One More Queue is Enough: Minimizing Flow Completion Time with Explicit Priority Notification

Yuanwei Lu*†, Guo Chen**†, Larry Luo‡, Kun Tan†, Yongqiang Xiong†, Xiaoliang Wang†, Enhong Chen†
*University of Science and Technology of China †Microsoft ‡Nanjing University

***Abstract*—Ideally, minimizing the flow completion time (FCT) requires millions of priorities supported by the underlying network so that each flow has its unique priority. However, in production datacenters, the available switch priority queues for flow scheduling are very limited (merely 2 or 3). This practical constraint seriously degrades the performance of previous approaches. In this paper, we introduce Explicit Priority Notification (EPN), a novel scheduling mechanism which emulates fine-grained priorities (*i.e.*, desired priorities or DP) using *only two switch priority queues*. EPN can support various flow scheduling disciplines with or without flow size information. We have implemented EPN on commodity switches and evaluated its performance with both testbed experiments and extensive simulations. Our results show that, with flow size information, EPN achieves comparable FCT as pFabric that requires clean-slate switch hardware. And EPN also outperforms TCP by up to 60.5% if it bins the traffic into two priority queues according to flow size. In information-agnostic setting, EPN outperforms PIAS with two priority queues by up to 37.7%. To the best of our knowledge, EPN is the first system that provides millions of priorities for flow scheduling with commodity switches.**

I. Introduction

Recent years have witnessed the unprecedented growth of large datacenters that contain hundreds of thousands of servers and host a wide-range of diverse distributed applications. Many key datacenter applications in cloud, such as web search, social and storage services, involve many small transactional communications and are very sensitive to the actual completion time of network flows. Thus there is a wide consensus that minimizing flow completion time (FCT) is of significant importance. To minimize FCT, previous work has shown that it is essential to prioritize network flows to approximate Shortest Job First (SJF) or Shortest Remaining Processing Time (SRPT) scheduling discipline [6, 7, 23, 27, 33, 34]. Ideally, the network should support many fine-grained priorities. For example, pFabric [6] has illustrated that "near-optimal" FCT can be achieved if each flow can be assigned to a unique priority, so that the link bandwidth can be allocated to flows strictly according to their priorities. There are millions of flows in a datacenter network (DCN). So ideally we'd like to have the same number of priorities.

Unfortunately, existing commodity switches only provide a very limited number of priority levels. For example, some commodity switches only support eight priorities queues, and in practice, not all of these priority queues can be used for flow scheduling. As we will discuss more in §II-B, the number of priority queues that can be allocated to one traffic class may be at most 2 or 3 in a production datacenter. With such small number of queues, many flows with different sizes have to coexist in the same queue, resulting in worse performance.

Indeed, other previous work has proposed to emulate fine-grained priorities with one drop-tail queue using *explicit rate control* (ERC). In ERC, the switch needs to examine every data packet and compute the sending rate for each flow based on the scheduling policy. Unfortunately, ERC is seldom supported by existing switch hardware due to its per-packet examination and complex logic. And most software-based ERC implementations cannot sustain high link speed of modern datacenters.

In this paper, we want to ask the following question: *Can we achieve fine-grained priority-based flow scheduling (i.e., millions priorities like in pFabric), but with only the features that are supported by existing commodity switch hardware?* We answer this question affirmatively with Explicit Priority Notification (EPN). We show that with merely one additional priority queue, EPN can schedule flows based on flows' fine-grained priorities (or *desired priority*, DP), and achieve similar FCT performance compared to schemes that natively support many priority levels, *i.e.*, pFabric.

The design of EPN is inspired by the following observation: With priority-based flow scheduling at any bottleneck network link, the bandwidth should be allocated entirely to the flow with the highest priority. Therefore, each link only requires two physical priority queues (called *transmission priority* or TP). One has higher priority (called HIGH queue) than the other queue (called LOW queue). The flow with the highest DP is always assigned to the HIGH queue and all rest flows are assigned to the LOW queue. When the flow in the HIGH queue finishes, the flow with the next highest DP will be promoted into the HIGH queue, and so forth. Essentially, this behavior mimics dynamic priority-based flow scheduling.

To implement this, EPN employs two software components: a *priority controller* at each switch and a *priority mapper* at each host. The priority mapper maintains a mapping between the flow's DP and TP, and tags this information on every packet of the flow[1]. The priority controller maintains a list of flows by examining the packets passing through the switch and decides flows' TPs by comparing their DPs. If a flow's TP should be changed, the prioirty controller will send an *Explicit Priority Notification* message to notify the priority mapper on the flow source. So the next new packet of the flow will be tagged with the new TP and be classified into the corresponding HIGH or LOW queue in switch hardware. EPN is a general mechanism to support various priority-

** Guo Chen is the corresponding author.

[1]In this paper, TP is marked in the DSCP field in the IP header. Current commodity switches can conduct priority scheduling according to the DSCP.

based flow scheduling disciplines with or without flow size information, which is useful in abound scenarios. However, we are aware that EPN still has some limitations. For example, currently, EPN cannot enforce some complex non-priority-based flow scheduling policies [11] or Coflow Scheduling policies [12, 14]. See more details in §VII.

EPN has been implemented in a commodity switch with Broadcom Trident II chipset and run in real-time in a 1Gbps testbed with 22 servers and 3 switches. Through both testbed and large-scale simulation experiments, we compare EPN to previous work in both flow size information-aware and information-agnostic settings. Our results show that with flow size information, EPN, with only two hardware queues, achieves similar FCT compared to pFabric that runs on clean-slate switch hardware supporting millions of priority queues. However, if only limited hardware queues are used, EPN greatly outperforms a practical version of pFabric (more in § VI) by up to 60.5% and 35.1%, when the practical version of pFabric uses 2 or 8 queues respectively. A bit to our surprise, in information-agnostic setting, EPN also outperforms PIAS [7], a recent information-agnostic flow scheduling framework, by up to 37.7% and 19.3%, when PIAS uses 2 or 8 hardware priority queues respectively. The reason is that the number of levels in Multi-Level Feedback Queue (MLFQ) in PIAS is tightly constrained by the number of hardware queues. But in EPN, we can always use the optimal levels of MLFQ to adjust flow's priority. To the best of our knowledge, EPN is the first system that provides millions of priorities for flow scheduling with commodity switch hardware.

II. Background and Motivation

A. Near-optimal flow scheduling to minimize FCT

It is well-known that for a single bottleneck link, SJF (shortest-Job-First) and SRPT (Shortest-Remaining-Processing-Time) achieve minimum average FCT in the static and dynamic scheduling scenario respectively. For a general network, the optimal flow scheduling is known to be NP-hard [6]. However, previous work, *e.g.*, [6, 23] uses a simple greedy heuristic to achieve near-optimal flow scheduling in a datacenter network: Network flows are scheduled across the network fabric in non-decreasing order of their priorities (determined by flow size or remaining size). Therefore, the bottleneck link bandwidth is always allocated to the most critical flow (the one with highest priority along its traversing path) as much as it requires and other flows should wait for their turns to come. A theoretical analysis has shown that such simple greedy algorithm achieves 2-approximation to optimum [9].

Clearly, to implement this near-optimal scheduling, one natural idea is to have switches to support many fine-grained priority queues, so that each flow can be assigned to one priority queue according to its size. Unfortunately, in practice, the number of priority queues in switch hardware is very limited.

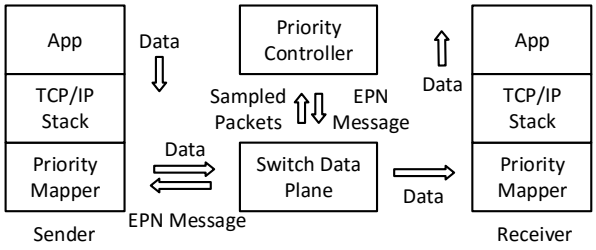


Fig. 1. EPN Sytem Overview

B. The lack of priority queues in commodity DCNs

There are two reasons that constrain the number of available priority queues for flow scheduling in a commodity switch. Firstly, many existing commodity switches support only two to four priority queues, *e.g.*, BCM5324M based switches support 4 priority queues [1]. Secondly, although many modern switches can support up to eight priority queues, many of them are already reserved for other purposes [24]. We have interviewed a senior network manager who is responsible to operate several datacenters of a large IT company and confirmed that typically 2 to 3 priority levels have been used to isolate new network protocols, *e.g.*, RDMA or DCTCP, from normal TCP traffic. Three priorities are used to provide different QoS classes for TCP traffic, *e.g.*, real-time, best-effort, or background class. Consequently, there are only 2 to 3 priority queues left for flow scheduling in one traffic class. With this limited number of priority queues, the performance of pFabric would be greatly degraded from the optimal.

In this paper, we set out to ask: *Can we achieve fine-grained priority-based flow scheduling with only the features provided by existing commodity switch hardware?* In the following sections, we answer this question affirmatively with EPN.

III. EPN Overview

The core idea of EPN is the decoupling of the desired fine-grained priority (*desired priority* or DP) and the transmission priority (TP) that is used for switch hardware to classify the flow packets. Each flow is assigned a DP, which is derived from its flow size or other metrics like the number of transmitted bytes [7]. During the lifetime of the flow, EPN may dynamically map the flow to different TPs based on the relative order of the flow's DP among all other flows that are sharing some network links with it. Clearly, to implement the priority-based scheduling outlined in §II-A, EPN requires only two TPs: a HIGH queue and a LOW queue. At any instance, the flow with the highest DP should be put into the HIGH queue, and all other flows should share the LOW queue.

Figure 1 shows the overview of the EPN system. EPN contains two major components: a priority mapper at the end host and a priority controller on each network switch. The priority mapper maintains a DP to TP mapping for each flow. The DP is carried in an EPN header that is attached to every data packet. The format of the EPN header is shown in Figure 2. The TP, however, is also tagged to each packet using existing mechanisms, *i.e.*, DSCP code in IP header. The switch hardware is instructed to selectively sample a few critical data packets, *e.g.*, TCP SYN, FIN and RST, and mirror them to the

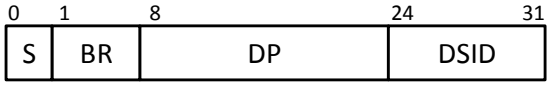


Fig. 2. EPN Header Format.

priority controller. From these sampled packets, the priority controller builds a list of flows and sorts them according to their DPs. Then, if the flow with the highest DP is not already mapped to the HIGH queue, the priority controller will promote it to the HIGH queue and at the meanwhile, demote the flow that is already in the HIGH queue to the LOW queue if any. If any flow's TP needs to change, the priority controller will send an EPN message to the priority mapper on the flow source. The EPN message is simply a copy of the packet headers (*i.e.*, IP, EPN and TCP) with the new TP tagged in the EPN header. Upon receiving the EPN message, the priority mapper will update its local record and map the flow to the new TP.

In this paper, we embed the EPN header inside the reserved bits of VxLAN [26], which is widely used in datacenters to tunnel packets among different virtual machines. Similarly, EPN can also work with other tunnel protocols, like NVGRE [31] or STT [15].

The decoupling between the flow DP and TP makes EPN a general framework for priority-based flow scheduling. EPN can incorporate various priority-based flow scheduling disciplines by adopting different DP adjustment algorithms. For example, we can set flow DP according to its remaining size if flow size information is known [6, 23]. Alternatively, EPN can also perform information-agnostic flow scheduling using a MLFQ-based algorithm [7] (more details in §IV-C).

IV. Detailed Design

A. EPN flow scheduling

At high level, the flow scheduling in EPN is very simple. The priority controller maintains a list of flows passing the switch, and for each output link, the controller will select the flow with highest DP to the HIGH queue and put all other flows into the LOW queue. If two flows have equal DP, ties are broken by comparing flow 5-tuples similar to [6]. Once a flow TP needs to change, the priority controller will send out an EPN message to the flow source.

However, there are a few detailed issues need to take care.

Inconsistent decisions from multiple switches. In DCN, a flow may bypass several switches to reach the destination and at each switch, the flow may share link with different sets of competing flows. Therefore, it is possible that two switches in the flow path may make different decisions of the flow TP. Figure 3 shows such an example. Flow f_2 traverses switch A and B and shares links with flow f_1 and f_3 at two switches respectively. Assume the three flows have $DP(f_1) < DP(f_2) < DP(f_3)$. At switch A, f_2 is the highest priority flow, so that switch A tries to promote it to the HIGH queue. But at switch B, f_3 has the highest DP. So if switch B finds f_2 is mapped to the HIGH queue, it will demote f_2 into the LOW queue. As a result, the TP of f_2 will oscillate. This issue can be simply removed by adding a **Demoting Switch ID** (or DSID) into the EPN header(Figure 2). DSID marks the switch that demotes the flow. If a switch finds the DSID of a flow does not point to itself, it should not try to promote the flow as the flow would be demoted at another switch.

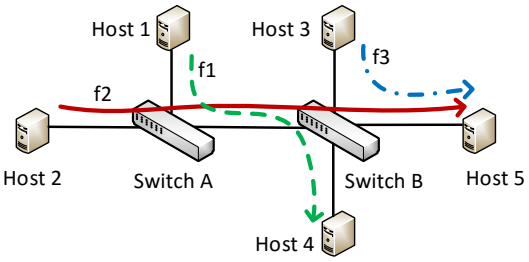


Fig. 3. Inconsistent TP decisions between two switches. f_3 has the highest DP and f_1 has the lowest DP. f_2 is in-between.

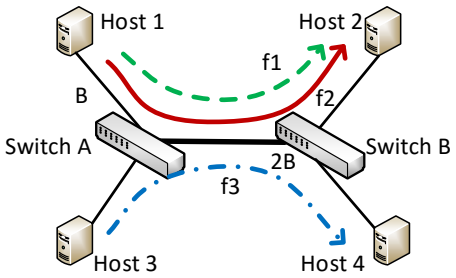
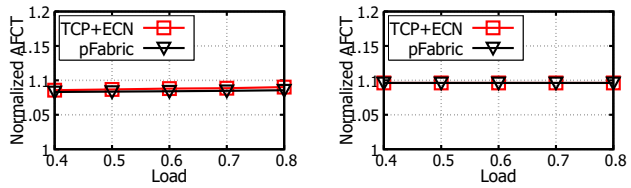


Fig. 4. An example for asymmetric network. f_3 has the highest DP and f_1 has the lowest DP. f_2 is in-between.

There is one more tricky thing. How should we identify a switch? Clearly, in a datacenter, there are thousands of switches. We could assign each switch a unique ID, but this is cumbersome: First, it is a tedious and error-prone process to do this assignment. Second, it may take more bits (*i.e.*, 12+) in the EPN header to record it, making it more difficult to embed the EPN header in existing tunnel standards, *e.g.*, VxLAN only reserves 32 bits. In this paper, we observe that for EPN, the switch ID needs only be unique along the network path of a flow. So we simply use the TTL field of the IP header as the switch ID, as it will decrease by one after passing a switch along the network path.

Asymmetric network. Commonly, links in DCN may have different speeds. For example, the link to a server may be 10Gbps. But the links that connect the ToR switch to the Spine may have much higher speed like 40Gbps. Figure 4 shows a concrete example, where the switch-switch link has twice capacity compared to the server-switch links. Now, if switch A only schedules one flow (*i.e.* f_3) in the HIGH queue on the switch-switch link, f_2 and f_1 are forced to share the LOW queue together. This is not optimal as the switch-switch link has enough bandwidth to support f_2 in the HIGH queue as f_3 is bottlenecked at its access link. To address this issue, EPN stores the *bottleneck rate*, *i.e.* the speed of the link that has minimal capacity in the path in the **BR** field of the EPN header. The priority controller computes the sum of bottleneck rates of all flows that are mapped to the HIGH queue. If this sum is below the link capacity, the priority controller will



(a) Web search (b) Data Mining

Fig. 5. [Simulation] FCT of tiny flows w/ and w/o strict priority scheduling.

try to promote more flows into the HIGH queue. In current design, the **BR** field has 7 bits and the unit is 1Gbps. So it can represent link speeds from 1G to 127Gbps. To find the bottleneck rate, when a flow starts, the priority mapper will first fill its output link rate in the **BR** field. And when the packet passes through the switches along the path, the switch will send an EPN notification to the source if the switch's output link speed is smaller than the value in the EPN header. After the packet arrives at the receiver, the source will pick up the lowest link speed as its bottleneck rate and tag the subsequent packets of this flow.

Tiny flows. For very tiny flows, whose sizes are less than bandwidth delay production (BDP), the EPN scheduling will not get any benefit, since the flow would have already finished when the EPN messages arrive at the flow source. Therefore, in EPN, we just put these tiny flows into the HIGH queue and the priority controller will simply ignore them. This decision is reasonable. Although the number of tiny flows are large, they only contribute to a small portion of data traffic. For example, Fig. 10 shows two typical workloads in a datacenter [5, 19]. The tiny flows only account for 1 % of overall traffic volume and their transmissions are seldom overlapped. Therefore, there is no need to schedule these tiny flows.

To verify this statement, we further conduct a simulation study. In our simulation, we first generates load consists flows across all sizes using the two real distributions in Fig. 10. Then we only put tiny flows that are smaller than 69KB (a BDP worth of data) into the network by filtering larger flows. We compare the normalized average FCT (AFCT) of these flows without any scheduling (labeled as "TCP+ECN") and with pFabric that performs strict priority scheduling. Fig. 5 shows the result. We can see that the performance in two cases are really close. pFabric provides just 0.1% to 0.5% advantage. This result confirms our decision not to perform scheduling over these tiny flows.

Algorithm 1 summarizes the EPN scheduling algorithm, which is executed on every sampled packets at the priority controller. Line 1 checks the DSID of the flow. If the DSID points to another switch, no further process is performed. Line 4 to 6 selects the set of flows in the HIGH queue and put rest of the flows into the LOW queue. Line 7 sends out EPN messages if needed.

B. Selective sampling

Before the priority controller can perform flow scheduling, it should first collect information of all flows that pass the switch. Since modern switch may serve multiple Tb traffic per second, it is simply infeasible for the priority controller

Algorithm 1 EPN Priority Controller On Receiving a Packet

Require: incoming packet pkt, flow table $FlowTable$;

1: **if** $pkt.DSID = otherswitch$ **then**
2: delete flow from $FlowTable$ if found;
3: **else**
4: sort flows in DP ascending order, ties are broken by comparing flow five-tuple;
5: select out κ flows, so that $\sum_{i=1}^{\kappa} BR_i \leq LinkRate$ and $\sum_{i=1}^{\kappa+1} BR_i > LinkRate$;
6: mark those κ flows' TP to HIGH queue, other flows' TP to LOW queue;
7: on flow TP change, send EPN message to flow sender;
8: **end if**

to examine every data packet. Therefore, in EPN, we use a selective sampling strategy. A preinstalled rule is used to match any packet with **S**-bit set in the EPN header and mirror this matched packet to the priority controller.

The priority mapper is responsible to S-mark a data packet. Following rules are applied to select packets to be sampled: 1) All TCP control packets like SYN, FIN and RST should be S-marked. For those persistent TCP connections, data are sent in bursts, and there are no SYN nor FIN packets for each burst. In those cases, each burst is treated as a different flow in EPN. EPN relies on application to specify the start and end packets of the flow, *i.e.* burst, and those packets are S-marked. 2) If the priority mapper receives an EPN message to change a flow TP, the next packet of the flow should be S-marked. This is equivalent to send an acknowledgment to the switch. 3) If a flow changes its DP, the priority mapper should S-mark the next data packet of the flow to notify the switches in the network path.

It is noteworthy that when a priority controller fails and reboots, it forgets all its current flow information. In such case, flows demoted by this controller may never have a chance to be promoted (other controllers cannot promote because they have different DSID). To prevent such deadlock, flows that have been demoted to LOW queue will S-mark their packets periodically to probe for promotion. As a failover scheme, this probing interval can be set quite long, say several ms.

Finally, the priority controller should keep a close track on each flow in the HIGH queue. If such a HIGH queue flow dies, the priority controller should be able to quickly detect it and schedule another flow into the HIGH queue. Note that it does not need to track the flows in the LOW queue, as a dead flow will be eventually detected by the controller when trying to promote it to the HIGH queue, or be removed after a long timeout. We use the programmable counters to track the liveness of a flow. Such counters are widely supported in current switch hardware. For example, in our BCM95334K switch, there are 1.7K programmable counters. When a flow is promoted to the HIGH queue, the priority controller will also install a rule in the switch hardware to count the number of bytes sent by the flow. Then, the priority controller will read the counter periodically, *e.g.*, every a few ms. If the counter

remains unchanged for several reads, the flow is detected as dead and removed from the list.

C. Setting DP

As aforementioned, the decoupling of flow DP and TP enables EPN a general flow scheduling framework for various priority-based scheduling principles. When accurate flow size information is available (*i.e.* information-aware), DP can be set to flow size or flow remaining size, thus minimizing FCT by using SJF or SRPT scheduling principle. In many cases, the accurate flow size information may not be available (*i.e.* information-agnostic), and recent work PIAS [7] proposes to use MLFQ to emulate SJF flow scheduling, where a flow is gradually demoted from higher-priority queues to lower-priority queues. EPN can easily implement similar MLFQ algorithm to support information-agnostic scheduling as well. If the flow size information is not available, the priority mapper can use MLFQ to adjust the flow DP based on the number of bytes it has transmitted. Under different DP settings, the behavior of the priority controller does not need to change, and always allocates link bandwidth to the flows with higher DP.

Unlike PIAS, which tightly couples the number of MLFQ levels with the physical priority queues, EPN can always apply the optimal number of MLFQ levels as long as there are two hardware queues. The ability to have many MLFQ levels greatly simplifies the threshold setting, which is one key challenge for information-agnostic flow scheduling [7]. In PIAS, if the levels of MLFQ are limited, the thresholds to move to a lower level should be carefully optimized according to the precise flow size distribution. However, in EPN, with many MLFQ levels, this threshold setting is less sensitive to the actual flow size distribution. Therefore, we can follow a simple scheme, which exponentially increases the threshold when moving to a lower level [13, 21], for all practical traffic patterns without specific knowledge of their flow size distributions. Later in §VI, we show that EPN outperforms PIAS significantly with this simple threshold setting strategy.

D. Discussion

We discuss a few issues in this section.

Application-constraint flows: In practice, some flows may be bottlenecked by the application instead of the network links. In those cases, HIGH priority flow cannot saturate the link, and the rest bandwidth is shared among multiple LOW priority flows. However, EPN is still work-conserving and the whole network bandwidth is fully utilized. Moreover, it still performs better than baseline TCP without any scheduling, because EPN still ensures the priority scheduling for the flow with the highest priority. Further improvements can be done using a rate estimator at the end host to consider the application sending speed when calculating the bottleneck rate BR. We plan to study more about this in the future.

TCP Out-Of-Order: EPN works with all existing transport protocols, predominately TCP. Since EPN schedules packets of one TCP flow between two priority queues, these packets may get out-of-order (OoO), which may adversely affect TCP performance [25]. However, the OoO packets generated by EPN scheduling is minor and can be ignored. The reasons are as follows. First, if a flow switches from HIGH queue to the LOW queue, there is no OoO packets. Second, when a flow switches from LOW queue to the HIGH queue, there is a chance for OoO, but the chance is very low. This is because the LOW queue is shared by many flows, so that most of flows have a small window, usually only one. When the flow is promoted to the HIGH queue, the sender will send out new data packets (marked as high TP) only when it receives ACKs from the receiver. This means the packets queued in the LOW queue have already drained. To qualify this effect, we perform an EPN simulation using NS3 [2] with a 10G FatTree topology with 128 servers (settings are discussed more in §VI). We use two different workloads as shown in Fig. 10. We run the simulation for 10 seconds and record the probability of consecutive duplicate ACKs. Of all the duplicate ACKs, we eliminate those caused by packet drops to show only OoO caused ones. As shown in Fig. 6, in about 99.9% cases, EPN scheduling does not cause any OoO packets. In merely 0.005% cases, EPN may cause 3+ duplicate ACKs, which may result in a false retransmission. In summary, the OoO packets caused EPN scheduling is actually minor and can be neglected in practice.

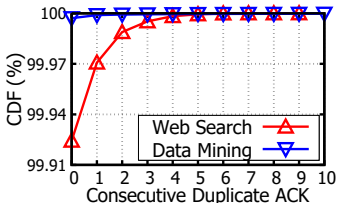


Fig. 6. [Simulation] CDF for consecutive duplicate ACK amount.

Frequency of DP adjustment: In EPN, when a flow DP changes, the priority mapper should S-mark a packet to notify this change to the priority controller. If a flow changes its DP too frequently, *e.g.*, updating for every packet, this will effectively force the priority controller to sample every data packet, adding considerable processing burden. To address this issue, the priority mapper will rate limit the S-marked packets below a threshold.

Multi-Path Support: Recent works propose to enable multi-path in datacenter networks [4, 10, 16, 22, 35]. In those works, a flow is cut into small parts, *e.g.*, sub-flows (or flowlets), and each part may traverse different paths. A flow is finished only when all its consisting parts are finished. Scheduling under multi-path settings is much like recently proposed task-aware scheduling [12, 14, 17], in which a task is finished only when all it's consisting flows are finished. EPN in its current form doesn't support scheduling under multi-path settings or task-aware scheduling. EPN works with the de facto ECMP routing in datacenters, *i.e.* a flow goes through a single path. Extending EPN to support multi-path settings and to be task-aware is part of our future work.

V. Implementation

A. Priority Mapper

We have implemented the priority mapper in Windows operating system. The architecture of the priority mapper is illustrated in Figure 7.

Kernel mode packet filter driver. We have developed an NDIS (Network Driver Interface Specification) filter driver in Windows kernel. This filter driver is under the TCP/IP stack and it intercepts all outgoing and incoming traffic. All intercepted packets are directed to a user-mode priority mapper module for processing. Since EPN needs to add additional VxLAN header for every packet, we turn off the TCP Large Segmentation Offloading (LSO) and Receive Segment Coalescing (RSC). We optimized the driver to make it able to operate at line rate on our testbed.

Priority mapper module. All functions of the priority mapper are implemented in a user-mode module. The prioirty mapper module reads packets captured by the kernel filter driver. For outgoing packets, the priority mapper will add a VxLAN header to each packet and also fill the EPN header. For incoming packets, the priority mapper will strip the VxLAN header and hand the original packets to the application. All flow information is maintained using a hash table, indexed by flow 5-tuples. Currently, we maintain a counter for the sent bytes for every flow and we also implement a MLFQ in the priority mapper with reconfigurable number of levels and thresholds.

B. Priority Controller

Due to the limited programmability of the switch in our testbed, we implement the priority controller in a commodity server that sits aside the switch (Figure 8). We configure the switch to match any S-marked packet and mirror them to the control server. Then, the controller software will capture these packets, execute the scheduling algorithm and send out EPN messages as outlined in §IV-A.

Ideally, the priority controller should be implemented in the switch CPU. To estimate the packet rate that the priority controller needs to handle, we refer to the Everflow system [37]. Similar to EPN, EverFlow also uses match-and-mirror function in the switch to sample all critical packets of TCP flows, like SYN, FIN and RST. According to their measurement in a large datacenter with 10K switches, the average rate of sampled packets from one switch is about 30Kpps, which should be rather easily supported even with a relatively weak CPU on a switch.

C. Switch Configuration

Packet Scheduling. Our testbed switches can support up to 8 queues. For EPN, we configure two Weighted Round Robin (WRR) queues with weights of 100 and 1. We deliberately reserve some bandwidth to the LOW queue to avoid starvation. For PIAS and pFabric, we configure strict priority queueing as they require.

ECN Marking. As stated in [7, 8], per-queue ECN marking may cause lots of packet drops when many queues are active simultaneously. Thus we configure per-port ECN for PIAS as it requires [7]. Also, for pFabric with limited number of queues (more in§VI), we use the same configuration as PIAS. For EPN and other schemes with 2 priority queues, switch buffer is large enough in a per-queue basis. Thus, we configure per-queue ECN for EPN and those schemes.

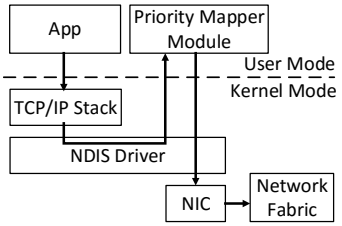


Fig. 7. EPN priority mapper.

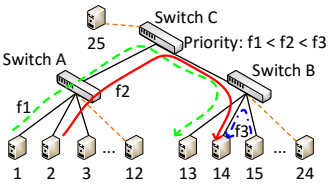


Fig. 8. Testbed Topology

D. Testbed Setup

We use 3 24-port GbE BCM956334K switches and 25 Dell PE R610s servers to build the testbed (Figure 8). 22 servers ($h1 \sim h11$, $h13 \sim h24$) are used for data communication while the other 3 ($h12$, $h24$, $h25$) are used for the priority controller. All servers are installed Windows 2008R2 along with our NDIS driver. The MTU is 1.5KB.

VI. Evaluation

Using both testbed experiments and large-scale packet-level NS3 [2] simulations, we have following three key observations:

- Targeted testbed micro-benchmarks show that EPN can provide fast and seamless switching between flows according to their priorities. (§VI-B)
- Testbed experiments using realistic workloads under both information-aware and information-agnostic scenarios show that EPN outperforms other latest practical solutions under various realistic scenarios. (§VI-C)
- Simulations show that EPN can scale to larger and faster network, and achieves comparable performance to pFabric that relies on clean-slate switch hardware. (§VI-D)

Performance Metric: Except the micro-benchmark experiment, in all the following experiments, we use normalized FCT as the performance metric for different scheduling schemes, *i.e.*, FCT is normalized to the ideal FCT which is the best FCT a flow can achieve on an idle link under TCP.

A. Schemes compared

TCP + ECN [36]: This scheme is compared as the baseline. In this scheme, TCP flows simply share the bandwidth with others (if any) and cuts its window by half in the presence of ECN [30]. Note that we don't use DCTCP [5], because it's currently not available on our Windows operating system and TCP + ECN with a deliberately tuned threshold can have similar performance as DCTCP [36].

PIAS: We implement PIAS in both simulation and testbed. We evaluate the performance of PIAS with different queues, *e.g.*, 8 queues (denoted as PIAS8, *etc.*).

TCP[α]Q: This is the dirty-slate version of pFabric described in §6 of [6]. It classifies flows to different priority

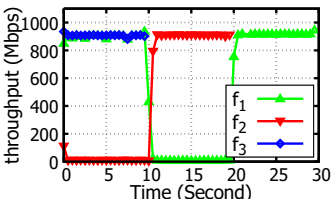


Fig. 9. [Testbed] Multi-Hop thrupt

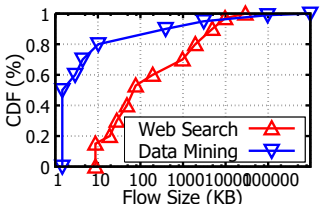


Fig. 10. Real traffic distribution

queues at the switch based on flow size. We have optimized the flow size thresholds of different priority queues using the model in [6]. TCP[α]Q means that it uses α switch queues.

pFabric: We implement all the features of pFabric in simulation, *e.g.* priority-based dequeue/drop at the switch and probing mode at the end host, based on the simulation code shared by pFabric authors.

All the schemes are evaluated both in testbed and simulation, except for pFabric not in testbed because it cannot be implemented using commodity switches.

B. Testbed micro-benchmark

Using the topology shown in Fig.s 8, we demonstrate EPN's ability to coordinate between switches and to seamlessly switch between flows. We test EPN via the example we used in Figure 3. At time 0, we simultaneously start 3 flows f_1, f_2, f_3 traversing paths as shown in Figure 8. The three flows has DP relation as $f_1 < f_2 < f_3$. We run f_1 for 30s, f_2 for 20s and f_3 for 10s.

As shown in Figure 9, at start, f_2 is demoted to LOW queue by switch B (S_B) because it shares path with a higher priority flow f_3. The DSID of f_2 is set to S_B thus S_A won't promote f_2. As a result, f_1 can transmit at HIGH queue in parallel with f_3. After 10s, f_3 finishes, S_B immediately promotes f_2 to HIGH queue and ToR switch A (S_A) demotes f_1 to LOW. Later when f_2 finishes, S_A promotes f_1. The flow switching time is within one RTT of our testbed.

This micro-benchmark experiment demonstrates that EPN is able to schedule flows strictly according to their DPs and ensures fast and seamless flow switching.

C. Testbed results under realistic traffic trace

We now use our 1Gbps testbed (Figure 8) to evaluate EPN's ability to minimize FCT in both information-aware and information-agnostic scenarios. We developed a query/response application with the client running on host h_{11}. The server applications are installed on 10 other hosts (h_1∼h_{10}) under the same rack. The client periodically generates queries to fetch a certain amount size of data from one of the servers. When received the query, the server sends out the requested data to the client as a response. The query data size is sampled from traffic distribution generated from real datacenters, namely web search [5] and data mining workload [19], as shown in Figure 10. The inter-arrival time of the queries is sampled according to a poisson distribution. We vary the load from 0.4 to 0.8 by controlling the inter-arrival time.

In this experiment, we configure 2/8 priority queues for PIAS and tune the per-port ECN threshold to be 40 data packets, which gives PIAS its best performance on our testbed. We use the thresholds shared by the authors of PIAS. For TCP[α]Q, switch configuration is the same as PIAS. And the priority queue thresholds are optimized according to the analysis model in [6].

In both senarios, EPN defines the tiny flows to be under 69KB. EPN uses per-queue ECN marking with an ECN threshold to be 40 packets.

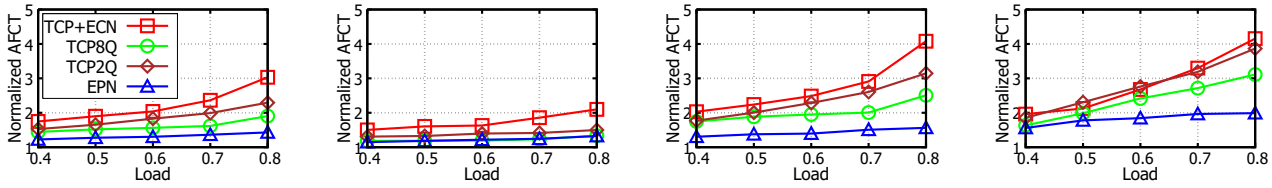
Information-Aware Scenario: We first evaluate the information-aware scenario, *i.e.* flow size information known to the DP allocator. Fig. 11(a) and Fig. 12(a) show the average FCT of all flows under two workloads respectively. Overall, EPN achieves 24.6%∼60.5% and 13.3%∼56.1% lower average FCT than TCP2Q from load 0.4 to 0.8 under web search and data mining workloads, respectively. Using more queues helps TCP8Q to achieve a better performance than TCP2Q. However, it's FCT is still 18.0%∼33.2% and 6.1%∼35.1% longer than EPN under the two workloads, respectively.

We now break down the results by size to reveal where EPN's gain comes from. We show the FCT performance for small flows ($\leq$100KB), middle flows (100KB∼10MB) and large flows (>10MB) in the first three sub-figures of Fig. 11 and Fig. 12.

For small flows, EPN has 12.0%∼14.0% and 6.2%∼45.9% lower FCT than TCP2Q at various loads under the two workloads, respectively. Even compared to TCP8Q with a well-optimized threshold setting, EPN still has -2.1%∼3.2% and 2.0%∼34.5% lower FCT under the web search and data mining workloads, respectively. EPN improves the performance of TCP[α]Q due to two reasons: 1) Flow sharing in TCP[α]Q still frequently happens with a small number of priority queues using a fixed flow size threshold setting, while EPN can dynamically mapping flows with different sizes into different queues. Although a well-optimized threshold in a few certain scenarios (*e.g.*, web search) can bring TCP8Q a good performance (even slightly better than EPN), this fixed setting manner still greatly falls short in other workloads such as data mining. 2) The per-port ECN marking scheme used in TCP[α]Q will make high priority flows slow down its sending rate when low priority flows build up queue in network, while EPN uses per-queue ECN marking (see §V-C for more discussion).

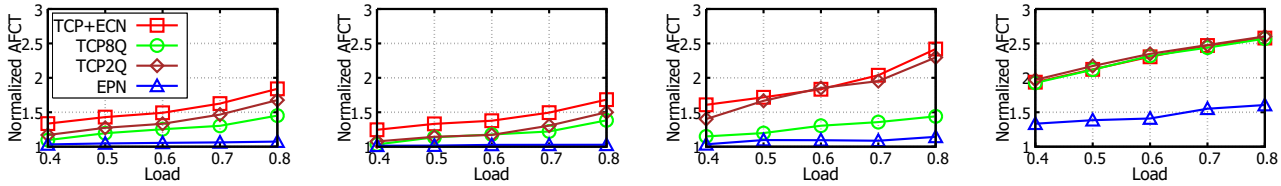
For middle and large flows with larger sizes, there is much richer time than small flows for EPN to conduct fine-grained serialization scheduling. Thus, EPN performs much better than other schemes for these flows. Specifically, for middle flows, EPN achieves 33.2%∼101.2% and 9.2%∼100.1% lower FCT than TCP2Q and TCP8Q under the two workloads at various loads, respectively. And for large flows, EPN's FCT is 4.8%∼94.2% and 44.4%∼66.6% lower than others.

Information-Agnostic Scenario: Then we evaluate the information-agnostic scenario, *i.e.* flow size information unknown to the DP allocator. Information-agnostic scheduling can be enabled naturally on EPN by using MLFQ as flow



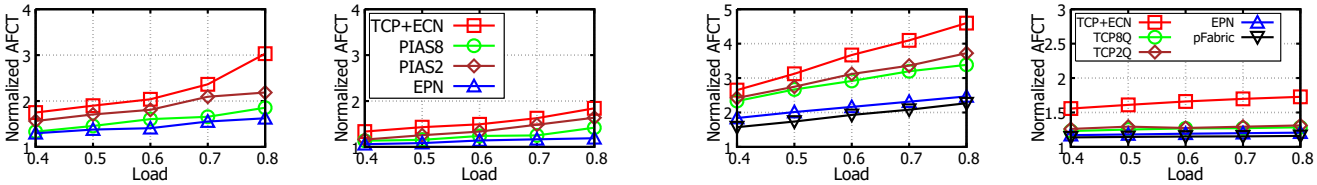
(a) Flow size: Overall (b) Flow size: $\leq 100KB$ (c) Flow size: $100KB \sim 10MB$ (d) Flow size: $> 10MB$

Fig. 11. [Testbed] Information-aware FCT performance under web search workload.



(a) Flow size: Overall (b) Flow size: $\leq 100KB$ (c) Flow size: $100KB \sim 10MB$ (d) Flow size: $> 10MB$

Fig. 12. [Testbed] Information-aware FCT performance under data mining workload.



(a) Web Search: w/o Info Overall (b) Data Mining: w/o Info Overall

Fig. 13. [Testbed] Information-agnostic scenario overall FCT.

(a) Web Search: w/ Info Overall (b) Data Mining: w/ Info Overall

Fig. 14. [Simulation] Information-aware scenario overall FCT.

DP setting paradigm (details in §IV-C). We use 16 MLFQ levels with an exponential threshold like in [21] and [13]. The base threshold is set to be 200KB and an exponential factor is 2. The threshold setting is the same for different workloads. We compare EPN with PIAS, which is the latest work on scheduling flows without size information. For PIAS, we configure 2 and 8 priority queues and use the thresholds setting shared by the authors.

For the space limitation, we only show the overall performance of all flows in Fig. 13, and omit the results of breakdown by the flow size. Thanks to the multiple MLFQ levels, it enables EPN to achieve finer grained flow size estimation as well as the scheduling, and makes EPN significantly outperform other schemes in this scenario. Specifically, for web search and data mining workload, EPN achieves 20.5%∼34.5% and 9.5%∼37.7% lower FCT than PIAS with 2 priority queues under the two workloads, respectively. More priority queues can give PIAS better performance, however, EPN still outperforms PIAS8 by 2.7%∼14.1% and 6.5%∼19.3% under the two workloads. Notice that, in both workloads, EPN's threshold setting remains the same. This verifies our previous claim that when there are enough MLFQ levels, a single exponential threshold setting in EPN can fit a wide range of workload distribution. In other word, EPN is able to achieve distribution-agnostic scheduling.

In conclusion, EPN is general enough to support both information-aware and information-agnostic flow scheduling under realistic traffic, and performs better than other existing practical solutions.

D. Large Scale Simulation

Simulation Setup: The topology used in our NS3 [2] simulation is a 128-server 10Gbps FatTree [3] with a base fabric RTT of 55.2us. We use ECMP [32] as the underlying load balance scheme. Again we evaluate all the schemes under web search [5] and data mining workload [19] from real datacenters. EPN treats flows with size less than one BDP (69KB in this experiment) as tiny flows.

Information-aware performance: As seen in Fig. 14, EPN outperforms other practical solutions (TCP2Q and TCP8Q) and performs close to pFabric. Specifically, under web search workload, EPN's overall FCT is 31.7%∼50.8% lower than TCP2Q and 26.8%∼38.1% lower than TCP8Q for various load. Compared with pFabric, EPN has 8.0%∼14.4% longer FCT. Under Data mining workload, EPN performs 7.0%∼9.6% and 5.1%∼6.5% better than TCP2Q and TCP8Q for various load respectively. EPN achieves only 2.7%∼3.87% longer FCT than pFabric.

We have also evaluated the information-agnostic scenario, but omit them due to space limitation. EPN performs better than other schemes, with 8.8%∼20.7% and 3.9%∼6.9% lower FCT than PIAS2 and PIAS8 under the two workloads.

In conclusion, EPN is able to scale to larger topologies and higher bandwidth fabrics.

VII. Related Work

Besides PDQ [23], pFabric [6] and PIAS [7] which we have deeply discussed before, there are also other rich related works in flow scheduling [18, 29, 34]. Due to space limitation, we are not able to cover them all. Here we only discuss those that are closely related to EPN but not covered before.

PASE [27] proposes a framework for flow scheduling that combines self-adjusting end-host, in-network prioritization and arbitration. PASE uses *designated servers* as arbitrators for network links. A scheduling decision is made through the communication among arbitrators. On the contrary, EPN

aims to support fine-grained priority-based flow scheduling on *commodity switches*. In this paper, we demonstrate the implementation feasibility of controller logic on commodity switches. Moreover, unlike PASE, EPN does not introduce communication among switches or modification to transport.

QJUMP [20] tries to provide bounded latency for small flows and high throughput for large flows. This is orthogonal to EPN aiming at fine-grained priority-based flow scheduling.

Recently, NUMFabric [28] provides flexible and fast bandwidth allocation control. It requires programmable priority queue support from switches. Contrarily, EPN aims to enable flow scheduling via commodity switches.

Finally, other task-aware [12, 14, 17] scheduling work try to minimize average task or Coflow completion time. Also, recently there appears Karuna [11] that studies scheduling under Mix-Flow (flows with and without deadlines) scenario. In this paper, we only focus on priority-based flow scheduling under non-Mix-Flow scenario. Extending EPN to be task-aware or Mix-Flow-applicable is our future work.

VIII. Conclusion

This paper presents Explicit Priority Notification (EPN), a priority-based flow scheduling mechanism for minimizing FCT in datacenter networks. By adding merely one additional priority queue, EPN can schedule flows with millions of fine-grained priorities. EPN is general and can support various priority-based flow scheduling disciplines with or without flow size information. We have implemented EPN on commodity switch and evaluate its performance with both testbed experiments and extensive simulations. Our results show that EPN achieves comparable FCT to pFabric which is a clean-slate design. But when the number of priority queues are limited, EPN greatly outperforms other related work in both information-aware and information-agnostic settings. To the best of our knowledge, EPN is the first system that provides millions of priorities for flow scheduling with commodity switch hardware.

Acknowledgements

We thank Yongguang Zhang, Ming Zhang and Kai Chen for their valuable suggestions, and anonymous reviewers of INFOCOM for their helpful comments. We also thank Wei Bai and Hao Wang for their help on paper writing and evaluation.

References

[1] Broadcom corporation, bcm5324m datasheet. https://www.broadcom.com/collateral/pb/5324M-PB01-R.pdf.

[2] Ns3. https://www.nsnam.org/.

[3] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In *SIGCOMM 2008*.

[4] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, V. T. Lam, F. Matus, R. Pan, N. Yadav, and G. Varghese. Conga: Distributed congestion-aware load balancing for datacenters. In *SIGCOMM 2014*.

[5] 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*.

[6] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker. pfabric: Minimal near-optimal datacenter transport. In *SIGCOMM 2013*.

[7] W. Bai, L. Chen, K. Chen, D. Han, C. Tian, and H. Wang. Information-agnostic flow scheduling for commodity data centers. In *NSDI 2015*.

[8] W. Bai, L. Chen, K. Chen, and H. Wu. Enabling ecn in multi-service multi-queue data centers. In *NSDI 2016*.

[9] A. Bar-Noy, M. M. Halldórsson, G. Kortsarz, R. Salman, and H. Shachnai. Sum multicoloring of graphs. *Journal of Algorithms*, 37(2):422–450, 2000.

[10] J. Cao, R. Xia, P. Yang, C. Guo, G. Lu, L. Yuan, Y. Zheng, H. Wu, Y. Xiong, and D. Maltz. Per-packet load-balanced, low-latency routing for clos-based data center networks. In *CoNEXT 2013*.

[11] L. Chen, K. Chen, W. Bai, and M. Alizadeh. Scheduling mix-flows in commodity datacenters with karuna. In *SIGCOMM 2016*.

[12] M. Chowdhury and I. Stoica. Coflow: An application layer abstraction for cluster networking. In *Hotnets 2012*.

[13] M. Chowdhury and I. Stoica. Efficient coflow scheduling without prior knowledge. In *SIGCOMM 2015*.

[14] M. Chowdhury, Y. Zhong, and I. Stoica. Efficient coflow scheduling with varys. In *SIGCOMM 2014*.

[15] B. Davie and J. Gross. Stt. *draft-davie-stt-04*, 2013.

[16] A. Dixit, P. Prakash, Y. C. Hu, and R. R. Kompella. On the impact of packet spraying in data center networks. In *INFOCOM 2013*.

[17] F. R. Dogar, T. Karagiannis, H. Ballani, and A. Rowstron. Decentralized task-aware scheduling for data center networks. In *SIGCOMM 2014*.

[18] P. X. Gao, A. Narayan, G. Kumar, R. Agarwal, S. Ratnasamy, and S. Shenker. phost: Distributed near-optimal datacenter transport over commodity network fabric.

[19] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. Vl2: A scalable and flexible data center network. In *SIGCOMM '09*, 2009.

[20] M. P. Grosvenor, M. Schwarzkopf, I. Gog, R. N. Watson, A. W. Moore, S. Hand, and J. Crowcroft. Queues don't matter when you can jump them! In *NSDI 2015*.

[21] C. Guo, W. Zhu, Z. Zhang, and Z.-L. Zhang. Utilizing the diversities and invariants of the internet to support multi-services. Technical report, Technical Report MSR-TR-2004-46, Microsoft Research, 2004.

[22] K. He, E. Rozner, K. Agarwal, W. Felter, J. Carter, and A. Akella. Presto: Edge-based load balancing for fast datacenter networks. In *SIGCOMM 2016*.

[23] C.-Y. Hong, M. Caesar, and P. B. Godfrey. Finishing flows quickly with preemptive scheduling. In *SIGCOMM 2012*.

[24] G. Judd. Attaining the promise and avoiding the pitfalls of tcp in the datacenter. In *NSDI 2015*.

[25] K.-C. Leung, V. O. Li, and D. Yang. An overview of packet reordering in transmission control protocol (tcp): problems, solutions, and challenges. *IEEE ToPDS*, 18(4):522–535, 2007.

[26] M. Mahalingam, D. Dutt, K. Duda, P. Agarwal, L. Kreeger, T. Sridhar, M. Bursell, and C. Wright. VXLAN. *IETF Internet Draft.*, 2013.

[27] A. Munir, G. Baig, S. M. Irteza, I. A. Qazi, A. X. Liu, and F. R. Dogar. Friends, not foes: Synthesizing existing transport strategies for data center networks. In *SIGCOMM 2014*.

[28] K. Nagaraj, D. Bharadia, H. Mao, S. Chinchali, M. Alizadeh, and S. Katti. Numfabric: Fast and flexible bandwidth allocation in datacenters. In *SIGCOMM 2016*.

[29] J. Perry, A. Ousterhout, H. Balakrishnan, D. Shah, and H. Fugal. Fastpass: A centralized zero-queue datacenter network.

[30] K. Ramakrishnan, S. Floyd, D. Black, et al. The addition of explicit congestion notification (ecn) to ip. In *RFC 3168, September*, 2001.

[31] M. Sridharan, A. Greenberg, Y. Wang, P. Garg, N. Venkataramiah, K. Duda, I. Ganga, G. Lin, M. Pearson, P. Thaler, and C. Tumuluri. NVGRE. *IETF Internet Draft.*, 2013.

[32] D. Thaler. Multipath issues in unicast and multicast next-hop selection. In *RFC 2991, Nov. 2000*.

[33] B. Vamanan, J. Hasan, and T. Vijaykumar. Deadline-aware datacenter tcp (d2tcp). In *SIGCOMM 2012*.

[34] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron. Better never than late: Meeting deadlines in datacenter networks. In *SIGCOMM 2011*.

[35] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley. Design, implementation and evaluation of congestion control for multipath tcp. In *NSDI 2011*.

[36] H. Wu, J. Ju, G. Lu, C. Guo, Y. Xiong, and Y. Zhang. Tuning ECN for Data Center Networks. In *CoNEXT 2012*.

[37] Y. Zhu, N. Kang, J. Cao, A. Greenberg, G. Lu, R. Mahajan, D. Maltz, L. Yuan, M. Zhang, B. Y. Zhao, et al. Packet-level telemetry in large datacenter networks. In *SIGCOMM 2015*.