REN: Receiver-driven Congestion Control using Explicit Notification for Data Center

Zhaoyi Li, *Member, IEEE,* Jiawei Huang, *Member, IEEE,* Jinbin Hu, *Member, IEEE,* Weihe Li, *Member, IEEE,* Tao Zhang, *Member, IEEE,* Jingling Liu, *Member, IEEE,* Jianxin Wang, *Senior Member, IEEE* and Tian He, *Fellow, ACM, IEEE*

Abstract—In recent years, receiver-driven transport protocols have been proposed to use proactive congestion control to meet the stringent latency requirements of large-scale applications in data center. However, the receiver-driven proposals face the challenges brought by network dynamic. Firstly, when the bursty flows start, the aggressive and blind line-rate transmission in the first RTT easily leads to persistent queue backlog. Secondly, when some flows finish transmissions, the remaining ones cannot increase their sending rates to seize the available bandwidth. To address these problems, this paper presents a new receiver-driven congestion control design, called REN, which uses the under- and over-utilization notifications from switch to handle the dynamic traffic. With the aid of explicit feedback, REN alleviates the traffic burstiness due to aggressive start, mitigates the conservativeness in utilizing available bandwidth, and still retains the receiver-driven feature to achieve ultra-low latency. We implement the prototype of REN using DPDK. The experimental results of real testbed and large-scale NS2 simulation show that REN effectively reduces the average flow completion time (AFCT) by up to 68% over the state-of-the-art receiver-driven transmission schemes.

Index Terms—Data center, receiver-driven, latency, link utilization.

1 Introduction

Large-scale data centers use over 100 thousand servers to host various applications. To provide high-bandwidth and low-latency communication, data center networks typically utilize the Clos network with shallow-buffered switches and 10/40 Gbps and even 100 Gbps links. To optimize the flow completion time in datacenter applications, numerous transport protocols such as DCTCP [1] are proposed to keep small queueing delay and mitigate Incast traffic. However, these sender-side proposals utilize reactive congestion control algorithms, which reduce sending rate after congestion already happens, inevitably inducing queue buildup and performance degradation of short or tiny flows in delay-sensitive applications.

To meet the stringent latency requirements (e.g., zero queueing delay) of large-scale applications in data center, many receiver-driven transport protocols such as Homa [2] have been proposed in recent years. In contrast to the traditional transport protocols which react to congestion after sending data packets, these state-of-the-art receiver-driven proposals proactively conduct congestion control even before sending data packets. In the receiver-driven congestion control, the data packets are paced by the grant or credit packets from the receiver. Thus, the sending rate of data packets does not exceed the capacity of bottleneck link, successfully avoiding queue buildup.

However, the receiver-driven congestion control mechanisms fundamentally face two problems. Firstly, since the senders start transmission at line rate in the first RTT to achieve high link utilizations, they are still prone to induce persistent queue buildup or even buffer overflow under bursty traffic. Secondly, the receiver-driven congestion control easily incurs low link utilization problem due to network dynamic. When some flows finish their transmissions, the remaining ones are not able to grab the available bandwidth of bottleneck link, because their data packets are only triggered by the arrivals of grant packets. In Section 2, we show that the receiver-driven approaches suffer from under-utilization issue under network dynamic, and experience large queueing latency under bursty traffic, resulting in suboptimal network performance.

In this paper, we propose a new congestion control mechanism, REN (Receiver-driven protocol based on Explicit Notification), which uses only two reserved ECN bits in packet header to deliver the under- and over-utilization notifications from switch. With the aid of switch-based explicit feedback, REN gracefully handles the bursty and dynamic traffic to achieve both zero-queueing and full link utilization. We implement REN using DPDK and show that REN introduces only a very small system overhead.

In summary, our major contributions are:

- We conduct an extensive experiment based study to investigate two key issues of current receiver-driven proposals: (1) the aggressive line-rate transmission in the first RTT easily leads to persistent queue backlog, (2) when some flows finish their transmissions in the dynamic traffic scenario, the released bandwidth can not be fully utilized by the other coexisting flows due to the receiver-driven manner.
- We propose a receiver-driven transport protocol REN, which uses packet marking to explicitly notify

- *Z. Li, J. Huang, J. Hu, W. Li, T. Zhang, J. Liu, and J. Wang are with the School of Computer Science and Engineering, Central South University, Changsha, Hunan, 410083.*
  E-mail: jiaweihuang@csu.edu.cn
- *T. He is with the Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN, USA 55455.*

the queue-buildup or spare bandwidth. Therefore, REN quickly eliminates queueing latency after the start of line-rate transmission, and meanwhile cautiously increases aggressiveness of receiver-driven transmission to achieve high link utilization. Moreover, REN can be easily deployed on the commercial switches with their built-in packet marking function.
- The test results of both testbed experiments and large-scale simulations demonstrate that REN significantly outperforms the state-of-the-art receiver-driven proposals in terms of link utilization and queueing delay. Compared with pHost [3], Homa and NDP [4], REN reduces the average flow completion time (AFCT) by up to 68% under typical datacenter workloads.

The rest of this paper is organized as follows. In Section 2 and 3, we describe our design motivation and details, respectively. In Section 4, we give the model analysis. In Section 5 and 6, we show the test results of testbed experiment and NS2 simulation, respectively. We present the related works in Section 7, and conclude this paper in Section 8.

2 Design Motivation

The receiver-driven transport protocols use proactive congestion control to achieve extremely low latency. When receiving one data packet, the receiver sends a corresponding grant packet, which allows the sender to transmit only one new data packet. However, modern data centers widely employ the Partition/Aggregate processing model to provide high performance. A large number of workers simultaneously send concurrent flows with various flow sizes, resulting in highly dynamic traffic [5], [6]. Fundamentally, the proactive congestion control has to face the problems of queue backlog and link under-utilization under network dynamic. In this section, we investigate these two issues in current receiver-driven transmission schemes.

2.1 Queue Backlog

The receiver-driven sender transmits only one data packet once receiving a grant packet from the receiver. Therefore, the sender naturally knows the safe sending rate without incurring congestion. However, the receiver-driven schemes cannot probe bandwidth before transmission. Without any prior knowledge about the available bandwidth, the sender has to start transmission in dark. To achieve full link utilization, when a new flow starts, the sending rate is link capacity (or line rate) in the 1st RTT. This aggressive and blind transmission easily leads to queue buildup and even buffer overflow. More seriously, after the 1st RTT, since the receiver-driven sending rate converges to the bottleneck link's output rate, the queue backlog still persists.

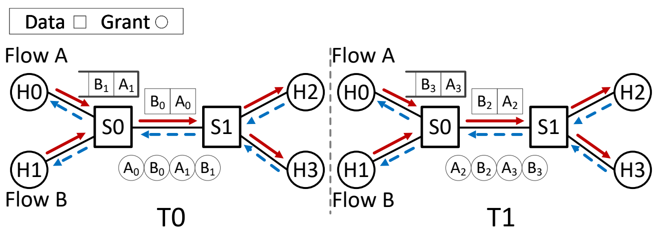


Fig. 1. Queue backlog issue.

We use an example to illustrate the queue backlog issue. Fig. 1 shows a typical many-to-many transmission scenario where two senders H0 and H1 respectively send flows A and B to 2 receivers H2 and H3. We assume that the bandwidth of each link is 2 packets per RTT. In the 1st RTT, H0 and H1 respectively send 2 packets at line rate to H2 and H3 simultaneously, resulting in a queue buildup at the switch S0. Then, the corresponding 4 grants are generated to drive 4 packets in next round. After that, since the number of in-flight packets is still 4, the queue backlog cannot be drained.

2.2 Link Under-utilization

In the concurrent transmission scenario of data center network, it is very common that multiple receiver-driven flows share one bottleneck link. Unfortunately, under the receiver-driven transmission, if some flows finish their transmissions and release bandwidth, the remaining ones are not able to increase their sending rates to make full use of bottleneck link, leading to under-utilization problem.

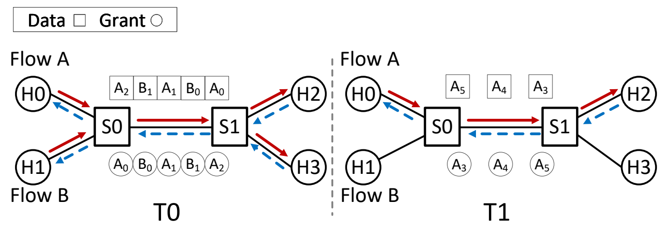


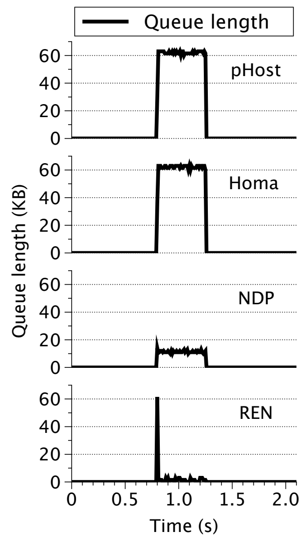
Fig. 2. Under-utilization issue.

We use another example to illustrate the link under-utilization issue. As shown in Fig. 2, at time T0, H0 and H1 send 3 and 2 packets, respectively. However, after H1 finish transmission, H0 still send only 3 packets in the next round T1, since the data packets are paced by the grant packets from the receiver. Therefore, the link cannot be fully utilized in the rest of transmission.

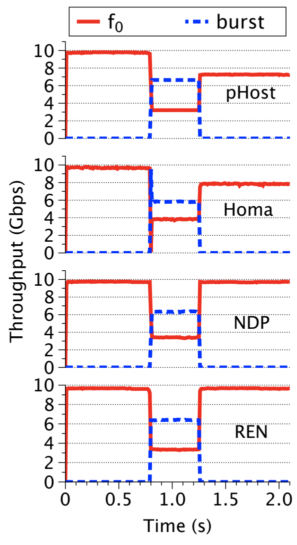
2.3 Empirical Study

We further conduct testbed experiments to investigate the performances of recent receiver-driven proactive proposals including pHost, Homa, NDP. We use a typical dumbbell topology including three senders H0, H1, and H2, and two receivers H3 and H4, which are connected through two switches S0 and S1. We generate f_0 and other two bursty flows with 1.5KB packet size. Flow f_0 is sent from host H0 to receiver R0, and 2 bursty flows are sent from hosts H1~H2 to another receiver H3. The link bandwidth and switch buffer size are 10Gbps and 64KB, respectively. The propagation delay of each link is 8μs. At the beginning, f_0 fully utilizes the bottleneck link. To simulate network dynamically, 2 bursty flows start at 0.8s and stop at 1.25s.

Fig. 3 (a) shows the real-time queue length of the bottleneck link. Under pHost and Homa, when two bursty flows start with line rate, the queue length immediately exceeds the buffer size, resulting in packet losses. Though NDP uses Cut Payload (CP) to strictly limit queue length to 8 packets, the queueing latency is still nonnegligible for



(a) Queue length



(b) Throughput

Fig. 3. Comparison of different protocols.

short flow with very small RTT. Under all three receiver-driven proposals, since the packet arrival rate at S0 is equal to the output rate after the 1st RTT, the queue length does not decrease until the bursty flows stop.

Moreover, Fig. 3 (b) shows the testbed results of the real-time throughputs under different protocols. At the beginning, the throughput of f_0 is 10Gbps. After two bursty flows stop at 1.25s, pHost and Homa cannot probe the available bandwidth and increase sending rates. The link is kept under-utilized to the end of test. Consequently, the link utilizations are reduced by 20%-30%. Compared with the other receiver-driven proposals, NDP quickly grabs the available bandwidth after the burst leave, since the payload-cut packets will trigger normal packets to utilize the available bandwidth.

2.4 Summary

Our analysis of the problem of receiver-driven transmission under network dynamic leads us to conclude that (1) current receiver-driven proposals potentially suffer from the queue buildup and cannot drain the queue under the burst traffic, resulting in persistent queueing latency. (2) The conservative nature of receiver-driven transmission makes it hard to fully utilize the bottleneck link even the spare bandwidth exists. To address these issues, we design and implement a receiver-driven transport protocol called REN to achieve low latency and high link utilization under network dynamic.

3 Protocol Design

To support the highly dynamic and concurrent flows in current datacenter applications, the transport design should be cost-effective in controlling queue buildup and probing available bandwidth. In this section, we present the design overview and details of REN.

3.1 Design Overview

To solve queueing latency and under-utilization problem, REN uses the under- and over-utilization notifications from switch to adjust the sending rate in its receiver-driven transmission. Fig. 4 describes the architecture of REN.

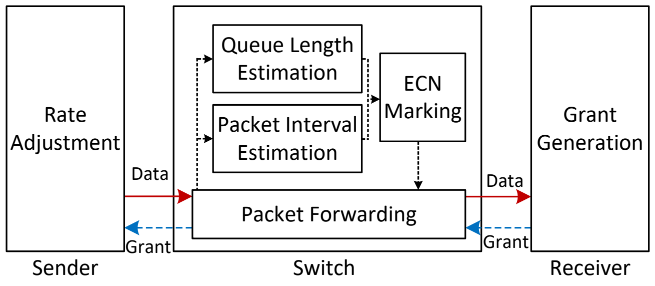


Fig. 4. Architecture.

(1) Switch. To provide fine-grained notifications, the switch measures the queue length and time interval of arrival packets in each sampling period. For each sampling period, if the queue length is larger than 0, the switch marks the reserved ECN bits in packet header to deliver the over-utilization signal. On the other hand, if the sum of time interval of consecutively arrival packets is larger than the transmission time of one data packet, the under-utilization signal is generated.

(2) Receiver. Once receiving a data packet, the receiver generates a corresponding grant packet, copies the ECN marks of data packet to the grant one, and sends it to the sender.

(3) Sender. With the explicit feedback of grant packets, the sender adjusts its sending rate. Specifically, if receiving one grant packet with over-utilization notification, to mitigate the queue buildup, the sender does not send out a new data packet. If one arrival grant packet carries under-utilization notification, two data packets will be triggered to improve the link utilization.

3.2 Design Details

The key points of REN design are to mitigate queue buildup and improve link utilization using ECN marking. We give the design details as follows.

3.2.1 *Mitigate queue buildup*

Even though the receiver-driven protocols conservatively generate only one data packet once receiving one grant packet, the newly arrival flows with line-rate start still lead to queue backlog. To address this issue, REN promptly reduces the sending rate once detecting the queue backlog.

For each sampling period, the switch updates the queue length ql (in packets). If ql is larger than 0, REN marks ql data packets with over-utilization signals. Since the corresponding grant packets with over-utilization notifications do not trigger new data packets, the queue backlog will be drained quickly.

As shown in Fig. 5, we use the same scenario as Fig. 1 to illustrate how to mitigate queue buildup. At time T0, H0 and H1 simultaneously start at line rate and respectively send 2 packets to H2 and H3, resulting in a queue buildup at the

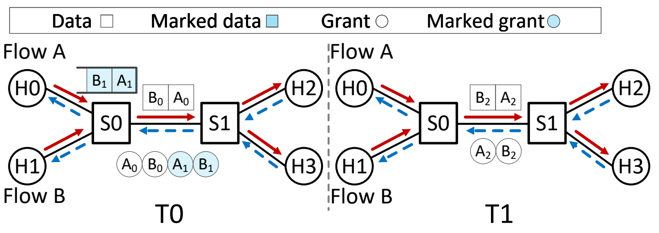


Fig. 5. Mitigate queue buildup.

switch S0. Then, S0 marks packets A1 and B1, which drive 2 marked grant packets. Since the marked grant packets do not trigger new data packets, H0 and H1 only generate 2 new data packets at time T1 to drain the buffered packets at the switch. Consequently, REN obtains the zero queueing delay, without degradation of link utilization. Intuitively, if receiving marked data packets, the receiver can stop sending corresponding grant packets to make the connection idle. However, to keep the end-to-end control loop, the receiver still generates the marked grant packets, which are only 40Bytes packet header without payload.

3.2.2 *Improve link utilization*

Due to the conservativeness in probing available bandwidth, the receiver-driven flows cannot grab the spare bandwidth even after some coexisting flows finish. In our design, the switch measures the total time interval, which is the sum of consecutive packets interval time in each sampling period. If the total time interval is larger than the transmission time of ti data packets, the switch marks ti packets with under-utilization signals. Note that we calculate the transmission time of each data packet as MSS/C, where MSS is the maximum segment size and C is the link capacity. Then, each corresponding grant packet from the receiver will trigger two new data packets at the sender to fill the gap between packets.

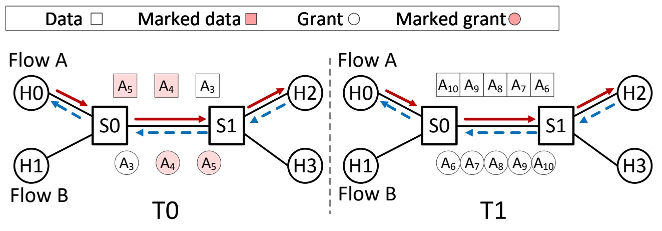


Fig. 6. Improve link utilization.

Fig. 6 gives the same scenario as Fig. 2 to show how REN improves the link utilization. At time T0, H0 sends 3 packets, in which the time interval between A3, A4 and A5 can accommodate 2 packets. When detecting the gaps between A3, A4 and A5, switch S0 marks A4 and A5 with under-utilization signals. At time T1, 2 data packets are generated on the arrival of each marked grant packet with under-utilization notification. Then, the sender transfers 5 packets to quickly fill up the bottleneck link in the next round of transmission.

3.2.3 *ECN marking*

To avoid extra traffic overhead in feedback, REN utilizes 2 reserved ECN bits in packet header to carry the state of link utilization. Specifically, the switch marks the ECN bits as $(00)_2$, $(01)_2$ and $(10)_2$ to indicate the under-, full- and over-utilization of its egress link, respectively.

Since the sending rate of each flow should be determined by the most congested bottleneck link in end-to-end transmissions, REN allocates different priorities to different feedback signals, that is, over- and under-utilization have the highest and lowest priorities, respectively.

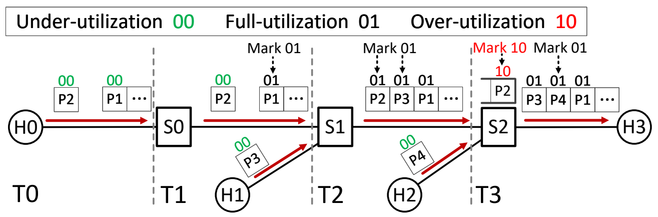


Fig. 7. ECN marking.

As shown in Fig. 7, H0 sends a flow to receiver H3. At time T0, all packets are marked as under-utilization (i.e., 00) by H0. When switch S0 detects the gap between the arrival packet and the previous one is less than the transmission time of one packet at time T1, the arrival packets are marked as full-utilization (i.e., P1 is marked as 01) since the full-utilization signal has higher priority than the under-utilization one. When P1 and P2 arrive at switch S1, P3 also arrives and fills up the gap between P1 and P2. At time T2, S1 marks P2 and P3 with full-utilization (i.e., 01). At time T3, packets P1∼P4 simultaneously arrive at switch S2, resulting in queue buildup. Then S2 marks P2 with over-utilization signal (i.e., 10) which has the higher priority than full-utilization one. Finally, H0 reduces its sending rate by one packet in the next round to achieve zero queueing latency and full utilization of bottleneck link between S2 and H3.

To provide real-time feedback, REN measures the queue length and packet interval in each sampling period (i.e., round trip propagation time) to make decision on marking packets. Due to high dynamic of network traffic, however, the switch may detect the queue buildup and idle link during the same sampling period. For example, during a sampling period, if the link is under-utilization at the beginning and then some new flows arrive, the bottleneck queue builds up and the bottleneck link becomes over-utilization. On the contrary, if the bottleneck link is over-utilization at the beginning and the some flows leave, the queue backlog drains quickly, resulting in under-utilization. To solve this issue, at the end of each sampling period, REN compares the values of queue length ql and total time interval of consequent packets ti, both of which are expressed in packets. If ql is larger than ti, $ql - ti$ packets are marked as over-utilization. Otherwise, $ti - ql$ packets are marked as under-utilization.

4 Model Analysis

Compared with the state-of-the-art receiver-driven proposals, REN features its ability in using packet marking to convey under- or over-utilization information. With the explicit notifications, the senders adjust the sending rates to avoid queue backlog and bandwidth wastage. Therefore, REN achieves near-zero queueing delay and full link utilization simultaneously. In this section, we construct the theoretical model to analyze the performance gain of REN in queue length and link utilization.

4.1 Queue Length

We assume that m_i flows start at the line rate and f_i flows finish in the i^{th} round trip time (RTT) round. All links have the same bandwidth C. The bandwidth-delay product BDP is $C \times RTT/MSS$ in packets.

In the 1^{st} RTT round, m_1 arrival flows make full use of the bottleneck link and generate a queue buildup with its queue length q_1 as

$$q_1 = (m_1 - 1) \times BDP. \tag{1}$$

In the i^{th} RTT round, m_i new flows starting at line rate increase the queue length by $m_i \times BDP$. Meanwhile, f_i flows finish transmissions. Assuming that all flows fairly share the bottleneck link, the released bandwidth rl_i by f_i flows is calculated as

$$rl_i = \frac{f_i}{\sum_{j=1}^{i-1} m_j - \sum_{k=1}^{i-1} f_k} \times BDP. \tag{2}$$

Under REN, the sender reduces the queue length by one BDP each RTT round through controlling the sending rate, Thus, the queue length in the i^{th} RTT round q_r_i is

$$q\_r_i = q\_r_{i-1} + m_i \times BDP - BDP - rl_i. \tag{3}$$

Under the traditional receiver-driven transports, however, the sender does not reduce its sending rate. Thus, we get the queue length in the i^{th} RTT round q_t_i as

$$q\_t_i = q\_t_{i-1} + m_i \times BDP - rl_i. \tag{4}$$

4.2 Link Utilization

We assume that m flows pass through the bottleneck link with capacity of C in the i^{th} RTT round, and the total rate of all flows is R_i. When R_i is less than C, the bottleneck link is not fully utilized. In our design, once the output rate is less than the bottleneck link capacity, the switch marks packets to indicate the under-utilization. Each corresponding grant packet triggers two data packets in the next round. Thus, we get the total rate R_{i+1} of all flows in the next RTT round as

$$R_{i+1} = \begin{cases} 2 \times R_i & R_i < \frac{C}{2}. \\ C & R_i \geq \frac{C}{2}. \end{cases} \tag{5}$$

From Equation (5), we calculate the number of RTTs N needed by the sender to increase its rate R to the bottleneck link bandwidth C as

$$N = \begin{cases} \lceil \log_2 \frac{C}{2R} \rceil + 1 & R < \frac{C}{2}. \\ 1 & R \geq \frac{C}{2}. \end{cases} \tag{6}$$

As shown in Fig. 8, a flow starts to increase its sending rate at time T_1. Under REN, the sender firstly uses $n = \lceil \log_2 \frac{C}{2R} \rceil$ RTT rounds to exponentially increase its sending rate and then needs one RTT round to converge

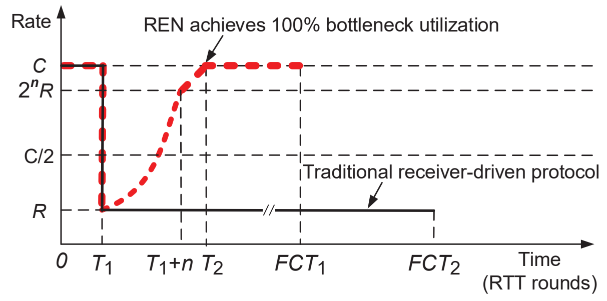


Fig. 8. REN vs. Traditional receiver-driven protocol.

its rate to the full utilization. REN fills up the full bandwidth at T_2. Compared with REN, the traditional protocols experience lower link utilization after T_1. When transmitting the same amount of data, REN finishes the transmission at FCT_1, earlier than the traditional protocols' completion time FCT_2.

Next, we construct a dynamic scenario. At the beginning, we assume that there are F active flows sharing the full bottleneck bandwidth fairly with no queue buildup. Then these flows leave during a period of time. The number of leaving flows in the i^{th} RTT round is f_i.

In the first RTT round, f_1 flows release their occupied bandwidth. The total throughput of remaining active flows thr_1 is

$$thr_1 = C - \frac{C}{F} \times f_1. \tag{7}$$

Under REN, the sender proactively seizes the available bandwidth once detecting link under-utilization. Given the bottleneck throughput thr_r_{i-1} in the $(i-1)^{th}$ round and the number of leaving flows f_i in the i^{th} round, we get the the bottleneck throughput thr_r_i in the i^{th} round as

$$thr\_r_i = \begin{cases} C - \frac{C}{F - \sum_{k=1}^{i-1} f_k} \times f_i & thr\_r_{i-1} \geq \frac{C}{2}. \\ 2 \times thr\_r_{i-1} - \frac{2 \times thr\_r_{i-1}}{F - \sum_{k=1}^{i-1} f_k} \times f_i & thr\_r_{i-1} < \frac{C}{2}. \end{cases} \tag{8}$$

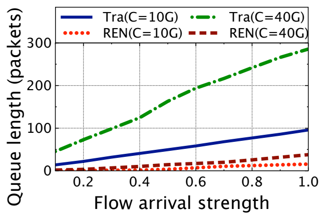
Under the traditional receiver-driven transports, however, the sender does not increase its sending rate though there is spare bandwidth. Thus, we get the bottleneck throughput in the i^{th} RTT round thr_t_i as

$$thr\_t_i = C - \frac{C}{F} \times \sum_{k=1}^{i} f_k. \tag{9}$$

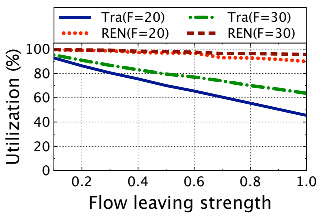
With the bottleneck throughputs, we get the link utilization ratios of REN and traditional receiver-driven transports as $\frac{thr\_r_i}{C}$ and $\frac{thr\_t_i}{C}$, respectively.

4.3 Numerical Analysis

We use numerical analysis to show the advantages of REN in draining queue and grabbing free bandwidth. We set the capacity of bottleneck link C to 10Gbps and 40Gbps, and the round trip time to 10μs. According to Poisson distribution, we generate the numbers of arrival flows and leaving flows. And the average flow arrival strength increases from 0.1 to 1, which is proportional to Poisson distribution's parameter λ. Fig. 9 (a) shows that REN achieves the ultra-low average queue length. The reason is that once the queue backlog occurs, REN marks packets according to the queue length to notify senders to reduce sending rate. Therefore, the queue length is effectively decreased especially when the flow arrival strength is small.



(a) Queue length



(b) Link utilization

Fig. 9. Numerical analysis.

The link utilization ratios of traditional receiver-driven protocols and REN are shown in Fig. 9 (b). In this numerical analysis, the capacity of bottleneck link C is set to 40Gbps. We generate the number of leaving flows f, according to the Poisson distribution. The flow leaving strength increases from 0.1 to 1, which is proportional to Poisson distribution's parameter λ. The initial number of flow F is changed from 20 to 30. In Fig. 9 (b), REN achieves significant benefits in link utilization. When the ratio of the flow leaving strength increases, REN can utilize more free bandwidth at the bottleneck link compared with the traditional receiver-driven transport protocols.

5 Testbed Evaluation

In this section, we firstly implement REN by using Intel DPDK and then evaluate the performance of REN on a real testbed. We compare it against pHost, Homa and NDP. All protocols are implemented via Intel DPDK and all parameters are set as the default values [2], [3], [4].

5.1 Real Implementation

We implement the prototype of both host-side and switch of REN using DPDK.

Host-side. We use $rte\_mbuf\_raw\_alloc()$ function to allocates an uninitialized struct rte_mbuf from the memory pool for constructing a packet. We use $rte\_pktmbuf\_mtod()$ function to point different offset of rte_mubf, corresponding to different fields of a packet header. Thus, we can read and modify any header fields easily. A constructed packet is send to the NIC by using $rte\_eth\_tx\_burst()$ function. And we use $rte\_eth\_rx\_burst()$ function to receive a packet from the NIC.

Specifically, at the receiver, once receiving a data packet by using $rte\_eth\_rx\_burst()$ function, REN constructs a rte_mbuf (grant) and copies the ECN of the data packet to this grant, and then sends it back to the sender by using $rte\_eth\_tx\_burst()$ function. At the sender, once receiving a grant by using $rte\_eth\_rx\_burst()$, REN constructs the different number of rte_mbuf (data packets) according to the ECN mark of grant header, and then sends them to the receiver by using $rte\_eth\_tx\_burst()$ function. Note that the ECN filed of each data packet is initialized as $(00)_2$ by using $rte\_pktmbuf\_mtod()$ function.

Switch. As shown in Fig. 10, the packet forwarding implementation of REN consists of the receiving, marking and transmitting modules. At the DPDK switch, each port has one receive queue and one transmit queue. One CPU core is used for each port to achieve fast packet processing.

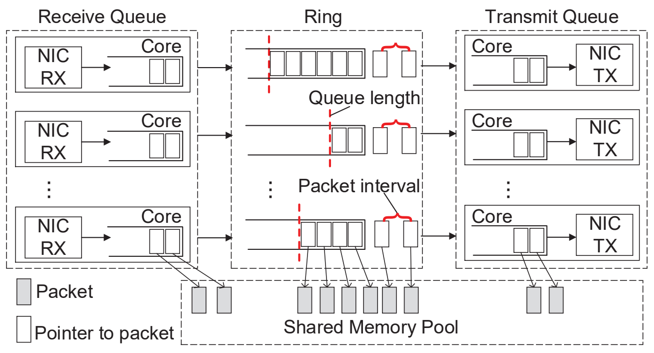


Fig. 10. REN's DPDK implementation.

Firstly, the receiving module uses the $rte\_eth\_rx\_burst()$ function to retrieve packets from receiving ports. When receiving a packet, the switch delivers it to a receive queue of the output port for further processing. The receive queue is allocated and set up by $rte\_eth\_rx\_queue\_setup()$ function. The output port is determined according to the route table. Note that one port has one logical queue in form of the *ring* structure that is created by using the $rte\_ring\_create()$ function. Each logical queue has a threshold that is set as the buffer size of each port. The arrival packets are dropped if the queue length of logical queue is larger than the threshold. The switch drops packets by calling the $rte\_pktmbuf\_free()$ function.

Then, the marking module is used to mark the packet to deliver the signal of over-utilization or under-utilization. Specifically, two custom counters are used to record the queue length of logical queue at each port and the packet interval between two consecutive packets, respectively. The queue length is obtained by the $rte\_ring\_count()$ function. The packet interval is calculated by the difference of dequeueing time of two consecutive packets, and the dequeueing time is obtained by the $rte\_rdtsc()$ and $rte\_get\_timer\_hz()$ functions. The switch firstly utilizes the two counters to assess the network utilization. Then the switch marks two reserved ECN bits in packet header according to the network utilization condition. Note that we use $rte\_pktmbuf\_headroom()$ function to access the two reserved ECN bits and utilize the $rte\_ipv4\_cksum()$ function to process the checksum of packet header.

Finally, the transmitting module moves packets from each logical queue to the transmit queue of corresponding output port, and then delivers the packets in the transmit queue to the network by using the $rte\_eth\_tx\_burst()$ function. We use $rte\_eth\_tx\_queue\_setup()$ function to allocate and set up the transmit queue.

Note that DPDK implements packets delivery by their pointer rather than real packet data. A pre-allocated memory pool is used to place real packet data described as rte_mbuf struct. Before sending out the packets to the network, DPDK driver gets packet data from shared memory pool. Besides, since each port is served by an individual CPU, the DPDK switch can process packets at the line rate.

5.2 Evaluation

The testbed consists of 6 servers, which are Dell PRECISION TOWER 5820 desktops. Each server has a Intel Core Xeon W-2255 CPU, 64GB memory, and 512 GB hard disk. The servers run Ubuntu16.06 with GNU/Linux kernel 3.18.20 and are equipped with Intel 10GbE 2P X520 Network Interface Cards (NICs). Two servers emulate two six-port DPDK switches. The other settings are as same as those in Section 2.

Firstly, we test whether REN effectively reduces the queue length and timely seizes the spare bandwidth. We use the same test settings in Section 2. According to the experience, we set the sampling period of REN as baseRTT 48μs. The bottom subfigures of Fig. 3 (a) and (b) show the queue length and flow throughput under REN, respectively.

At the beginning, f_0 starts at the line rate and then fully utilizes the bottleneck link. At 0.8s, two bursty flows start at line rate. As shown in Fig. 3 (a), although the instantaneous queue length increases after bursty flows arrive, REN drains queue quickly in the next RTT and then the queue length keeps near zero. The reason is that REN marks packets at the switches once detecting queue buildup and reduces the sending rate after receiving the corresponding marked grant packets. In Fig. 3 (b), f_0 and two bursty flows share the link bandwidth after 0.8s. When bursty flows stop at 1.25s, since REN marks packets to send under-utilization notification, f_0 is able to quickly grab the free bandwidth and achieve the full link utilization. Compared with test results of the other receiver-driven protocols shown in Fig. 3, REN simultaneously obtains ultra-low queue length and high link utilization.

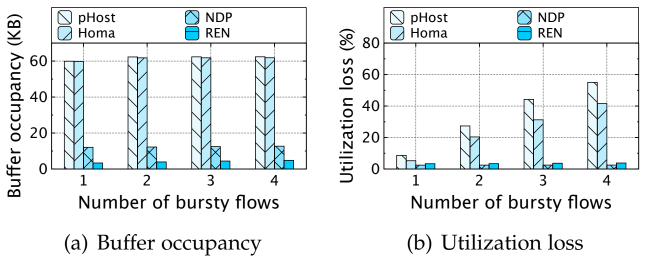


Fig. 11. Perfomance with varying burst degree.

Then, we test REN performance under different bursty strengths. We increase the number of sending hosts of bursty flows in the topology described in section 2. Fig. 11 (a) shows the buffer occupancy under bursty traffic from 0.8s to 1.25s. When the number of bursty flows increases from 1 to 4, REN keeps a very low buffer occupancy with different bursty strengths. Fig. 11 (b) shows the loss of link utlization after the bursty flows finish transmissions. Due to the conservativeness in receiver-driven transmission, pHost and Homa experience greater loss of link utilization with more busrty flows.

Next, we test the flow completion time (FCT) to evaluate the performance of REN with different flow sizes. Specifically, two long-lived background flows between two pairs of hosts are sent in advance to fully occupy the bottleneck link. During the transfer of background flow, three hosts send three short flows to one receiving host. We choose the total flow size as 30KB, 40KB, 50KB, and 60KB. All the short flows need to compete with the background flows at the bottleneck link. We repeat the test for 100 times with each flow size.

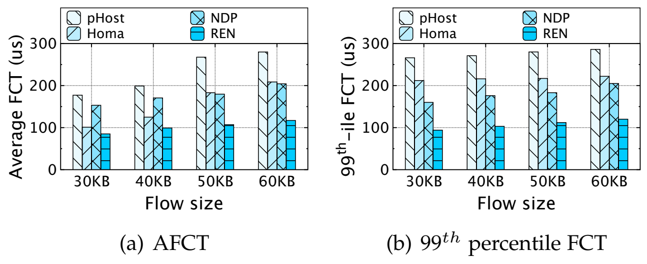


Fig. 12. FCT with varying flow size.

In Fig. 12 (a), REN has the lowest AFCT across all different flow sizes. Since REN drains the queue before the arrivals of short flows, it successfully avoids the packet loss. However, pHost and Homa suffer from persistent queue backlog, which results in a higher probability of tail packet loss and timeout. Although NDP uses packet payload cutting to avoid packet loss and keep queue length around a small threshold, the packet retransmission still enlarges the FCT of short flows. For example, with the flow size of 60KB, REN brings about $\sim$58%, $\sim$42% and $\sim$43% AFCT reductions over pHost, Homa and NDP, respectively. The same trend is observed for the 99th percentile tail FCT. As shown in Fig. 12 (b), REN gets the smallest 99th FCT and achieves great improvement for short flows.

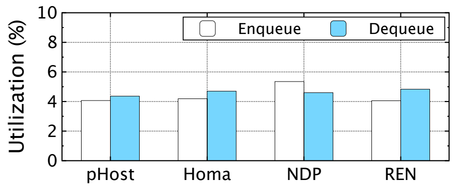
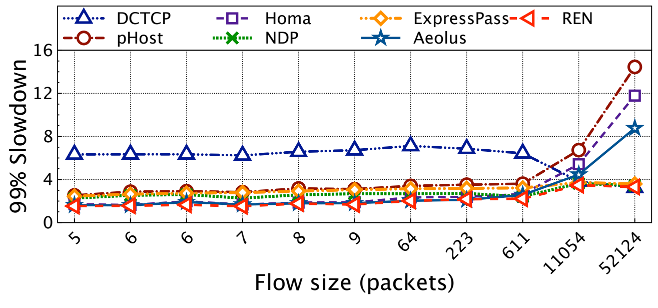
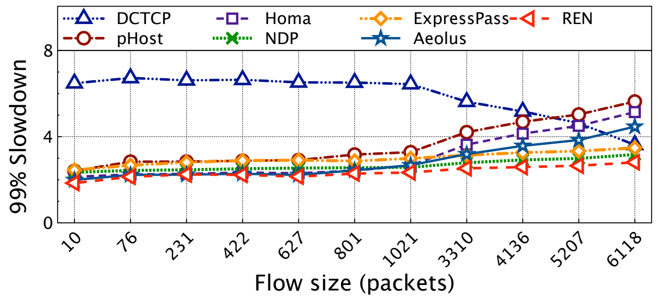


Fig. 13. CPU overhead.

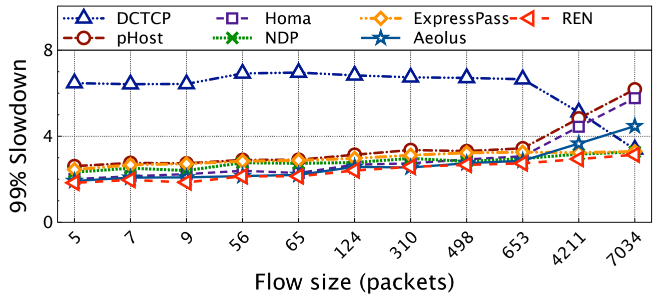
Finally, we evaluate the system overhead of REN. The marking packets function of REN is implemented on DPDK switch, resulting in computing overhead in the enqueue and dequeue modules. In this test, we measure the CPU utilization, which is the ratio of the CPU cycles of each module to that of whole forwarding process. REN simply marks the packets using the build-in ECN function. Thus, as shown in Fig. 13, REN still achieves very low CPU utilizations in both enqueue and dequeue operations. Moreover, REN maintains only 16Bytes for ql, ti, and a timestamp state at each port. Therefore, a 32-ports switch only needs to maintain 512Bytes of state information. The switch measures the under-utilization and over-utilization states at each port and does not record per-flow information. Thus, the required SRAM resources for REN are independent of the number of flows.



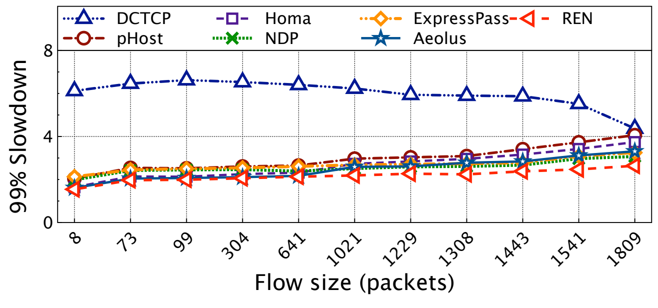
(a) Data Mining



(b) Web Search



(c) Hadoop Cluster



(d) Cache Follower

Fig. 14. The 99^{th} percentile slowdown.

6 Simulation Evaluation

We conduct NS2 simulations to evaluate REN performance in a large-scale network topology with typical datacenter workloads. We measure the average FCT and 99^{th} percentile FCT of short flows, the throughput of long flows, and the average flow completion time of all flows. We compare performances of REN, DCTCP, and the state-of-the-art receiver-driven transport protocols including pHost, Homa, NDP, ExpressPass [7] and Aeolus [8]. We set the ExpressPass parameters α and w_{init} to 1/16, as recommended in Ref.[7]. And Aeolus is integrated with the latest receiver-driven protocol Homa.

Network topology:

We use a common leaf-spine topology with 8 top-of-rack (ToR) switches and 8 core switches [2], [8]. Each ToR switch connects to 48 hosts and the whole network has 384 end-hosts. The bandwidth of each link is 40Gbps and the round trip propagation delay is 10μs. Thus, the oversubscription ratio is 6:1. The packet size and switch buffer size are 1.5KB and 128KB, respectively.

Traffic workloads: We use four traffic workloads of data mining, web search, Hadoop cluster and cache follower, which are typical applications in productive data center networks [2], [3], [7]. Specifically, the four realistic workloads cover a wide range of average flow sizes ranging from 64KB to 7.4 MB. The distributions of flow size are heavy-tailed with about more than 90% bytes contributed by less than 10% large flows. For example, in data mining application, more than 80% of flows are smaller than 100KB and about 10% flows are larger than 1MB. We use all-to-all communication pattern, and generate flows between randomly selected host pairs in the all-to-all transmissions and the arrival rates of flows obeys a Poisson process. We simulate seven traffic loads ranging from 0.1 to 0.7.

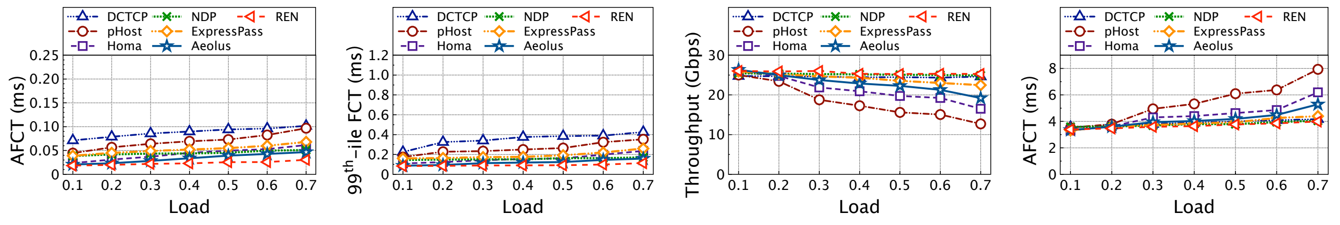
6.1 Performance Under Realistic Workloads

To provide a comprehensive test, we measure the 99^{th} percentile tail slowdowns of all flows in four realistic workloads at 0.5 network load. The simulation results with increasing flow size are shown in Fig. 14. The slowdown is defined as the ratio of the actual flow completion time to the ideal time required to complete the flow transmission without experiencing any congestion. The x-axis for each figure in Fig. 14 is scaled to match the distribution of flow size. Specifically, the axis is linear in the total number of flows and each tick is corresponding to 10% of all flows in workload.

Fig. 14 shows the 99^{th} percentile tail slowdown as a function of flow size at 0.5 network load. In general, the tail latency of REN is lower than that under the other four receiver-driven protocols and DCTCP. Since DCTCP, pHost, NDP and Homa are not able to actively reduce the queue length and timely fill up the spare bandwidth, the FCTs of short and long flows are larger than those under REN across the four workloads, resulting in worse slowdown performance than REN. Moreover, the performance improvement of REN is normally higher for larger flows in each workload, and the similar trend is observed in the workloads with larger average flow size since REN has more opportunities to drain queue and seize spare bandwidth. Specifically, in data mining and cache follower workloads, REN improves slowdown by ∼9%, ∼51%, ∼39%, ∼4%, ∼6%, ∼21% and ∼50%, ∼35%, ∼29%, ∼14%, ∼18%, ∼20% with the flow size of 11054 packets and 1809 packets over DCTCP, pHost, Homa, NDP, ExpressPass and Aeolus, respectively.

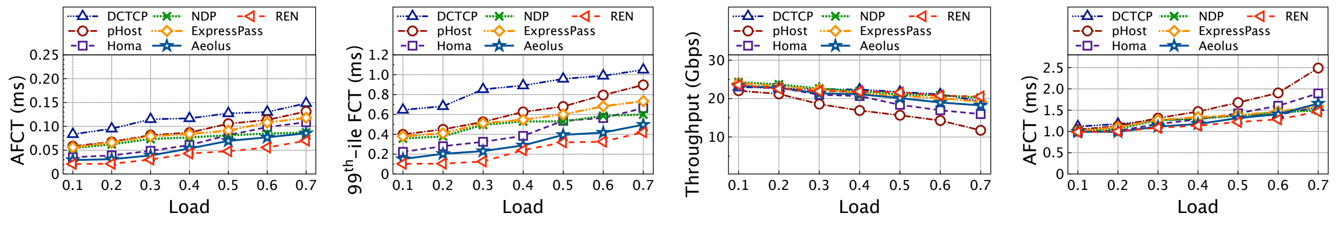
Next, we measure the FCT of short flows and throughput of long flows under varying load. The test results of four workloads are shown in Fig. 15∼Fig. 18.

As shown in Fig. 15 (a), Fig. 16 (a), Fig. 17 (a) and Fig. 18 (a), REN achieves the lowest average FCT of short



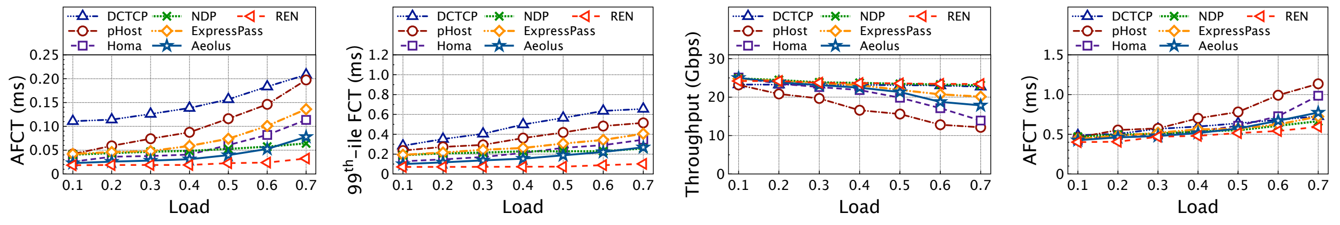
(a) AFCT of short flows (b) 99th percentile FCT of short flows (c) Throughput of long flows (d) AFCT of all flows

Fig. 15. Data Mining application.



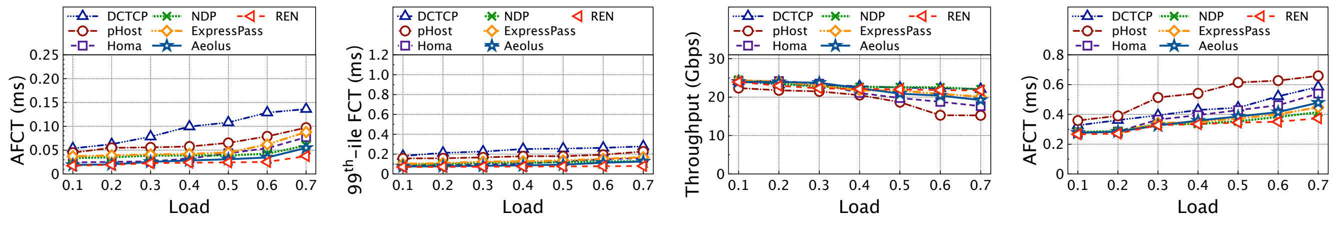
(a) AFCT of short flows (b) 99th percentile FCT of short flows (c) Throughput of long flows (d) AFCT of all flows

Fig. 16. Web Search application.



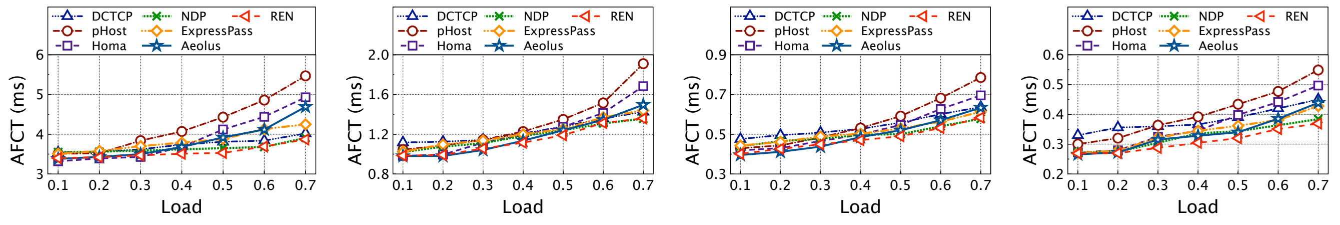
(a) AFCT of short flows (b) 99th percentile FCT of short flows (c) Throughput of long flows (d) AFCT of all flows

Fig. 17. Hadoop Cluster application.



(a) AFCT of short flows (b) 99th percentile FCT of short flows (c) Throughput of long flows (d) AFCT of all flows

Fig. 18. Cache Follower application.



(a) Data Mining (b) Web Search (c) Hadoop Cluster (d) Cache Follower

Fig. 19. AFCT of all flows in 4 realistic workloads.

flows with varying load from 0.1 to 0.7. Since REN drains queue by controlling the sending rate according to the marked grant packets, the short flows experience much lower queueing delay and thus obtain lower FCT compared to the other receiver-driven protocols and DCTCP. Since REN starts the transmission at line rate while ExpressPass wasting the first RTT to request the credits, REN reduces the average FCT of short flows by up to ∼75% than ExpressPass.

Fig. 15 (b), Fig. 16 (b), Fig. 17 (b) and Fig. 18 (b) show that, REN significantly reduces the tail FCT for short flows. Specifically, in data mining workload, REN reduces the 99^{th} percentile FCT for short flows by ∼73%, ∼68%, ∼52%, ∼33%, ∼56% and ∼28% at 0.7 load over DCTCP, pHost, Homa, NDP, ExpressPass and Aeolus, respectively.

In Fig. 15 (c), Fig. 16 (c), Fig. 17 (c) and Fig. 18 (c), REN significantly outperforms pHost and Homa in long flow throughput across all workloads, because REN is able to timely seize the free bandwidth by marking packets to notify the senders to fill the packet gap accordingly. In addition, REN obtains larger improvements in data mining and web search scenarios than Hadoop cluster and cache follower applications. The data mining and web search applications have more large flows, which cannot grab the spare bandwidth after experiencing congestion under the traditional receiver-driven protocols. Therefore, REN achieves more improvements in link utilization under data mining and web search applications. Moreover, under heavier load, REN performs much better because it has more opportunities to grab the released bandwidth. Since ExpressPass uses per-flow ECMP to ensure in-order transmssion, the link utilization is degraded compared with NDP and REN, which use per-packet loader balancing.

The average FCT of all flows are shown in Fig. 15 (d), Fig. 16 (d), Fig. 17 (d) and Fig. 18 (d). As the load increases, REN performs better than the other protocols across all workloads, because it achieves the ultra-low queueing delay and high link utilization simultaneously by adjusting the sending rate based on explicit notifications. Specifically, in web search workload, REN reduces the AFCT of all flows by ∼6%, ∼41%, ∼22%, ∼2%, ∼8% and ∼11% at 0.7 load over DCTCP, pHost, Homa, NDP, ExpressPass and Aeolus, respectively. In cache follower workload, REN reduces the AFCT of all flows by ∼36%, ∼43%, ∼30%, ∼9%, ∼17% and ∼21% at 0.7 load over DCTCP, pHost, Homa, NDP, ExpressPass and Aeolus, respectively.

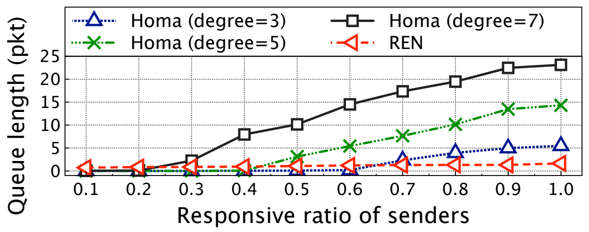
To provide a comprehensive experiment, we compare REN with the state-of-the-art receiver-driven protocols when the oversubscription of the topology is 1. We use a leaf-spine topology with 8 spine switches, 8 leaf switches and 64 servers. Each leaf switch connects to 8 hosts. All the links have 40Gbps bandwidth and the base RTT is set to 10μs. We use the same 4 realistic workloads as above to evaluate REN. Fig. 19 shows the average FCT in 4 realistic workloads under varying network loads. Though the oversubscription of the topology is 1, REN still reduces the average FCT by up to ∼18%, ∼33%, ∼26%, ∼4%, ∼13% and ∼17% over DCTCP, pHost, Homa, NDP, ExpressPass and Aeolus, respectively. The reason is that when the sources from multiple leaf switches send flows to different destinations from a same leaf switch, congestion and packet loss occur on the spine switch. Therefore, after congestion occurs and some flows are completed, there is spare bandwidth. By using the under-utilization signal, REN seizes the free bandwidth quickly. Although Aeolus solves the first-RTT problem well, it cannot fill up the free bandwidth, resulting in longer average FCT.

6.2 Many-to-many Communication Scenario

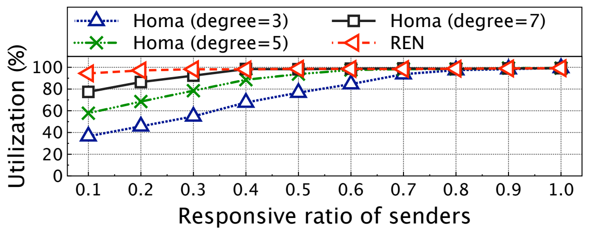
In many-to-many communication scenarios, the receiver-driven protocols introduce a new challenge: a sender can not immediately respond to all grants from different receivers. In such cases, the links connected to the receiver are potentially wasted, resulting in performance degradation especially at the heavy network load.

To address this issue, Homa implements the overcommitment mechanism to improve link utilization. That is, a receiver can send grants to multiple senders simultaneously (i.e., degree of overcommitment is larger than 1). Even though a sender cannot respond, the link can be fully utilized by the other senders. In this test, we compare the REN performance with Homa's overcommitment mechanism in a many-to-many communication scenario.

In this test, we use a leaf-spine topology with two ToRs, each of which connects 40 senders. Each sender establishes 2 connections with 2 receivers under different ToRs. The other simulation settings are as same as those in Section 6. We measure the bottleneck link utilization and average queue length with the varying responsive ratio of the senders. We change the degree of overcommitment of Homa. The test results are shown in Fig. 20.



(a) Average queue length



(b) Bottleneck utilization

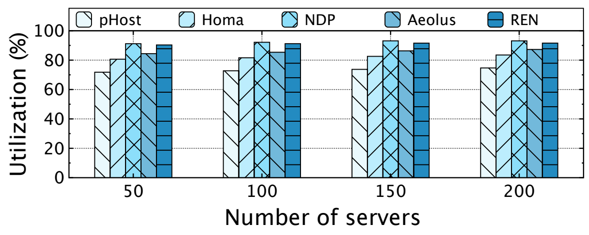
Fig. 20. The link utilization and queue length with varying responsive senders.

As shown in Fig. 20 (a) and Fig. 20 (b), with the increasing degree of overcommitment and ratio of responsive senders, Homa achieves higher link utilization. However, the queue length also increases at the same time. It is hard for Homa to achieve good tradeoff between small queue length and high link utilization with fixed degree of overcommitment. For REN, once the bottleneck link is not fully utilized, the marked packets will notify the sender to increase sending rate to sustain high link utilization. Meanwhile, once detecting queue buildup, the switch notifies the sender to decrease sending rate to reduce buffer occupancy. Therefore, REN is able to obtain both near-zero queue length and high link utilization.

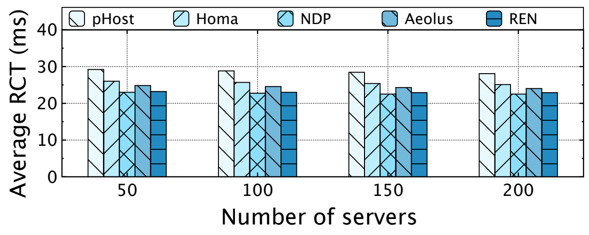
6.3 Incast Scenario

The Incast scenario potentially causes performance deterioration due to bursty traffic in data centers. In this section, we test the effectiveness of REN under the Incast scenario.

We use the same leaf-spine topology and simulation settings as in Section 6.2, except for the number of end-hosts and ToRs. We generate the Incast traffic similar to that in [3]. Specifically, M hosts under each of the first two leaf switches act as servers, and two hosts under the third leaf switch act as a client. Each client requests 100MB/M bytes from M servers under the first two leaf switches, and each server synchronously responds a fixed amount of required data to the client. The client could not issue a new round of requesting until it receives all responses in the current round. Therefore, the completion time of the last response in current round is the request completion time (RCT). We repeat this pattern 100 times. In this test, we vary the number of servers M from 50 to 200. Each server generates 100MB/M data in a single flow.



(a) Bottleneck utilization



(b) Average request completion time

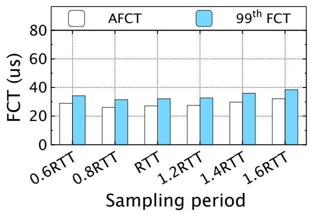
Fig. 21. Incast performance.

As shown in Fig. 21 (a), REN achieves higher link utilization than the other receiver-driven protocols because it actively utilizes the spare bandwidth. Moreover, each of four transport protocols provides high link utilization without experiencing Incast throughput collapse. Fig. 21 (b) shows the average RCTs of four schemes. Since REN retains the proactive congestion control in the receiver-driven transmission and uses marking-based feedback to improve link utilization, it obtains lower RCT than the others. In general, all of these protocols successfully avoid performance degradation under the Incast scenario because they effectively reduce the contention across flows to the same destination by employing the conservative receiver-driven congestion control.

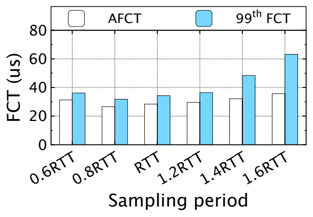
6.4 Parameter Sensitivity

To provide timely feedback, REN measures the queue length and packet interval with a sampling period. To evaluate the parameter sensitivity of sampling period, we set up a concurrent transmission scenario and test the impact of sampling period under 0.2 and 0.8 network loads.

In this test, we use a dumbbell topology with multiple pairs of end-hosts. The simulation settings are as same as those in Section 6. At the beginning, two long-lived background flows fairly share the bottleneck link. Then the concurrent short flows start at the line rate. Since the network dynamic has large impact on short flows, we measure the AFCT and 99^{th} percentile FCT of short flows with varying sampling period.



(a) Under 0.2 load



(b) Under 0.8 load

Fig. 22. Sampling period.

As shown in Fig. 22 (a), with different sampling periods, REN generally achieves the satisfied FCT of short flows under light load. Under heavy load, however, the tail FCT of short flows increases greatly with larger sampling period as shown in Fig. 22 (b). This result means that, under heavy traffic load, it is hard to well control the queue length with too large sampling period. On the other hand, the rate adjustment with too small sampling period easily causes frequent queue oscillation and utilization losses, resulting in small increase of FCT. Therefore, we choose one round trip propagation time as the sampling period in our design.

7 Related Works

Recent years have witnessed a series of novel transport designs to achieve low latency and high throughput in data centers. In general, these mechanisms can be roughly divided into four categories: sender-based, rate-based, explicit-based and receiver-driven ones.

As a classic representative of sender-based design, DCTCP [1] marks packets according to queue length. The arrival packet is marked via ECN when the queue length is larger than the marking threshold. In this fashion, DCTCP obtains small queueing delay and high throughput. To meet the demand of deadline-aware flows, D^3 [9] allocates bandwidth to each flow according to application deadline information. HULL [10] employs phantom queues to deliver the congestion signal and reserves bandwidth headroom for short flows to ensure low flow completion time. As a fine-grained end-to-end congestion control, DCQCN [11] adjusts the sending rate with the help of ECN marking to alleviate the impacts of Priority-based Flow Control (PFC), such as the head-of-line blocking and unfairness. TIMELY [12] uses NIC timestamps and fast ACK turnaround methods to accurately measure the round-trip time as the congestion feedback, achieving high throughput and low latency without the the requirements of switch supports. Swift [13] improves TIMELY by dividing end-to-end RTT into NIC-to-NIC (fabric) delay and endhost delay to respectively respond to congestion of fabric and endhost. By this way, Swift provides a fine-grained delay signal for congestion control, and adjusts the sending rate precisely. To quickly respond to the sudden congestion and achieve both high throughput and low latency, based on the existing congestion control protocols (e.g. TIMELY and DCQCN), On-Ramp [14] adds the one-way delay measurement to sense the congestion. When the sudden congestion occurs, On-Ramp quickly reduces the network latency by pausing the packets sending. These protocols achieve relatively low queueing delay and high throughput compared with traditional TCPs. Under large traffic burstiness, however, these designs still easily experience large queueing delay and even buffer overflow.

To accelerate the convergence speed and achieve the target rate rapidly, some rate-based approaches have been proposed. PDQ [15] employs preemptive flow scheduling through sophisticated rate calculation to reduce the flow completion time. pFabric [16] implements priority queues and isolates short and long flows to reduce the flow completion time. Fastpass [17] applies a centralized arbiter to determine when each packet should be sent as well as the path it should take. Karuna [18] uses priorities to segregate the flows with or without deadline. Nevertheless, this kind of mechanisms requires complicated control and global scheduling, potentially resulting in performance degradation of short and tiny flows due to the time overhead in centralized control and scheduling.

In wide area network (WAN), a variety of classic high speed TCPs employ explicit feedback information to adjust the sending rate to precisely match the available bandwidth of bottleneck link. For example, XCP [19] leverages multiple bits in packet header to deliver the load state of bottleneck link to sender. According to the explicit feedback, the sender regulates its congestion window size to achieve low queueing delay and high link utilization. RCP [20] emulates the processor sharing at router, which calculates a proper rate for each flow and returns it through packet header to sender. However, since these protocols are designed for WAN, they do not allow the flow to start at line rate, resulting in slow convergence for short or tiny flows in data center. Therefore, several explicit feedback protocols are proposed for data center networks. For example, under the window-based protocol TFC [21], the switch allocates the proper window size for each flow through the measurement of both network capacity and number of active flows. However, before starting transmission, each flow still needs one more round-trip time to get the initial window size, which is detrimental for short flows. With the popularity of in-network telemetry (INT) technique, HPCC [22] controls the sending rate based on the fine-grained link load information. However, this design generally needs extra feedback bits in packet head, making it hard to strike a balance between the small traffic overhead and precise rate adjustment.

For achieving near-zero queueing delay and ultra-low packet loss, many receiver-driven schemes have been proposed. pHost [3] separates scheduling strategies from the network fabric and leverages distributed per-packet scheduling at the end hosts. Moreover, it requires no switch-support and achieves high performance. ExpressPass [7] effectively controls congestion even before sending data packets via shaping the flow of credit packets at the switch. NDP [4] assigns new flows with full rate at the beginning. When the queue length exceeds the given threshold, the switch trims the packet payloads and gives the header of trimmed packets the highest priority. Once receiving a header, the receiver immediately sends an NACK to inform the sender to retransmit the packet. For each received header or packet, the receiver generates a pull packet and utilizes pull packets to control the sending rate. Homa [2] utilizes in-network priority queues and the shortest remaining processing time first (SRPT) policy to guarantee low latency for short messages. Moreover, it also employs overcommitment mechanism to ensure efficient bandwidth utilization in many-to-many communication scenarios. Aeolus [8] makes an adequate utilization of spare bandwidth in the first RTT, rather than sending packets blindly. Through distinguishing traffic at the end side and implementing selective dropping in the network, Aeolus permits aggressive fast start without affecting the traffic scheduled by proactive congestion control. Polo [23] marks the packets with ECN when the queue length exceeds a small threshold, preventing the large queue backlog. And it uses high priority packets and keeps them flight to detect packet loss, recovering the lost packets as fast as possible. AMRT [24] exploits ECN to mark packets on underload link, and rises the sending rate according to each marked packet, alleviating the under-utilization problem. Unfortunately, pHost and Homa may suffer from low link utilization and queue buildup under high dynamic traffic scenario, leading to a suboptimal network performance. ExpressPass wastes the first RTT of a new flow to wait for credits. NDP and Aeolus keep the queue length around the threshold through cutting payload and dropping packets, respectively. However, these methods increase the probability of packet loss (or payload cut). Though they could retransmit the packet quickly, it is still detrimental for short flows' FCT. Though Polo keeps the queue length around a small threshold, it still increases the delay in network. Since AMRT has no slow down scheme, it still suffers from queue backlog and network congestions.

Recently, a new congestion control protocol ABC [25] is proposed for time-varying wireless links. It marks each packet with the accelerate or brake signals on the router output port according to port load. The sender respectively increases or decreases the size of congestion window when receiving the accelerate or brake signals to achieve high throughput and low latency. However, as a window-based protocol, ABC begins transmission with the slow start phase and needs several RTTs to reach full bandwidth, making it unsuitable for short or tiny flows in data centers. Moreover, since ABC is not receiver-driven, it cannot schedule flows with different priorities at the receiver in many-to-one communication scenarios.

Compared with the above transport approaches, REN delivers the under- and over-utilization information of bottleneck link via only two reserved ECN bits in the packet header. With the explicit notification from switch, REN tackles the bursty traffic well, achieving high link utilization and zero queueing latency.

8 Conclusion

We presented REN, a new receiver-driven transport protocol for data center networks. REN utilizes the explicit under- and over-utilization notifications from switch to adjust the sending rate. It provides much better performance than current receiver-driven proposals which aggressively start transmission at line rate and cannot seize the spare bandwidth due to the conservative receiver-driven transmission. Moreover, the real implementation of REN is not expensive in terms of CPU utilization. In our DPDK implementation and large-scale simulations, REN exhibits excellent low-latency and high-utilization behaviors. It reduces the average FCT by up to 68% over the state-of-the-art receiver-driven proposals.

References

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

[2] B. Montazeri, Y. Li, M. Alizadeh, and J. Ousterhout. Homa: A receiver-driven low-latency transport protocol using network priorities. In Proc. ACM SIGCOMM, 2018.

[3] P. Gao, A. Narayan, G. Kumar, R. Agarwal, S. Ratnasamy, and S. Shenker. pHost: Distributed near-optimal datacenter transport over commodity network fabric. In Proc. ACM CoNEXT 2015.

[4] M. Handley, C. Raiciu, A. Agache, A.Voinescu, A. Moore, G. Antichi, and M. Wójcik. Re-architecting datacenter networks and stacks for low latency and high performance. In Proc. ACM SIGCOMM, 2017.

[5] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat. Hedera: Dynamic flow scheduling for data center networks. In Proc. USENIX NSDI, 2010.

[6] T. Wang, F. Liu, J. Guo, and H. Xu. Dynamic SDN controller assignment in data center networks: Stable matching with transfers. In Proc. IEEE INFOCOM, 2016.

[7] I. Cho, K. Jang, and D. Han. Credit-scheduled delay-bounded congestion control for datacenters. In Proc. ACM SIGCOMM, 2017.

[8] S. Hu, W. Bai, G. Zeng, Z. Wang, and Y. Wang. Aeolus: A building block for proactive transport in datacenters. In Proc. ACM SIGCOMM, 2020.

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

[10] M. Alizadeh, A. Kabbani, T. Edsall, B. Prabhakar, A. Vahdat, and M. Yasuda. Less is more: Trading a little bandwidth for ultra-low latency in the data center. In Proc. USENIX NSDI, 2012.

[11] Y. Zhu, H. Eran, D. Firestone, C. Guo, M. Lipshteyn, Y. Liron, J. Padhye, S. Raindel, M. H. Yahia, and M. Zhang. Congestion control for large-scale RDMA deployments. In Proc. ACM SIGCOMM, 2015.

[12] R. Mittal, V. T. Lam, N. Dukkipati, E. Blem, H. Wassel, M. Ghobadi, A. Vahdat, Y. Wang, D. Wetherall, and D. Zats. Timely: Rtt-based congestion control for the datacenter. In Proc. ACM SIGCOMM, 2015.

[13] G. Kumar, N. Dukkipati, K. Jang, H. M. Wassel, X. Wu, B. Montazeri, Y. Wang, K. Springborn, C. Alfeld, M. Ryan and D. Wetherall. Swift: Delay is simple and effective for congestion control in the datacenter. In Proc. ACM SIGCOMM, 2020.

[14] S. Liu, A. Ghalayini, M. Alizadeh, B. Prabhakar, M. Rosenblum and A. Sivaraman. Breaking the transience-equilibrium nexus: A new approach to datacenter packet transport. In Proc. USENIX NSDI, 2021.

[15] C. Y. Hong, M. Caesar, and P. B. Godfrey. Finishing flows quickly with preemptive scheduling. In Proc. ACM SIGCOMM, 2012.

[16] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker. pFabric: Minimal near-optimal datacenter transport. In Proc. ACM SIGCOMM, 2013.

[17] J. Perry, A. Ousterhout, H. Balakrishnan, D. Shah, and H. Fugal. Fastpass: A centralized ”zero-queue” datacenter network. In Proc. ACM SIGCOMM, 2014.

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

[19] Katabi D, Handley M, Rohrs C. Congestion control for high bandwidth-delay product networks. In Proc. ACM SIGCOMM, 2002.

[20] Dukkipati N, Kobayashi M, Zhang-Shen R, et al. Processor sharing flows in the internet. In Proc. IWQoS, 2005.

[21] J. Zhang, F. Ren, R. Shu, and P. Cheng. TFC: Token flow control in data center networks. In Proc. ACM EuroSys, 2016.

[22] Y. Li, R. Miao, H. H. Liu, Y. Zhuang, F. Feng, L. Tang, Z. Cao, M. Zhang, F. Kelly, M. Alizadeh, and M. Yu. HPCC: High precision congestion control. In Proc. ACM SIGCOMM, 2019.

[23] C. Ruan, J. Wang, W. Jiang, and T. Zhang. Polo: Receiver-driven congestion control for low latency over commodity network fabric. In Proc. ACM ICPP, 2020.

[24] J. Hu, J. Huang, Z. Li, J. Wang, and T. He. AMRT: Anti-ECN marking to improve utilization of receiver-driven transmission in data center. In Proc. ACM ICPP, 2020.

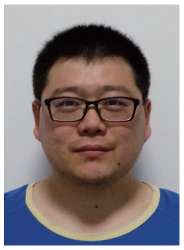
[25] P. Goyal, A. Agarwal, R. Netravali, M. Alizadeh, and H. Balakrishnan. ABC: A simple explicit congestion controller for wireless networks. In Proc. USENIX NSDI, 2020.

Zhaoyi Li received the B.S. degree from Central South University, China, in 2019. He is currently pursuing the Ph.D. degree in the School of Computer Science and Engineering at Central South University, China. His research interests are in the area of data center networks.

Jiawei Huang obtained his PhD (2008) and Masters degrees (2004) from the School of Computer Science and Engineering at Central South University. He also received his Bachelor's (1999) degree from the School of Computer Science at Hunan University. He is now a professor in the School of Computer Science and Engineering at Central South University, China. His research interests include performance modeling, analysis, and optimization for wireless networks and data center networks.

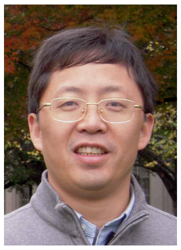
Jinbin Hu received the B.E. and M.E. degrees from Beijing Jiao Tong University, China, in 2008 and 2011, respectively. She also received her Ph.D. degree from the School of Computer Science and Engineering, Central South University, China, in 2020. Her current research interests are in the area of data center networks.

Weihe Li is currently pursuing the M.S. degree in the School of Computer Science and Engineering at Central South University, China. His research interests are video streaming and data center networks.



Tao Zhang received his Ph.D. degree in the School of Computer Science and Engineering, Central South University, China. He is now an associate professor in Hunan Province Key Laboratory of Industrial Internet Technology and Security, Changsha University, China. His research interests include congestion control, load balancing, performance modeling, analysis, and data center networking.

Jingling Liu received the B.E. degrees from Central South University, China, in 2016. She is currently pursuing her Ph.D. degree in the School of Computer Science and Engineering, Central South University, China. Her current research interests are in the area of data center networks.



Jianxin Wang received the B.E. and M.E. degrees in computer engineering from Central South University, China, in 1992 and 1996, respectively, and the PhD degree in computer science from Central South University, China, in 2001. He is the chair of and a professor in the School of Computer Science and Engineering, Central South University, Changsha, Hunan, P.R. China. His current research interests include algorithm analysis and optimization, parameraized algorithm, Bioinformatics and computer network. He is a senior member of IEEE.

Tian He received the PhD degree under Prof. John A. Stankovic from the University of Virginia, Charlottesville in 2004. He is currently a professor with the Department of Computer Science and Engineering at the University of Minnesota, Twin Cities. His research includes wireless sensor networks, cyber-physical systems, intelligent transportation systems, real-time embedded systems and distributed systems, supported by the US National Science Foundation, IBM, Microsoft and other agencies. He is the author and coauthor of over 200 papers in journals and conferences with over 20,000 citations (H-Index 59). His publications have been selected as graduate-level course materials by over 50 universities in the United States and other countries. He has received a number of research awards in the area of networking, including five best paper awards. He is also the recipient of the NSF CAREER Award 2009 and McKnight Land-Grant Professorship. He served a few program chair positions in international conferences and on many program committees, and also currently serves as an editorial board member for six international journals including IEEE Transactions on Computer. He is an ACM and IEEE fellow.