, J. Thong, P. Y. Xiao, and D. Burger. A reconfigurable fabric for accelerating large-scale datacenter services. In *Proc. ACM ISCA*, 2014.

[27] S. Blake and D. Black and M. Carlson and E. Davies and Z. Wang and W. Weiss. An Architecture for Differentiated Services. RFC 2475, 1998.

[28] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in system design. *ACM Trans. Comput. Syst.*, 1984.

[29] H. Sariowan, R. L. Cruz, and G. C. Polyzos. Sced: A generalized scheduling policy for guaranteeing quality-of-service. *IEEE/ACM Trans. Netw.*, 1999.

[30] A. Shpiner, E. Zahavi, V. Zdornov, T. Anker, and M. Kadosh. Unlocking Credit Loop Deadlocks. In *Proc. ACM HotNets*, 2016.

[31] M. Shreedhar and G. Varghese. Efficient Fair Queueing Using Deficit Round Robin. In *Proc. ACM SIGCOMM*, 1995.

[32] A. Sivaraman, K. Winstein, S. Subramanian, and H. Balakrishnan. No Silver Bullet: Extending SDN to the Data Plane. In *Proc. ACM HotNets*, 2013.

[33] D. L. Tennenhouse, J. M. Smith, W. D. Sincoskie, D. J. Wetherall, and G. J. Minden. A survey of active network research. *IEEE Communications Magazine*, 1997.

[34] L. Zhang. Virtual Clock: A New Traffic Control Algorithm for Packet Switching Networks. In *Proc. ACM SIGCOMM*, 1990.