A DRAM-friendly priority queue Internet packet scheduler implementation and its effects on TCP.

Katsushi Kobayashi
The University of Tokyo
Tokyo, JAPAN
ikob@acm.org

Abstract—To meet various latency requirements for network applications, latency aware packet schedulers that respect per-packet deadlines requested by end systems, such as least slack-time first (LSTF), have emerged. Even though latency aware schedulers are beneficial, few issues remain. First, it is not compatible with best-effort networks, such as the Internet, because of infinite buffer delays by keeping every packet without discarding even in congestion. Second, its implementation is challenging with increasing network bandwidths due to memory device restrictions, i.e., DRAM access latencies. For the congestion issue, we presented earliest deadline first with reneging (EDFR), which can be applied to best-effort services by taking advantage of its packet drop feature.

This paper discusses EDFR scheduler implementations on FPGA and its impact on TCP stack with real systems. We designed and implemented skip-FIFO based EDFR onto FPGA, which performed to take advantage of DRAM memory bandwidth, such as 75% of the theoretical limit of High Bandwidth Memory (HBM). The TCP behaviors on EDFR were almost unchanged in terms of throughputs and losses, even for coexisting flows requesting different latencies.

Index Terms—networking, internet, TCP

I. Introduction

Packet buffer delay of Internet routers is a significant issue because it impacts user experience with applications. Latency sensitive network applications, such as interactive Voice/Video communication and online games, prefer smaller buffer sizes to reduce network latency as low as possible. In contrast, some other applications, which transfer large data, prefer deep packet buffer sizes such as bandwidth-delay product (BDP), well-known as the rule of thumb that maximizes single flow TCP throughputs, even increasing end-to-end latencies. On Internet routers, a single FIFO packet buffer is shared among all applications' flows, and its sizes follows the rule of thumb. Latency tolerant applications' packets produce a queue build-up to maximize their own throughput, and they block all other packets, including latency-sensitive ones. As a result, latency-sensitive packets miss their deadline requirements, and the application quality is degraded.

Several QoS frameworks, such as IntServ and Diffserv have been developed to meet various application requirements, including latencies. However, such frameworks have not taken a market share from simple best-effort services because they rely on dedicated network resource provisioning for each QoS requirement. Dedicated resource provisioning also requires an expensive economic infrastructure in addition to network infrastructure. Thus far, building such economic infrastructure has not been a better investment than increasing the bandwidth of best-effort services [1]. Therefore, when designing a future Internet to satisfy latency requirements, it must work without economic infrastructure, or in other words, within a best-effort network service. In addition, Deterministic Networking (DetNet) standardization is in progress, which shares the same objectives with ours, such as bounded latency. However, unlike our study, DetNet focuses on networks under a small closed group control, not for large groups like the Internet.

On a best-effort network service, link overload conditions or congestions have to be considered. On the Internet, congestion always occurs when the traffic is concentrated on a specific network location, such as popular Web services and distributed denial of service (DDoS) attacks. As the time, location, and frequency of such congestion cannot be predicted, dropping packets is the only feasible solution. When an existing FIFO packet scheduler receives a packet beyond its buffer capacity, it either drops that packet or some other packet/packets in a simple tail-drop or active queue management (AQM) manner [2]. Further, when designing a new packet scheduler, the endpoint transport behavior should also be discussed because it copes with congestion using packet drop and/or delay as a signal.

Priority queue packet schedulers, such as Earliest Deadline First (EDF) and Least Slack Time First (LSTF), can support various latency requirements [3], [4]. However, such priority queue approaches can increase the buffer delay to infinity because they keep all buffered packets even in case of congestion. To address congestion, we proposed EDF with reneging (EDFR) packet scheduler that avoids a significant buffer latency [5]. We also present that the impact of EDFR on existing TCP transports is insignificant by performing ns-2 simulations.

From a practical viewpoint, the feasibility of a packet scheduler algorithm depends on available device technologies, especially in memory. DRAM is the only practical choice for packet buffers owing to its capacity advantages. As the network link bandwidth grows, the required packet

JSPS KAKENHI Grant Number 18K11256

Annex to ISBN 978-3-903176-28-7

buffer size increases. For instance, when applying the rule of thumb of buffer size to a router accommodating 24×100 Gbps ports with 100 ms $\overline{RTT}$, a 30 GB buffer capacity, is required. Such buffer sizes cannot be realized without DRAM in the current technology [6]. In addition, even though DRAM is the only choice, ordinary priority queue algorithms, such as heap and skip-list, are inappropriate for it because of their random access nature. Therefore, a DRAM-friendly priority queue implementation is expected.

The primary objective of this study is to examine the feasibility of the latency support mechanism realized by EDFR scheduler. To this end, first, we present the EDFR scheduler architecture and its FPGA implementations that can take advantage of state-of-art memory technology such as high bandwidth memory (HBM). Second, we present that TCP transport behaviors are almost unchanged except for latencies using real end systems, to show not to require significant changes in transport and applications. This paper is organized as follows: In section II, the network architecture and EDFR scheduler are outlined. In section III, the skip-FIFO approach and its FPGA implementations are elucidated. In section IV, the TCP behaviors are presented with our Skip-FIFO implementations. Section V contains a review of related research. Finally, we conclude our discussions in section VI.

II. Network architecture and EDFR scheduler

This study proposes a latency aware network architecture that enables different latency requirements from various applications without QoS provisioning. The architecture supports generic use cases to avoid excessive delay of latency-sensitive applications packets, such not only as VoIP, but also as an interactive Web, from a queue build-up. Queue build-ups are caused not only by traditional data transfer but also by emerging streaming video applications. For instance, streaming video applications send as much data as possible at the start of playing the video to reduce the time lag between choosing content and its actual starting. This is because streaming applications do not start a video until sufficient data is buffered to avoid disruptions in playback. In short, one buffer size cannot fit all applications.

On our latency aware network architecture, applications specify their per-packet latency requirements, or deadlines, into packet headers such as IP ToS or DSCP field. Subsequently, routers forward the packets in an order based on per-packet deadlines by a priority queue. This design is similar to DiffServ in terms of the relation between DSCP and PHB. However, DiffServ intends to provide service discrimination on the Internet, but our architecture to satisfy latency requirements without dividing network resources, in other words, within a simple best-effort service.

On best-effort packet networks, packet drop must be accounted for because traffic overload can occur at any time. While ordinary EDF keeps all incoming packets without discard as FIFO having infinite capacity, EDFR drops a drops packets when a packet misses its deadlines. Such a drop policy might be reasonable for the existing network applications and transports. For instance, on VoIP, even if a packet is received after a playout deadline, it is simply discarded by the receiver application. On TCP, when a TCP sender does not receive an acknowledgment until retransmission timeout (RTO), it regards the corresponding packet as discarded and retransmits the packet.

Packet drop behavior on priority queues has been well studied on EDFR. EDFR has two significant properties in terms of drop rates. First, even though incoming packets' deadlines on EDFR are distributed, the complete drop rates depend only on the mean deadlines of incoming packets. Specifically, the complete drop rates are approximated by those of limited FIFO in which the buffer size is the mean deadline. Second, in an EDFR system, the drop rates of even different deadlines are equal or fair. This is because packet drop decisions are made based on the remaining deadlines at that time, not based on the deadlines set at incoming. In addition, a limited FIFO scheduler is a special case of EDFR in which all packets have the same deadline corresponding to the buffer size of limited FIFO.

According to ns-2 simulations, the TCP behaviors are almost unchanged when an ordinary FIFO scheduler is replaced with EDFR. In other words, existing network applications, most of which are based on TCP, are able to work on EDFR based networks as well as on FIFO based existing networks on the Internet.

III. Skip-FIFO design and implementations

While DRAM is the only choice for the packet buffer due to its capacity, recent DRAMs have been improved not in terms of latencies, but also in terms of access throughputs. For instance, on HBM, the throughputs on random memory access are three times lower than those on sequential memory access [7].

Ring buffer, an ordinary FIFO scheduler implementation, can take advantage of the exiting DRAM bandwidths accelerated using burst and pipelined access. Although emerging flexible packet scheduler frameworks, such as PIFO and PIEO, are priority queue implementations, they are not compatible with DRAM architecture because of their per packet random access nature. In PIFO, the memory locations where packets will be pushed in, or written, are not predictable, and its access granularity is very small. In the case of a 64-byte packet on a 10 Gbps link and 512-bit bus, both the enqueue and dequeue actions must be finished within a 51 ns period. In short, typical DRAM access latency of ∼100 ns or more is too slow for such priority implementations.

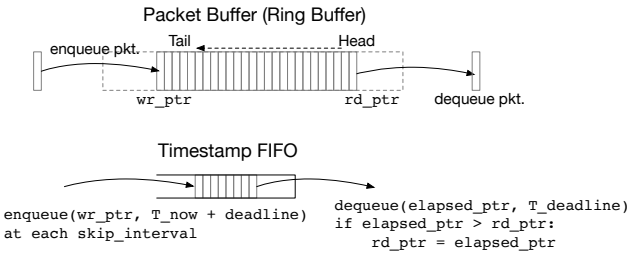


Figure 1: skip-FIFO

Although an unlimited number of service classes can be provided by generic priority queues, it is clearly infeasible from the viewpoint of protocol design. Because the service class identifier in the IP header has limited space, i.e., 8 bits for ToS and 6 bits for DSCP, such a small space cannot provide high-number classes. For instance, in the case of 1 ms granularity, which corresponds to the default minimal CPU scheduler tick interval on Linux, 8-bit space can represent up to 256 ms deadlines. Such deadlines are adequate to cover worldwide communications.

Therefore, to maximize throughputs with DRAM, we chose the naive FIFO priority queue. A FIFO priority queue aggregates ordinary FIFO queues, such as ring buffer, with a priority encoder. The number of FIFO queues is equal to that of the supported deadlines. When a packet is received, it is enqueued into a corresponding queue with its deadline as metadata. When dequeuing, the priority encoder takes packets from the queue having the earliest deadline at its top.

FIFO priority queues can realize an EDFR packet scheduler with its minor modification. That is, when a dequeued packet has already elapsed its deadline, the priority encoder drops it. However, such implementations consume memory bandwidth with dropping packets. Such bandwidth resource wastage is a critical hinderance for high-speed networks in which the memory and network performance limits are comparable.

To overcome these limitations, we designed the skip-FIFO packet scheduler(Fig. 1). A skip-FIFO comprises of two FIFOs: one is a ring buffer for accommodating packet data, and another is a timestamp FIFO for recording packet pointers and the corresponding deadlines. The ring buffer FIFO is an ordinal one that consists of address mapped memory and two address indices, i.e., the write- and read-pointers, abbreviated as wr_ptr and rd_ptr, respectively. When enqueuing data, a ring buffer writes the data into mapped memory sequentially from wr_ptr, and it advances the wr_ptr by the data size. When dequeuing, a buffer reads data from the rd_ptr to the wr_ptr, and it updates the rd_ptr.

Simultaneously with the ring buffer operations, the timestamp FIFO stores the wr_ptr of the ring-buffer for the start index of the latest packet, and its deadline timestamp as $T_{now} + deadline$ at every skip tick interval, where T_{now} is the current time. On the dequeue port, the timestamp FIFO outputs data named as elapsed_ptr and $T_{deadline}$. If the elapsed_ptr is "newer" than the rd_ptr, the rd_ptr is updated by it. As a result, the deadline elapsed packets in the ring-buffer are skipped or discarded. In addition, if $T_{deadline}$ is exceeded, or if the elapsed_ptr is "older" than the rd_ptr, the timestamp FIFO can be dequeued. In our implementations, the depth of the timestamp FIFO was fixed at 14 bits, and the skip-tick interval was $200ms/2^{14}$, or 12μs.

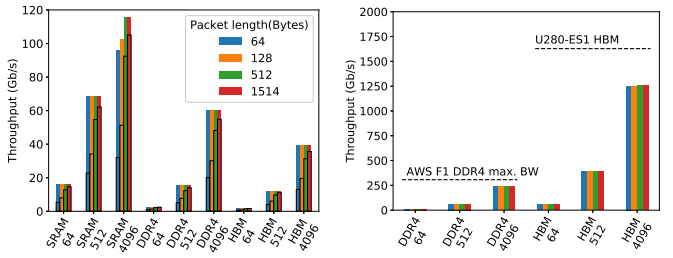
Because skip-FIFO based EDFR is an approximation of ideal EDFR, it has several system inaccuracies. In particular, the accuracy of the timestamp FIFO depends on the skip-interval. Skip interval on this study provides more than a thousand times accuracy than modern AQM such as PIE, CoDel which suggest 10-15 ms as the update interval of the drop probability [8], [9]. Thus, the inaccuracy on skip-FIFO caused by its timestamp is expected insignificant. Furthermore, even if the inaccuracy causes a significant impact, it can be improved by the priority encoder arbitrating skip-FIFO outputs. For example, the priority encoder discards deadline elapsed packets in addition to skip-FIFO, which can offset the inaccuracy dealing with the bandwidth wastage by dequeuing packets.

We implemented an EDFR algorithm on top of Skip-FIFO on two generations of FPGA families: one is 28 nm FPGA (Xilinx Kintex7) on NetFPGA-CML composed of 512 MB DDR3 DRAM, and four Gigabit Ethernet (GbE) interfaces; the other is 16 nm FPGA (Vertex Ultra+) on AWS-F1 of 64 GB DDR4 DRAM, and on Xilinx Aveo U280-ES of 8 GB HBM [10], [11], [7].

The two implementations were done with different interests. On Kintex7, we conducted network transport evaluations using the physical ethernet switch hardware. On the other hand, on Vertex Ultra+, we evaluated the potential of Skip-FIFO throughputs without physical interface; in other words, we used the platforms as a testbench. We used a NetFPGA reference ethernet switch as our RTL implementation basis by replacing internal SRAM (BRAM) based FIFO scheduler with DRAM=based EDFR.

The resource utilization of our implementations are approximately 1700 LUTs and 10 BRAMs, or 20% more in LUTSs and five times in BRAMs than those of ordinary FIFO. Even though the observed BRAM resource usage of the proposed architecture is in multiples of that of ordinary FIFO, it is not critical for the skip-FIFO architecture. This is because such multiplications were attributed to small FIFO components accommodating deadline timestamp and packet head indices. The BRAM utilization depends on the interval of storing packet index and deadline, not on bandwidth or the ring buffer capacities. Therefore, the BRAM utilization can be compromised with the FPGA resource and the skip-interval.

Fig 2 shows the throughputs of the skip-FIFO scheduler on the Vertex Ultra+ FPGA system, where the clock



(a) Single channel (b) Entire system

Figure 2: Skip-FIFO bandwidth throughputs.(a) Single channel throughputs. The edged bar areas eliminate metadata overhead.(b) Entire system throughputs aggregating available memory channels. The dotted lines represent theoretical memory bandwidth limits.

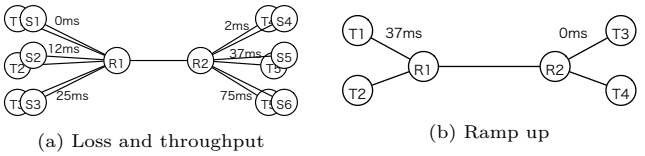
speed was 250 MHz. The scheduler latency with a 1514-byte packet and 4096-byte AXI-MM burst size was approximately 190 clocks on DDR4, which is approximately three times more than that on BRAM at 64 clocks. Overall, the burst length impacts for the throughputs are clear regardless of memory types such as HBM, DRAM, and BRAM: the shorter the burst length was, the worse the FIFO throughput was. Furthermore, the shorter the packet length was, the worse the throughputs were; this throughput degradation was caused by the data encoding format as edged bars in Fig 2(a). In particular, in the case of a 64-byte packet, such packets can be encoded into just one clock cycle on 512-bit data width; however, the entire data, including the 128-bit tuser bus and the 8-bit control lines, required three clock cycles.

The best throughput, including metadata, was 115 Gbps of BRAM, 60 Gbps of DDR4, and 39 Gbps of HBM. While the throughput of DDR4 was better than that of HBM, the entire HBM throughput will be 1.25 Tbps, or 156 GBytes/s by aggregating with 16 pseudo channels, which overcomes the 220 Gbps with four-channel DDR4. Because a scheduler throughput includes bidirectional memory accesses, the entire HBM throughput achieved 76% of the theoretical limit of 410 GBytes/s. In summary, the above results show that skip-FIFO is a DRAM-friendly priority queue that can take advantage of the DRAM bandwidth throughputs regardless of packet sizes.

IV. TCP traffic behaviors with EDFR

In this section, the impact of the EDFR algorithm on existing transports is discussed based on real-system evaluations.

Two NetFPGA-CML ethernet switches that performed the wire rates of GbE were used for evaluating the TCP traffic. The switches were connected using one of the four GbE interfaces. The rest of the interfaces, six in total, were connected to dedicated Docker nodes on an Ubuntu 18.04 Linux server. Three EDFR schedulers aggregating skip-FIFOs were implemented onto the connected interfaces;



(a) Loss and throughput (b) Ramp up

Figure 3: Simulation topologies. (a) A 6×6 dumbbell topology that comprises two 3×3 flow groups, i.e., (T1...T6) for testing, and (R1...R6) for reference. All links have a capacity of 1 Gbps. The dumbbell in (b) has two pairs of nodes, (T1,T2) and (T2,T3).

thus, up to three deadline classes can be supported. For comparison, we implemented ordinary FIFO tail drop schedulers with a round-robin arbiter in which the buffer sizes of each FIFO were set to the average RTT among all flows on each experiment.

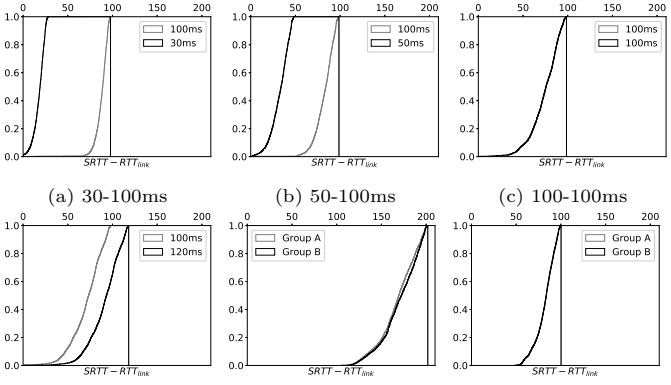
Flowgrind, which is used as the TCP traffic generator, was running on each Docker node [12]. For emulating link propagation delays, the Linux netem framework was used [13]. As the delays propagated by netem were only applied to the egress traffic, doubled delays were set at physical GbE ports. Note that the software-based netem emulator is less accurate than FPGA based one, but we used netem because it can provide sufficient accuracy for the link delays.

A. Latencies, losses, and throughputs with mixed RTT flows

Fig. 3(a) shows the network topology for evaluating TCP behaviors with different deadline flows on the EDFR algorithm. In order to compare our ns-2 results, the topology followed Latency specific experiments in the TCP evaluation suite except for the delay of backbone [14].

Pairs of 3×3 TCP flow groups in which each group had two different deadlines were generated. The deadline of one flow group was varied from 30 to 120 ms, and that of the other was fixed at 100 ms, which approximately corresponds with the average RTT among the nine connections. While real traffic traces are recommended by the TCP evaluation suite, due to restrictions of measurement tools, we used long-lived flows for overloaded conditions and 3GPP HTTP model traffic for moderate traffic load instead [15]. In order to close a steady-state, the data from the first 100 s were omitted; then, data from the subsequent 500 s were used. It should be noted that because the traffic is unidirectional, the buffer delays could be up to twice as long when the reverse traffic is accounted.

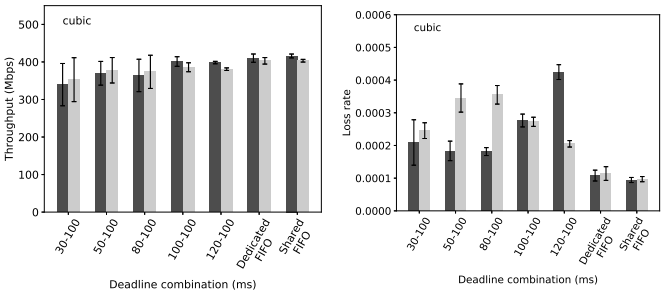
Fig. 4 shows the CDF of buffering delay obtained as $sRTT - RTT_{link}$ among TCP flows on every second, where $sRTT$ is smoothed RTT measured by TCP stack, and RTT_{link} is the cumulative link propagation delay for each flow. In order to focus on the buffering delay, long-lived TCP flows were generated to create heavily congested conditions. On all deadline combinations, all buffer delays clearly fell within the requested delay ranges. This reveals



(a) 30-100ms (b) 50-100ms (c) 100-100ms

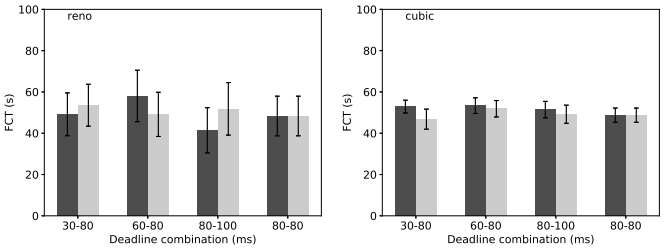
(d) 100-120ms (e) Dedicated FIFOs (f) Shared FIFO

Figure 4: EDFR scheduler delay distributions with different deadline flow combinations on CUBIC.



(a) Throughputs (b) Loss rates

Figure 5: Average CUBIC throughputs and loss rates in different deadline flow combinations on EDFR with the 3GPP HTTP traffic model. Each pair of bars represents the deadline combinations labeled by the x-axis, and the left dark bars represent the shorter deadlines of each pair. For example, in the case of labeled 30-100, the deadlines are 30ms for the left and 100ms for the right bar. Error bars show 95% confidence intervals.



(a) NewReno (b) CUBIC

Figure 6: Average flow Completion Time (FCT) at 1.5GB when competing two long-lived TCP flows requesting different deadlines. The bars represent the FCTs of the 2nd flows that start 100s after the 1st one. Each pair of bars represents the deadline combinations labeled by the x-axis, and the left dark bars represent the shorter deadlines of each pair. Error bars show 95% confidence intervals.

that per-packet deadlines were realized by the EDFR algorithms.

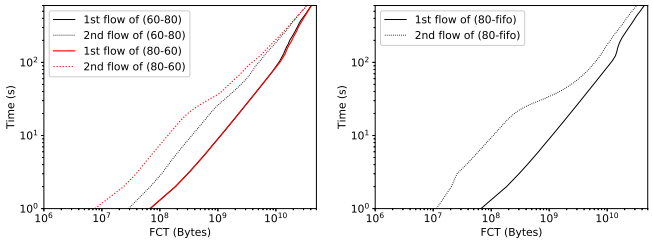
Fig. 4(e),(f), show the delays for ordinary FIFO: (e) shows two dedicated FIFOs assigned to different ToS classes, and (f) shows one FIFO shared by the traffic. As seen in (e), because two FIFO queues blocked each other, the delay increased to 200 ms. However, while 100-100 (c) was provided with two dedicated EDFR schedulers, its CDF was similar to that of shared FIFO (f) but dissimilar to that of the dedicated FIFOs (e). Such similarity reveals that even if an entire EDFR scheduler comprises two or more skip-FIFOs, it works like a single FIFO for each deadline flow.

Fig. 5 shows the entire throughputs (a) and packet loss rates (b) with 3GPP HTTP model traffic [15]. In order to compare with the simulation results with the real-traffic trace, the 3GPP model was used because it is more similar to real-traffic than long-lived TCP flows [5]. With regard to throughputs (Fig.5(a)), 80-90% bottleneck bandwidth capacities were consumed, and significant throughput differences were not found for all deadline combinations. Furthermore, with regard to packet loss rates (Fig. 5(b)), lower loss rates were observed for the shorter deadline flows, e.g., 50 in 50-100, than the longer ones. These loss rate differences disagreed with those of the ns-2 simulation results and an EDFR nature for which the same rates were observed even when incoming deadlines are distributed. However, the differences were not significant because the throughputs remained at the same levels.

B. FCT on competitive flows

Fig. 3(b) depicts the topology for evaluating the Flow Completion Time (FCT) on two competitive deadline flows. The topology followed the Ramp up time in TCP evaluation suite specification [14]. While the original specification recommends the addition of background traffic using 3×3 TCP connections, no background traffic was generated owing to the insufficient number of switch interfaces. We generated two long-lived TCP flows having different deadlines that can be represented as a pair of deadlines, such as 30-80. In all experiments, the flows of the second deadline were started 100 sec. after that of the first. We conducted FCT experiments 20 times for each deadline combination.

Fig. 6 shows the FCTs of 1.5 GB flow sizes in various deadline combinations. While the FCT results are scattered, overall, the FCTs are almost equal in all deadline combinations. Fig. 7 shows the typical FCT growth plots on EDFR and FIFO schedulers. Although all FCTs of 2nd flows (dotted lines) were worse than those of 1st (solid), all FCTs grew in steady as well as FIFO. In addition, the above FCT results agreed with those of corresponding ns-2 simulations [5]. In short, if existing flows consume link capacities, new TCP flows can get appropriate bandwidth on EDFR scheduler as well as FIFO.



(a) EDFR (b) FIFO

Figure 7: Flow Completion Time(FCT) plots with two TCP CUBIC flows competing (a) 60-80 ms deadline pair on EDFR, and (b) on FIFO.

As discussed in this section, the above experimental results show that the exiting TCP stacks will work well if ordinary FIFO schedulers are replaced with EDFR. Due to space limitation, we mainly show the TCP CUBIC results; however, most properties were fully retained with TCP Reno.

V. Related Work

Modern AQM algorithms, such as CoDeL and PIE, aim to reduce buffer delays to avoid queue build-up [8], [9]. In terms of focusing on best-effort networks, these goals are similar to those of this study. However, such an AQM cannot satisfy different latency requirements, unlike EDFR, because all flows share one FIFO scheduler. Furthermore, such AQM allows transient increasing latencies to compromise for the slow start of the TCP.

Least slack time first (LSTF) packet scheduler, which supports end-to-end latency, is another priority queue sharing the same goal, while this study focuses on per-hop latency supports. In contrast to other end-to-end latency support frameworks such as IntServ, it eliminates the propagation delay in prioritization on its scheduler. As a result, it is resilient against the changes in the propagation delay, such as on network path changes. The TCP behavior properties found in this study, i.e., loss and throughput, are fair to latency differences, which could also be applied to LSTF. This is because such TCP fairnesses are kept on multi-hop configurations according ns-2 simulation. Therefore, LSTF might be used in best-effort services such as the existing Internet.

VI. Conclusion

In this study, EDFR was implemented using skip-FIFO, which is a DRAM-friendly latency aware packet scheduler. The skip-FIFO based EDFR scheduler takes advantage of the DRAM bandwidth by using its sequential access characteristic. It also showed no significant impact on existing TCP traffic. This suggests that the EDFR algorithm has the potential to replace existing FIFO schedulers in routers.

To deploy the latency supports, updates are required for both applications and network routers, but these efforts and costs are not expensive due to the followings reasons. Applications can be adapted without major modification, since existing network socket API already have QoS interface that sets ToS or DSCP field. For routers, our latency aware architecture can contribute its cost by reducing buffer size. If a preferred buffer size is declared by every application, its aggradation is expected to be smaller than BDP. As a result, a packet buffer size can be reduced by fully satisfying the application requirements.

It should be noted that UDP-based transport, such as QUIC, is emerging for latency-sensitive applications, while we discussed on TCP behaviors [16]. Such emerging transport can utilize the knowledge brought from our TCP experiments since they mimic the congestion control of existing TCP.

References

[1] S. Floyd and M. Allman, "Comments on the Usefulness of Simple Best-Effort Traffic," RFC 5290, Internet Engineering Task Force, Jul. 2008. [Online]. Available: http://www.ietf.org/rfc/rfc5290.txt

[2] F. Baker, "Rfc1812: Requirements for ip version 4 routers," 1995.

[3] R. Mittal, R. Agarwal, S. Ratnasamy, and S. Shenker, "Universal packet scheduling," in Proceedings of the 14th ACM Workshop on Hot Topics in Networks, ser. HotNets-XIV. New York, NY, USA: ACM, 2015, pp. 24:1–24:7. [Online]. Available: http://doi.acm.org/10.1145/2834050.2834085

[4] V. Sivaraman, F. M. Chiussi, and M. Gerla, "Traffic shaping for end-to-end delay guarantees with edf scheduling," in 2000 Eighth International Workshop on Quality of Service. IWQoS 2000 (Cat. No. 00EX400). IEEE, 2000, pp. 10–18.

[5] K. Kobayashi, "Lawin: A latency-aware internet architecture for latency support on best-effort networks," in 2015 IEEE 16th International Conference on High Performance Switching and Routing (HPSR). IEEE, 2015, pp. 1–8.

[6] J. Hennessy and D. Patterson, Computer Architecture: A Quantitative Approach. Morgan Kaufmann 8. Textbook, 2018.

[7] Alveo U280 Data Center Accelerator Card User Guide, Xilinx, 2019.

[8] K. Nichols and V. Jacobson, "Controlling queue delay," Communications of the ACM, vol. 55, no. 7, pp. 42–50, 2012.

[9] R. Pan, P. Natarajan, C. Piglione, M. Prabhu, V. Subramanian, F. Baker, and B. V. Steeg, "PIE: A Lightweight Control Scheme To Address the Bufferbloat Problem," Working Draft, Internet-Draft draft-pan-tsvwg-pie-00.txt, Dec. 2012.

[10] "NetFPGA-CML," http://netfpga.org.

[11] Intel, "Official repository of the AWS EC2 FPGA Hardware and Software Development Kit," https://github.com/aws/aws-fpga/.

[12] A. Zimmermann, A. Hannemann, and T. Kosse, "Flowgrind-a new performance measurement tool," in 2010 IEEE Global Telecommunications Conference GLOBECOM 2010. IEEE, 2010, pp. 1–6.

[13] S. Hemminger et al., "Network emulation with netem," in Linux conf au, 2005, pp. 18–23.

[14] L. Andrew, S. Floyd, and G. Wang, "Common TCP Evaluation Suite," Working Draft, Internet-Draft draft-irtf-tmrg-tests-02.txt, 2009.

[15] "cdma2000 evaluation methodology revision a."

[16] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar et al., "The quic transport protocol: Design and internet-scale deployment," in Proceedings of the Conference of the ACM Special Interest Group on Data Communication, 2017, pp. 183–196.