No Silver Bullet: Extending SDN to the Data Plane

Anirudh Sivaraman, Keith Winstein, Suvinay Subramanian, and Hari Balakrishnan
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology, Cambridge, Mass.
{anirudh, keithw, suvinay, hari}@mit.edu

ABSTRACT

The data plane is in a continuous state of flux. Every few months, researchers publish the design of a new high-performance queueing or scheduling scheme that runs inside the network fabric. Many such schemes have been queen for a day, only to be surpassed soon after as methods — or evaluation metrics — evolve.

The lesson, in our view: there will never be a conclusive victor to govern queue management and scheduling inside network hardware. We provide quantitative evidence by demonstrating bidirectional cyclic preferences among three popular contemporary AQM and scheduling configurations.

We argue that the way forward requires carefully extending Software-Defined Networking to control the fast-path scheduling and queueing behavior of a switch. To this end, we propose adding a small FPGA to switches. We have synthesized, placed, and routed hardware implementations of CoDel and RED. These schemes require only a few thousand FPGA "slices" to run at 10 Gbps or more — a minuscule fraction of current low-end FPGAs — demonstrating the feasibility and economy of our approach.

1. INTRODUCTION

In packet-switched networks, each switch makes two important decisions on a per-packet basis:

> **Queue-management:** How long can the queue grow, and which packet should be dropped if it grows too long?
>
> **Scheduling:** When an outgoing link is free, which packet should be sent next?

Over the past three decades of research and practice in Internet resource management, researchers have produced a long and celebrated series of answers to these two questions — starting with WFQ [9] and SFQ [22], to RED [13] and BLUE [11], to ECN [12], XCP [18] and RCP [27], to DCTCP [3], PDQ [16], CoDel [24], and pFabric [4].

As these systems have evolved and multiplied, we think there has been a tacit belief in the existence of a final answer to these two questions. For example, since the development of RED in the early 1990s, there has been a quest for the "one-size-fits-all" ideal active queue management (AQM) algorithm [13, 11, 25, 20, 24]. Numerous proposals exist for switch scheduling, and the consensus today appears to be to implement some variant of weighted fair queueing [9], augmented with some queues for higher priority traffic.

In our view, the prospect of a final answer is a mirage. The answers depend crucially on what the applications running at the endpoints want. For example, the requirements of a computer backup application (throughput) are different from an interactive web site (page load times), which in turn differ from a videoconference (some combination of throughput and per-packet delay). In fact, even when the applications want the same thing (e.g., throughput), the answers depend on how the endpoint's transport protocol reacts to congestion signals; the right answer for TCP NewReno may well be different from TCP CUBIC [14] or Compound TCP [28].

We believe that the search for a "one-size-fits-all" solution to these questions forces network implementers to make choices prematurely. There are applications for which an AQM such as CoDel [24] may be the right answer (at least compared with existing alternatives), and there are applications for which a "bufferbloated" drop-tail router is better than the alternatives (e.g., transferring a large file reliably).

Moreover, new network applications are certain to be developed, and new queueing strategies, scheduling techniques, and endpoint control protocols will be invented. New applications will require different objectives, e.g., a different trade-off between throughput and delay, an emphasis on throughput or delay variation above other metrics, and so on.

How should the switch designer respond to these concerns about the diversity of objectives and mechanisms? Traditionally, software routers support such configurability while hardware routers — even those that support OpenFlow or other realizations of Software-Defined Networking — don't.

We propose a middle ground: carefully extending the ideas of SDN to the data plane, to allow network operators to align queueing and scheduling behavior with application objectives. To achieve this, our proposal adds a small FPGA to the fast-path of a switch, with well-defined interfaces to packet queues on the switch.

This paper makes two concrete contributions: (1) a clear demonstration that bidirectional cyclic preferences exist among queueing and scheduling schemes today, suggesting that the quest for a "best" such scheme is futile, and (2) an evaluation of our successful synthesis of two popular queue-management schemes for an FPGA, suggesting our approach is feasible and economical in practice.

Our hope is to build support for the position that we should not be searching for a silver bullet for router queueing and scheduling; instead, switches should support multiple schemes by investing in a small amount of reconfigurable hardware, extending the ideas of SDN to the data plane in a realistic and helpful manner.

2. RELATED WORK

2.1 Active Networks

In the mid 90s, research on active networks [29] argued for fully programmable networks by proposing that switches execute code contained in packets. The biggest criticisms of active networks were security and performance: allowing arbitrary code to run on a switch could significantly slow down the data plane, or compromise the switch if the code was malicious. We argue, instead, for reconfigurable hardware accessible to administrators, with a restricted interface that allows operator-specified code only to influence queueing and scheduling decisions at the switch.

2.2 Hardware Router Platforms

Research hardware platforms such as SwitchBlade [6] NetFPGA [21] and Chimpp [26] are both programmable and provide high performance. However, they lack the port density to be commercially viable. Our proposal is inspired by these systems in that it adds an FPGA to a switch. We show in Section §4.4 that this idea is economically feasible, and practical at high line rates. This allows flexibility in queueing and scheduling algorithms, without degrading performance.

2.3 Software Routers

Software routers such as Click [19] are routinely used by researchers to evaluate new scheduling and queueing algorithms. With time, software routers have improved in performance by exploiting parallelism (RouteBricks [10]) or GPU offloading ([15]). Although they are easy to program, compared with a hardware switch with the same port count and non-blocking bandwidth, a software implementation will be more expensive, if feasible at all. For economic and practical reasons, high performance switches invariably use dedicated hardware.

2.4 Software-Defined Networks

Software-Defined Networks [23, 7] (SDN), and OpenFlow [23], have eased the management of enterprise networks in the last few years. These systems, largely directed at influencing or centralizing control-plane decisions, have allowed network operators to configure their production systems [17] more efficiently and realize new behaviors not otherwise available from commercial vendors. Several switch vendors today support OpenFlow, which is often hailed as the antidote to architectural ossification because it allows new routing protocols to be tested in a production environment.

Even as the control plane has become more flexible, the obstacles toward widespread deployment of XCP [18], RCP [27], or CoDel [24] speak to the consequences of today's data-plane rigidity. We propose to extend SDN's flexibility to cover queueing and scheduling decisions made in the data plane, without giving up the "fast-path" benefits that come from a hardware implementation.

3. NO SILVER BULLET

To support our position that there is no silver bullet for scheduling and queue management in a switch, we evaluate three contemporary gateway configurations to demonstrate how they can give rise to arbitrary and contradictory preferences.

A) **CoDel+FCFS**: CoDel running on a single shared first-come, first-served queue for all traffic.

B) **CoDel+FQ**: A separate queue for each flow with an independent instance of CoDel running on each queue. Queues are serviced using fair queueing.

C) **Bufferbloat+FQ**: A separate queue for each flow with a deep buffer that doesn't drop any packets. Queues are serviced using fair queueing.

We will demonstrate that among these three configurations, depending on the application's preference, $A > B$, $B > C$, and $C > A$. Furthermore, $B > A$, $C > B$, and $A > C$!

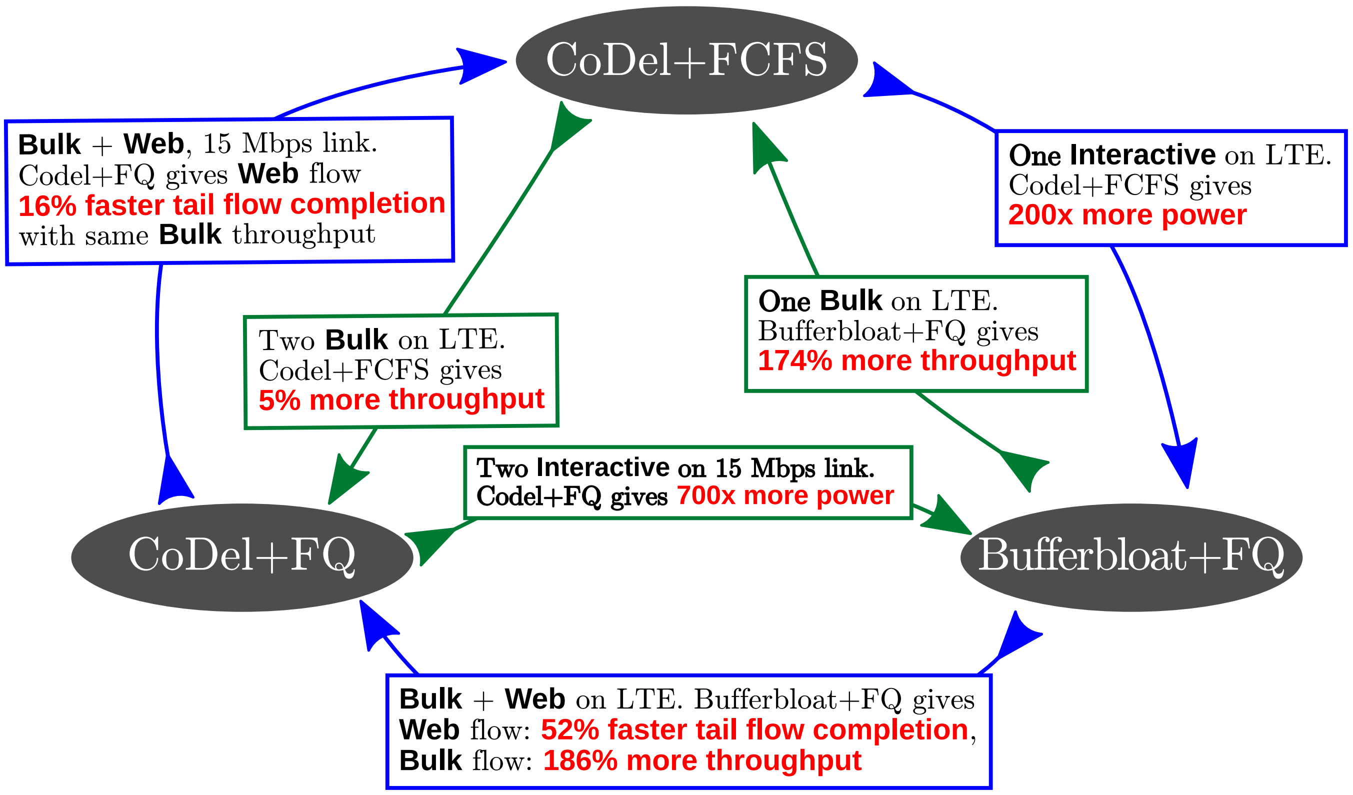
Figure 1 is an overview of our results, demonstrating bidirectional preference loops among the three schemes. One loop is shown in blue, and another in green. All experiments use NewReno as the transport protocol, and a minimum round-trip time of 150 ms. We adopt the following notation for our workloads:

- ***Bulk***: A single long-running TCP NewReno flow. Objective: maximize average throughput.

- ***Web***: A switched TCP NewReno flow. The flow is "off" for intervals drawn from an exponential distribution with mean 0.2 seconds. The flow then switches "on" until it has transferred an amount of bytes drawn from an empirical distribution of Internet flow lengths [5].[1] Objective: minimize flow completion time at the 99.9th percentile.

- ***Interactive***: A single long-running TCP NewReno flow that represents a real-time interactive application. Objective: maximize the ratio of the average throughput and the average one-way delay, called the "power."

[1]To improve link utilization, we add 16 KB to each sample drawn from the empirical distribution.



CoDel+FCFS
Bulk + Web, 15 Mbps link. Codel+FQ gives Web flow 16% faster tail flow completion with same Bulk throughput
One Interactive on LTE. Codel+FCFS gives 200x more power
Two Bulk on LTE. Codel+FCFS gives 5% more throughput
One Bulk on LTE. Bufferbloat+FQ gives 174% more throughput
Two Interactive on 15 Mbps link. Codel+FQ gives 700x more power
CoDel+FQ
Bufferbloat+FQ
Bulk + Web on LTE. Bufferbloat+FQ gives Web flow: 52% faster tail flow completion, Bulk flow: 186% more throughput

Figure 1: None of these queue-management and scheduling configurations is best. Power is throughput/(one-way delay). $A \rightarrow B$ indicates that A is better than B.

Nwk config.	Avg. throughput
CoDel+FCFS	2.00 Mbps
CoDel+FQ	1.90 Mbps

(a) CoDel+FCFS is better than CoDel+FQ on LTE

Nwk config.	Bulk Throughput	Web Tail FCT
CoDel+FCFS	9.48 Mbps	22.25 secs
CoDel+FQ	9.48 Mbps	18.71 secs

(b) CoDel+FQ is better than CoDel+FCFS on a 15 Mbps link

Table 1: CoDel+FQ vs. CoDel+FCFS

Nwk config.	Bulk Throughput	Web Tail FCT
Bufferbloat+FQ	11.22 Mbps	20.94 secs
CoDel+FQ	3.92 Mbps	43.72 secs

(a) Bufferbloat+FQ is better than Codel+FQ on LTE

Nwk config.	Avg. throughput, delay	Power
Bufferbloat+FQ	7.47 Mbps, 62165 ms	0.12 $Mbit/s^2$
CoDel+FQ	6.55 Mbps, 76.5 ms	85.6 $Mbit/s^2$

(b) CoDel+FQ is better than Bufferbloat+FQ on a 15 Mbps link

Table 2: CoDel+FQ vs. Bufferbloat+FQ

Nwk config.	Bulk throughput
Bufferbloat+FQ	11.96 Mbps
CoDel+FCFS	4.35 Mbps

(a) Bufferbloat+FQ is better than CoDel+FCFS on LTE

Nwk config.	Interactive throughput, delay	Power
Bufferbloat+FQ	11.96 Mbps, 46028 ms	0.26 $Mbit/s^2$
CoDel+FCFS	4.35 Mbps, 83.2 ms	52.28 $Mbit/s^2$

(b) CoDel+FCFS is better than Bufferbloat+FQ on LTE

Table 3: CoDel+FCFS vs. Bufferbloat+FQ

The workload and network configurations for our six head-to-head experiments are shown in Table 4. Below, we explain the more surprising results.

#	Workload	Link type	Nwk. config.
1	Bulk+Web	15 Mbps static	CoDel+FQ, CoDel+FCFS
2	Bulk+Bulk	Trace of Verizon LTE	CoDel+FQ, CoDel+FCFS
3	Interactive + Interactive	15 Mbps static	CoDel+FQ, Bufferbloat+FQ
4	Bulk+Web	Trace of Verizon LTE	CoDel+FQ, Bufferbloat+FQ
5	Bulk	Trace of Verizon LTE	Bufferbloat+FQ, CoDel+FCFS
6	Interactive	Trace of Verizon LTE	Bufferbloat+FQ, CoDel+FCFS

Table 4: Workloads used in cyclic preference experiments

1. Tables 2a and 3a show that using CoDel on this variable link results in almost a $3\times$ loss in bulk throughput compared with a bufferbloated link. Time-varying links present tricky challenges for queue-management schemes. There is an inherent tradeoff between throughput and delay on such link. A protocol seeking high throughput should send as much as it can and keep the queue continuously backlogged, so that the link doesn't starve for packets if the link quality suddenly improves. In such cases, deep buffers are the best solution. With separate per-flow queues, users do not cause large in-network delays for one another.
2. Table 1a shows that when flows are equally aggressive, FCFS is preferable to Fair Queueing. In this case, the two ***Bulk*** flows have the same RTT and are equally aggressive, so they don't need protection from each other to converge to their fair share of throughput. Running an independent CoDel instance on each queue, as in CoDel+FQ, over-controls and lowers bulk throughput for both flows.
3. Table 1b shows the opposite effect. When a ***Web*** and ***Bulk*** flow compete on the same bottleneck link, fair queueing protects the ***Web*** flow from the ***Bulk*** flow. Deploying CoDel on an FCFS queue helps, by keeping queue occupancy low. But by itself, CoDel is insufficient, because it doesn't distinguish between flows while dropping packets. Running a separate instance of CoDel helps because each flow is penalized only for its own delays. This observation is codified in the recommendation that CoDel be deployed with SFQ [2].

3.1 Conclusion

The results demonstrate that there can be no universal preference for one of these three contemporary in-network schemes. We found the results were not highly sensitive to the metrics; we observed similar behavior if we replaced the tail flow completion time with the mean flow completion time, or the mean one-way delay with the tail one-way delay. Furthermore, this behavior persists if we restrict ourselves to exclusively static links, or exclusively cellular links. We observe cyclic preferences in both cases.

To be sure, these results prove only that there are bidirectional paradoxical preferences among **these** candidate schemes. We do not claim that it's impossible that a new AQM and scheduling scheme will be invented in the future that dominates all other schemes all the time. The stark disagreement in application preferences makes only for probative evidence that there is unlikely to be a "mother of all schemes."

4. EXTENDING SDN TO THE DATA PLANE

Our approach to the problems described above is to posit that the data plane should be flexible enough to handle diverse and unanticipated application requirements. This implies the need for software, of some sort, to define the data plane's behavior.

Such flexibility could be realized easily in a software router running on a general-purpose microprocessor. However, as described in Section 2, software-only solutions are either too slow or too expensive to be commercially viable. By contrast, today's hardware switches have tightly integrated designs with little flexibility in the data plane.

To achieve both high performance and a modicum of flexibility, we propose adding a small FPGA[2] to the fast path of a hardware switch to allow queueing and scheduling to be reconfigured by the network operator. We outline a standardized interface between the FPGA and the rest of the switch that is enough to support today's in-gateway schemes.

To test the viability of our proposed interface, we used SystemVerilog and freely-available Xilinx tools to synthesize, place, and route hardware implementations of two such schemes: CoDel and RED. We report on this experience below.

4.1 What interface should the switch expose?

We considered closely the question of the appropriate interface to configure a novel queue-management or scheduling algorithm on a switch. Naively, a high-level interface where algorithms are formed out of commonly-used building blocks or gadgets has considerable appeal.

However, after attempts to synthesize popular algorithms and factor out their common components, we found that real queue-management and scheduling algorithms have such diverse control-flow graphs that no short list of high-level building blocks can easily be reconfigured to form today's existing queue-management schemes, much less those of the future.

Recognizing this, we have designed a lightweight *low-level* interface that provides a few well-defined primitives we think will be expressive enough to encompass all current schemes as well as those likely to be explored in the future.

In this framework, the primitives are literally wires or signals exposed by the network hardware to the reconfigurable

[2]A few thousand gates, not an entire board.

Class of primitive	Primitive Name	Description
Utilities	*Now*	Get current time.
Queue primitives	*Size*	Get queue length.
Queue primitives	*DropFront*	Drop packet from head of queue.
Queue primitives	*DropTail*	Drop packet from tail of queue.
Queue primitives	*Enqueue*	Enqueue packet at tail of queue.
Queue primitives	*Dequeue*	Dequeue packet from head of queue.
Queue primitives	*Transmit*	Transmit single packet.
Signaling primitives	*LinkReady*	Link is ready to accept packet.
Signaling primitives	*Arrival*	New packet just arrived.
Packet primitives	*Timestamp*	Packet's arrival timestamp.
Packet primitives	*Mark*	Set ECN bit.
Cross-layer primitives	*LinkRates*	Get current link rate.

Table 5: Data-plane primitives: wires exposed by the network processor to and from the FPGA

portion, or FPGA. In our framework, the network operator is allowed to reconfigure the FPGA arbitrarily, wrangling its "slices" as the hardware allows in order to synthesize arbitrary behavior.

Table 5 describes the minimal set of primitives that we used in implementing CoDel and RED in SystemVerilog.

Although we used SystemVerilog, a comparably low-level language, to synthesize the algorithms for an FPGA, we imagine that a practical deployment would most likely work differently. In our view, the designer of a new queue-management algorithm would be better off accessing these primitives through a high-level procedural language, such as C++, through the Verilog Procedural Interface [8].

4.2 CoDel in hardware

We implemented CoDel in SystemVerilog and successfully synthesized it for a Xilinx FPGA, using Xilinx's Vivado WebPack compiler and synthesis tools. A block diagram of the implementation is shown in Figure 2.

Our implementation used the following resources on the lowest-end FPGA available (the Xilinx Kintex-7):

CoDel resource utilization

Resource	Usage	Fraction of FPGA
Slice logic	1,256	1%
Slice logic dist.	1,975	2%
IO/GTX ports	27	2%
DSP slices	0	0%
Maximum speed	12.9×10^6 pkts/s	

We began with the CoDel pseudocode included in [1]. We maintain state in registers that track if CoDel is currently in "dropping" mode (`dropping`), the next scheduled time for a drop (`drop_next`), and the number of drops (`count`) in the current drop cycle. When *LinkReady* is signaled, we execute the same logic as specified in the *dequeue* function of the CoDel pseudocode.

SystemVerilog does not allow the synthesis of an unbounded loop. We eliminated CoDel's `while` loop, turning it into an `if` statement. Because our version of the CoDel *dequeue* function is level-triggered and fires as long as *LinkReady* remains asserted, this emulates the original effect of the `while` loop. *LinkReady* will remain asserted as long as the link can accept a packet.

CoDel also uses the *Timestamp* primitive to access the arrival time of the dequeued packet, and uses this to compute the sojourn time using the *Now* primitive. It then uses the *DropFront* primitive of the appropriate queue to drop a packet if necessary. Our implementation also uses the *Size* primitive to determine the queue length, to prevent packet drops when there is less than an MTU worth of bytes in the queue. We implemented CoDel's square-root function in fixed-point arithmetic as a lookup table.

4.3 RED in hardware

We also implemented RED in SystemVerilog, using the same interface and primitives as the CoDel implementation. We began with the RED implementation described in [13]. RED maintains state that tracks the average queue size, start of the current queue idle time, and a count of the packets seen since the last marked or dropped packet.

Unlike CoDel, all of RED's work is done when the *Arrival* signal is raised. RED needs to determine if the queue is empty for two reasons: one, to update its moving average queue size differently when the queue is empty, and two, to keep track of the start of the current idle period. To achieve this, RED uses the *Size* primitive to check if the queue is empty. Unlike CoDel, RED uses neither *DropFront* nor *DropTail*, because it drops incoming packets on arrival and doesn't need to drop packets that are already in the queue.

Our implementation used the following resources on the Xilinx Kintex-7:

RED resource utilization

Resource	Usage	Fraction of FPGA
Slice logic	1,721	1%
Slice logic dist.	3,138	3%
IO/GTX ports	27	2%
DSP slices	6	3%
Maximum speed	13.0×10^6 pkts/s	

RED requires somewhat more challenging calculations than CoDel, as the algorithm includes an exponentially-weighted moving average, which was synthesized using multiply-accumulate DSP slices, and a linear-feedback shift register for pseudo-random number generation. This also uses the DSP slices.

4.4 Feasibility

Our unoptimized implementations of CoDel and RED pass timing checks at packet-processing rates of approximately 13 million packets per second. At a typical Ethernet MTU of 1500 bytes, this translates to more than 150 Gbps. At the smallest packet size, it is about 6.7 Gbps.

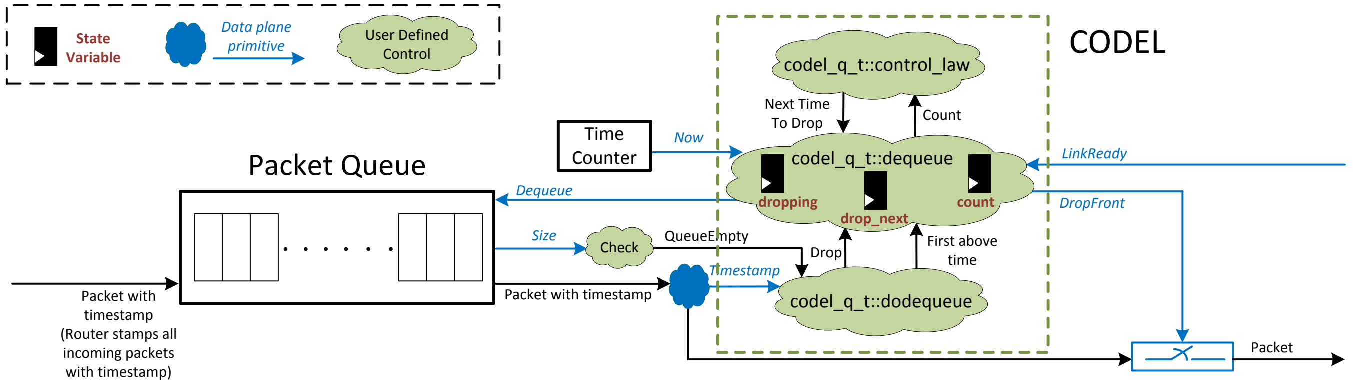


Figure 2: Block diagram of CoDel implementation

Our implementations are within the capabilities of low-end FPGAs, such as Xilinx's Spartan 3E, which sell for $10 or less. We do not know the marginal cost of such a part in a high-volume ASIC, but estimate it may be nominal. These numbers suggest that, for the purposes of scheduling and queue management, adding a small reconfigurable FPGA unit to a hardware router may allow it as much flexibility as a software router without affecting performance.

5. Limitations and Future Work

It is natural to ask what kind of schemes cannot be expressed by our abstractions. Any scheme that needs to look beyond the packet header, for instance, "deep packet" inspection, intrusion detection, encryption, and spam filtering cannot be expressed using the abstractions above. To make our proposal cost-effective, the FPGA we propose to add is constrained, limiting the sophistication of any implementable scheme.

Our proposed design currently runs at a line rate of 10 gigabits/sec. More work needs to be done before the same design can run at 10 gigabits/sec on several ports simultaneously. If the queueing or scheduling algorithm performs all its work only when packets are dequeued, this problem can be solved by replicating digital logic. However, several algorithms perform some work during both packet enqueues and dequeues, and the enqueueing rate can be up to P times the line rate, where P is the number of ports. We have also not clearly specified how packets are mapped to per-port hardware queues, which are then used for packet scheduling. One option is to leverage OpenFlow's existing match-action capabilities for classifying packets into per-port hardware queues.

We have not specified a scheme by which applications can signal their objectives (throughput, low delay, power, flow completion, transaction completion, tail completion) to the network fabric. We imagine that the ToS or DiffServ codepoint bits in the IP header would be used for this purpose. This may work within a datacenter where agreement can be secured as to the meaning of various code points. But today, Internet Service Providers generally do not trust the DiffServ code points they receive from other autonomous systems, and the practical obstacles to Internet-wide deployment of such a signaling scheme remain formidable.

More work needs to be done to characterize the specifications sufficient to guarantee interoperability between an arbitrary switch and a queue-management or scheduling algorithm that may be synthesized in an FPGA, in order to provide cross-vendor portability of a design. We don't claim to have answers to this question, and leave this for future work.

6. Conclusion

We have demonstrated perverse bidirectional preference loops among queue-management and scheduling schemes that run inside the network, suggesting there can be no overall winner for the best AQM or scheduling behavior.

Instead, we advocate for an open interface tailored to queueing and scheduling that is portable across vendors. We propose a compromise position between fully software switches and current hardware switches, which are configurable only in their control-plane functions. In our scheme, switches would contain a small FPGA with a simple interface to the switch's packet queues.

We have described this interface and implemented two schemes — CoDel and RED — as a proof of concept. Their resource requirements were exceedingly modest, giving hope that such a system may be practical. Instructions to replicate our results are available at: http://web.mit.edu/anirudh/www/sdn-data-plane.html

7. Acknowledgements

We are grateful to Patrick Bosshart, Glen Gibb, and the HotNets reviewers for their thoughtful feedback and comments. We thank Chia-Hsin Owen Chen for assisting us in developing the SystemVerilog implementations of CoDel and RED, and Murali Vijayaraghavan for help using the Xilinx tools. This work was supported in part by NSF grant CNS-1040072. We also thank the members of the MIT Center for Wireless Networks and Mobile Computing (Wireless@MIT), including Amazon.com, Cisco, Google, Intel, Mediatek, Microsoft, ST Microelectronics, and Telefonica, for their support.

8. References

[1] Appendix: CoDel pseudocode. http://queue.acm.org/appendices/codel.html, 2012.

[2] Benchmarking CoDel and FQ CoDel. http://www.bufferbloat.net/projects/codel/wiki/Benchmarking_Codel_and_FQ_Codel?version=4, 2012.

[3] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data Center TCP (DCTCP). In *SIGCOMM*, 2010.

[4] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKewon, B. Prabhakar, and S. Shenker. pfabric: Minimal near-optimal datacenter transport. In *Proceedings of the ACM SIGCOMM 2013 conference*, SIGCOMM 2013, New York, NY, USA, 2013. ACM.

[5] M. Allman. Comments on Bufferbloat. *ACM SIGCOMM Computer Communication Review*, 43(1), Jan. 2013.

[6] M. B. Anwer, M. Motiwala, M. b. Tariq, and N. Feamster. Switchblade: a platform for rapid deployment of network protocols on programmable hardware. In *Proceedings of the ACM SIGCOMM 2010 conference*, SIGCOMM '10, pages 183–194, New York, NY, USA, 2010. ACM.

[7] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker. Ethane: Taking control of the enterprise. In *ACM SIGCOMM Computer Communication Review*, volume 37, pages 1–12. ACM, 2007.

[8] C. Dawson, S. K. Pattanam, and D. Roberts. The verilog procedural interface for the verilog hardware description language. In *Proceedings of the 1996 IEEE International Verilog HDL Conference (IVC '96)*, IVC '96, pages 17–, Washington, DC, USA, 1996. IEEE Computer Society.

[9] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. In *Symposium proceedings on Communications architectures & protocols*, SIGCOMM '89, pages 1–12, New York, NY, USA, 1989. ACM.

[10] M. Dobrescu, N. Egi, K. Argyraki, B.-G. Chun, K. Fall, G. Iannaccone, A. Knies, M. Manesh, and S. Ratnasamy. Routebricks: exploiting parallelism to scale software routers. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pages 15–28. ACM, 2009.

[11] W. Feng, K. Shin, D. Kandlur, and D. Saha. The BLUE Active Queue Management Algorithms. *IEEE/ACM Trans. on Networking*, Aug. 2002.

[12] S. Floyd. TCP and Explicit Congestion Notification. *CCR*, 24(5), Oct. 1994.

[13] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Trans. on Networking*, 1(4), Aug. 1993.

[14] S. Ha, I. Rhee, and L. Xu. CUBIC: A New TCP-Friendly High-Speed TCP Variant. *ACM SIGOPS Operating System Review*, 42(5):64–74, July 2008.

[15] S. Han, K. Jang, K. Park, and S. Moon. Packetshader: a gpu-accelerated software router. *ACM SIGCOMM Computer Communication Review*, 40(4):195–206, 2010.

[16] C.-Y. Hong, M. Caesar, and P. B. Godfrey. Finishing flows quickly with preemptive scheduling. *SIGCOMM Comput. Commun. Rev.*, 42(4):127–138, Aug. 2012.

[17] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hozle, S. Stuart, and A. Vahdat. B4: Experience with a globally-deployed software defined wan. In *Symposium proceedings on Communications architectures and protocols*, SIGCOMM 2013, New York, NY, USA, 2013. ACM.

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

[19] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The click modular router. *ACM Transactions on Computer Systems (TOCS)*, 18(3):263–297, 2000.

[20] S. Kunniyur and R. Srikant. Analysis and Design of an Adaptive Virtual Queue (AVQ) Algorithm for Active Queue Management. In *SIGCOMM*, 2001.

[21] J. W. Lockwood, N. McKeown, G. Watson, G. Gibb, P. Hartke, J. Naous, R. Raghuraman, and J. Luo. Netfpga–an open platform for gigabit-rate network switching and routing. In *Microelectronic Systems Education, 2007. MSE'07. IEEE International Conference on*, pages 160–161. IEEE, 2007.

[22] P. E. McKenney. Stochastic Fairness Queueing. In *INFOCOM*, 1990.

[23] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

[24] K. Nichols and V. Jacobson. Controlling Queue Delay. *ACM Queue*, 10(5), May 2012.

[25] R. Pan, B. Prabhakar, and K. Psounis. CHOKe—A Stateless Active Queue Management Scheme for Approximating Fair Bandwidth Allocation. In *INFOCOM*, 2000.

[26] E. Rubow, R. McGeer, J. Mogul, and A. Vahdat. Chimpp: A click-based programming and simulation environment for reconfigurable networking hardware. In *Architectures for Networking and Communications Systems (ANCS), 2010 ACM/IEEE Symposium on*, pages 1–10. IEEE, 2010.

[27] C. Tai, J. Zhu, and N. Dukkipati. Making Large Scale Deployment of RCP Practical for Real Networks. In *INFOCOM*, 2008.

[28] K. Tan, J. Song, Q. Zhang, and M. Sridharan. A Compound TCP Approach for High-speed and Long Distance Networks. In *INFOCOM*, 2006.

[29] D. L. Tennenhouse and D. J. Wetherall. Towards an active network architecture. *Computer Communication Review*, 26:5–18, 1996.