IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received February 2, 2019, accepted March 8, 2019, date of publication March 18, 2019, date of current version April 5, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2905850

PTCP: A Priority-Driven Congestion Control Algorithm to Tame TCP Incast in Data Centers

JIANHUI ZHUANG[1], XIANLIANG JIANG[1,2], GUANG JIN[1], JIAHUA ZHU[1], AND HAIMING CHEN[1], (Member, IEEE)

[1]Faculty of Electrical Engineering and Computer Science, Ningbo University, Ningbo 315211, China
[2]State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China

Corresponding author: Guang Jin (jinguang@nbu.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61601252, in part by the Public Technology Projects of Zhejiang Province under Grant LGG18F020007, in part by the Zhejiang Provincial Natural Science Foundation of China under Grant LY18F020011, in part by the Ningbo Natural Science Foundation under Grant 2017A610129 and Grant 2017A610116, in part by the Open Foundation of State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, under Grant SKLNST-2016-2-13, and in part by the K. C. Wong Magna Fund in Ningbo University.

ABSTRACT Data centers have become a prevalent infrastructure to host a large number of services, such as social networking and Web search, which generally have rigorous delay requirements. Even a slight increase in delay can seriously affect the performance of applications. Therefore, there have been several efforts focusing on minimizing the flow completion time (FCT) to meet the delay requirements, such as ICTCP, which adjusts the receive window according to the available bandwidth and DCTCP that uses ECN to provide the feedback to end hosts in mixing workloads. However, both approaches are only for a specific scene and cannot effectively solve the problem that packets losses lead to timeouts due to TCP incast. Moreover, they all use the fair-share mechanism, which causes the increasing of the FCT for small flows, when background flows pour into the network. For example, background flows cause the increasing($\approx$ 5ms) of the FCT for the ICTCP in an incast experiment. To solve these problems, we propose a priority-driven congestion control algorithm, PTCP, which can effectively avoid the incast problem and improve the FCT in mixing workloads. The PTCP leverages the priority to adjust the receive window and control the interval time of ACK. To evaluate the PTCP, we conduct extensive experiments in NS-2. The simulation results show that the PTCP outperforms greatly previous schemes both in the incast scenario and mixing workloads.

INDEX TERMS Congestion, TCP incast, ACK control, data centers, priority.

I. INTRODUCTION

In recent years, the intensive development of information services has led to the migration of applications in the Internet to data centers. Modern data centers support a variety of cloud services and applications, such as distributed file system [1], web search, social networking, etc., which typically have soft real-time constraints [2], [3] and employ the Partition/Aggregate pattern as showed in Figure 1. Though TCP plays an important role in the data center applications, its design is still Internet-centric and is not suitable for high-bandwidth, low-latency environments such as data center networks [4]. A lot of recent work [5]–[7] try to minimize the flow completion time (FCT) by achieving a low queue occupancy. For example, DCTCP [5] reduces the queue length of switch buffer by an adaptive congestion control. pFabric [6] estimates the remaining flow size to achieve an approximate short-remaining-size first scheduling and the

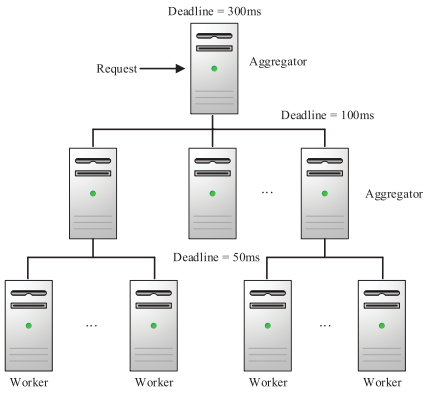


FIGURE 1. Example of the partition/aggregate pattern with different deadlines.

The associate editor coordinating the review of this manuscript and approving it for publication was Chuxiong Hu.

best performance in theory. PIAS [7] uses multiple priority queues and leverages a multi-level feedback queue mechanism to schedule flows. PIAS is closed to the shortest job first (SJF) scheduling because the small flow often finishes in the first few high priority queues. However, these methods cannot effectively solve the problem when the data packet losses lead to timeouts due to TCP incast [8]. TCP timeouts can cause delays of hundreds of milliseconds and throughput reductions of up to 90% [9]. This would result in performance degradation of cloud services and applications.

Due to the adverse effects of TCP incast, many researchers tried to solve it in terms of hardware or software. Depending on the strategy, the existing method can be divided into three categories. The first is window-based solutions such as ICTCP [10] and M21TCP [11]. These methods avoid the switch buffer overflow by adjusting the receive window. The second is fast recovery-based solutions, which mainly focus on fast return to the normal state when congestion occurs. They aim to reduce the value of retransmission timer and fast retransmission, such as reducing RTO_{min} [12], CP [13] and GIP [14]. Finally, solutions based on ACK control, such as IATCP [15] and PAC [16], reasonably delay the transmission of the ACK to adjust the sending rate of workers. They can support more concurrent connections and achieve high throughput, especially PAC whose maximum number of supported connections is about 1600. However, They do not take into account the impact of background traffic on the throughput of small traffic. In addition, they cannot adapt to the real data center environment for soft real-time applications such as MapReduce [17] and social networking.

This paper proposes a novel priority-driven congestion control algorithm called PTCP to address the incast problem and reduce the FCT in mixing workloads. PTCP adjusts the receive window in terms of the priority to control the sending rate of workers. Furthermore, we use an active ACK control to support more connections. The delay requirements can be well met in the face of soft real-time applications even with background flows. In addition, PTCP only makes minor modifications at the receiver, which does not affect the work of traditional TCP, and shows good compatibility. Simulation results show that PTCP can effectively avoid incast congestion as the number of worker increases. What's more, PTCP greatly outperforms previous schemes for TCP incast and realistic mixing workloads.

The rest of the paper is organized as follows. We introduce the related work in Section II. Section III provides the motivation for PTCP. Then we propose PTCP congestion control algorithm in Section IV. Section V describes the experimental setup and simulation results with the NS-2 simulator. Finally, the conclusion is presented in Section VI.

II. RELATED WORK

This section will firstly reveal the incast problem. Then we analyze the existing solutions especially about how to minimize the FCT in realistic mixing workloads.

A. INCAST PROBLEM

TCP incast problem was first discovered by Nagle *et al.* [18] in a scalable storage architecture. They increase the bandwidth of a cluster of small files accessed in parallel and observe that the throughput decreases when multiple senders send data to the same single client. This phenomenon, called TCP incast, has been observed empirically in data centers, which is inseparable from its two key features. First, the many-to-one pattern is widely applied in data center applications such as MapReduce, web search and cluster-based storage workload [19]. Second, shallow buffer switches are commonly used in data centers to achieve high bandwidth and low latency. Figure 2 shows an example of the incast scenario. When such bursts from multiple senders are aggregated to the same switch port, it will cause the switch buffer overflow quickly, resulting in packet losses which lead to TCP timeouts, which causes the performance degradation of cloud services and applications.

B. EXISTING CONGESTION CONTROL ALGORITHM

As mentioned above, there are many protocols for incast and congestion control in data centers nowadays. This subsection mainly focuses on introducing their principles and deficiencies.

ICTCP [10] only dynamically adjusts the size of the receive window. It only increases the receive window when the link has enough available bandwidth and the measured bandwidth is close enough to the expected bandwidth. M21TCP [7] calculates the maximum congestion window in every RTT at the switch. M21TCP informs all parallel senders to send the maximum rate of packets, which will not cause the buffer overflow. IATCP [15] is a congestion avoidance algorithm that controls the number of packets in the network pipe to achieve high throughput. IATCP adjusts the ACK interval to control the sending rate of the workers. PAC [16] uses the switch buffer size as the threshold. When the in_flight traffic reaches the threshold, the receiver delays the transmission of the ACK to reduce the amount of data packets in the network. However, both IATCP and PAC need to accurately estimate the in_flight traffic. The CP [13] protocol is based on accurate packet loss notifications. When the switch is overloaded, the sender of the lost message will be accurately informed through a method similar to TCP SACK. The sender retransmits the lost message and adjusts the transmission rate. The GIP [14] protocol detects the boundary of the stripe units. It uses the boundary information adjusting the congestion window to avoid timeouts caused by the window message loss and ACK missing. However, both GIP and CP need to modify the TCP/IP stack. What's more, the CP even needs to modify the switch hardware.

To minimize the flow completion time, lots of prior work assume the flow information is knowable, e.g., flow size. For example, PDQ [20] and pFabric [6], all assume the flow size is a priori and use the approximate SJF scheduling to minimize the average FCT. However, the flow size is difficult to be obtained in practice. DCTCP [5] uses ECN [21] to

perceive the congestion information. It adjusts the congestion window at the sender according to the congestion signal, thus achieve high burst tolerance and throughput under commodity switches. It also effectively reduces the FCT in mixing workloads.

III. MOTIVATING EXPERIMENTS

Our primary goal is to design an effective congestion control mechanism for both incast scenario and realistic mixing workloads.

Some solutions for incast perform inefficiently with background flows, e.g., ICTCP and IATCP. There are two essential reasons accounting for this. First, they are only for incast scenario other than mixing workloads. Second, they mainly use the fair-share mechanism at the receiver to allocate the window size, which is not friendly to small flows. To demonstrate this conflict, in the experiment, we use the many-to-one communication pattern with background flows similar to Figure 2. Each sender sends 64KB data to the same receiver. We measure the FCT for small flows with different connections and the behaviors are shown in Figure 3. The experiment result shows that the background flows cause the increasing($\approx$ 5ms) of the FCT of ICTCP in the scenario without incast.

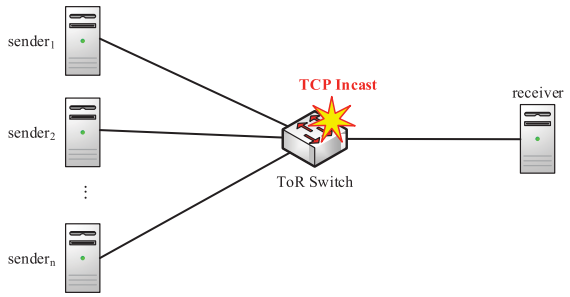


FIGURE 2. The classical scenario showing multiple senders communicating with a single receiver through a bottleneck Link.

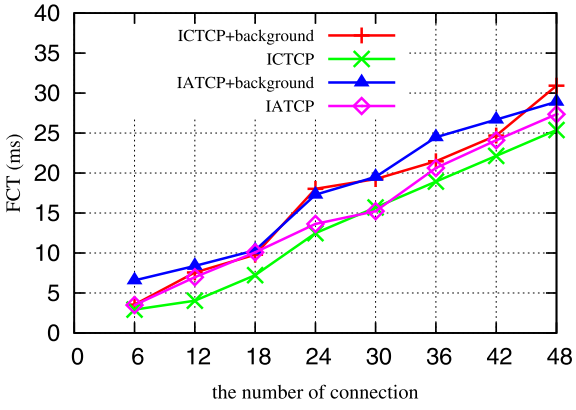


FIGURE 3. The impact of background flow on FCT.

In data centers, some schemes focus on minimizing the FCT such as DCTCP. Even though DCTCP ordinarily achieve

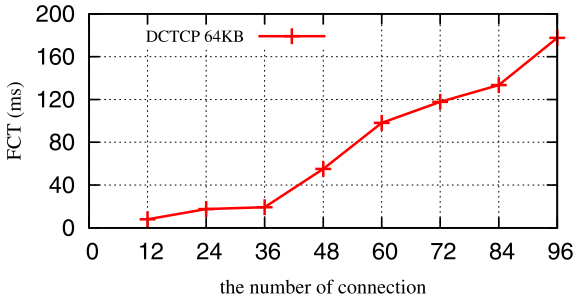


FIGURE 4. FCT with TCP incast under per server volume 64KB.

the low average FCT in mixing workloads, the performance is not so good in the TCP incast scenario. DCTCP is prone to produce timeouts in the many-to-one communication pattern, because of shallow switch buffer. To verify this, we design an incast experiment that each sender sends 64KB data to a receiver and the link bandwidth is 1Gbps. We increase the number of connection and the result is shown in Figure 4. We observe that the FCT of DCTCP increases rapidly because of TCP incast when the number of connection is greater than 36. For TCP incast flows, the number of connections supported by DCTCP is limited. Suppose that there are N incast flows in an RTT interval, the capacity of bottleneck link is L_c and the buffer size is B_s. We define w_i as the window size of flow i. The queue length Q is given by:

$$Q = \sum_{i=1}^{n} w_i - L_c * RTT \tag{1}$$

Assume that the window size of each flow is equal, i.e., $w_1 = w_2 = \cdots = w_N = w$. Therefore we can simplify the equation (1) as follows:

$$Q = N * w - L_c * RTT \tag{2}$$

To avoid the incast congestion, we must guarantee the value of Q is smaller than B_s. Therefore, the number of connections supported by DCTCP can be given by:

$$N \leq (B_s + L_c * RTT)/w \tag{3}$$

When w achieves a minimum, the number of connection reaches its maximum. In the incast experiment, the switch buffer size is about 150KB, the RTT is 100μs, and the minimum w for DCTCP is 4.5KB. Therefore, its theoretical maximum number of connection is only about 36.

IV. DESIGN OF PTCP

This section will introduce how to address the TCP incast congestion and minimize the FCT. In SectionIV-A, we describe how to compute the priority based on the flow size and deadline. SectionIV-B analyzes how to adjust the receive window according to the priority. SectionIV-C will introduce how to reasonably regulate the ACK intervals at the receiver among multiple flows.

A. PRIORITY CALCULATION

Many schemes [3], [22] have shown that it is a good choice to control the network congestion by the flow size or deadline. However, they follow the fair-share manner which causes the increasing of FCT for small flows. Thus, to well react to the rigorous delay requirements for applications, it is meaningful to set the priority for each flow to tame TCP incast. In this paper, the priority is defined based on the flow size and deadline. However, the flow size is usually agnostic in data centers. To calculate the priority of each flow, we need to estimate the current flow size and detect the deadline. Note that we want to estimate the flow size and detect the deadline without additional overheads. Thus, we do not consider complex prediction algorithms and measurement algorithms to estimate the flow size and obtain the deadline in this paper.

1) ESTIMATING THE FLOW SIZE

The DCTCP [5] states that the typical traffic size of Partition/Aggregation pattern is between 2KB and 20KB, and the background flow size is more than 10MB. In data centers, more than 80% of the flows less than 10KB follow heavy-tailed distribution. Therefore, we use the number of bytes sent to distinguish between small flows and large flows without the flow size information, and it is easy to count the number of bytes sent at the receiver when packets are delivered.

2) CALCULATING THE PRIORITY OF THE FLOW SIZE

We classify the flow size <= 10KB small, > 10KB and <= 10MB medium and >= 10MB large. We expect the small flows can get the higher priority compared to medium flows and large flows. Therefore, the priority of small flows is set to 1. We simply set the priority of the large flows to 0.1 rather than 0. On one hand, we prevent the throughput of the large flows from being too small, and on the other hand, large flows reserve the room for small flows. For the priority of the medium flows, we expect to choose a curve that well matches the following features:

- The curve should be monotonically decreasing and the range is 0.1 to 1.
- We expect the flow whose size is close to 10KB to get a priority which approximates the priority of small flows.
- We hope the medium flow can quickly lower the priority to match the short flow size first strategy.
- The flow whose size approaches to 10MB can avoid throughput underflow caused by low priority.

To avoid the largeness of the curve definition domain, we simply divide both sides by 1MB. Then, the range of flow size is 0 to 10. Therefore, we define the curve S as follows:

$$S(x) = \frac{1}{15.5 - 14.5 * e^{-x}} \quad x \in (0, 10) \tag{4}$$

Analysis: To prove the monotonicity of curve S. We take the derivative of the variable x in Equation 4. We have:

$$\frac{dS}{dx} = \frac{-14.5 * e^{-x}}{(15.5 - 14.5 * e^{-x})^2} \leq 0 \quad x \in (0, 10) \tag{5}$$

Therefore, the monotonic decreasing of the curve S is constant. We can obtain the maximum and minimum value at 0 and 10, respectively, with values of approximately 1, 0.1. Thus, we set the priority of flow size called P_1 as follows:

$$P_1 = \begin{cases} 1 & \text{if } n = 0 \\ \dfrac{1}{15.5 - 14.5 * e^{-x}} & \text{if } n \in (0, 10) \\ 0.1 & \text{if } n = 10 \end{cases} \tag{6}$$

3) CALCULATING THE PRIORITY OF THE DEADLINE

Deadline is generated by the application. We assume that the deadline can be obtained before the data deliver begins. Our basic strategy is to give higher priority to the tight deadline flows compared to non-deadline flows which have no requirements for the deadline in flow completion time. Supposing that the flow which misses the deadline is pointless. Thus, we drop the flow that nearly misses its deadline. We expect that the flow also completes within the deadline by the priority of deadline P_2 when the network congestion occurs. We find that much recent work such as D^{2}TCP and DIATCP, all adjust the window size by the remaining time of a flow until its deadline expires. It is easy to measure the remaining time because the start time of a flow can be obtained at the TCP sender. Therefore, we set the priority based on the deadline as follows:

$$P_2 = 1.58 * e^{-T_c/D} - 0.58 \tag{7}$$

T_c is the remaining time for a flow until its deadline expires, and D is the value of the deadline for a flow. In equation (7), we can see that the value of P_2 decreases monotonously with the increasing of T_c and its range is between 0 and 1. Now if the flow can just meet its deadline, then $P_2 > 0$ is appropriate. If the value of T_c approaches to 0, it means that the flow needs to be completed as soon as possible to avoid missing the deadline, then we increase the value of P_2 to complete the flow transmission quickly. If $T_c = 0$, it means the flow can not finish transmission within the deadline, we drop the flow.

Now we obtain the priority of the flow size and the priority of the deadline. Our basic strategy is the higher-priority first. And the total priority called P_{sum} is as follows:

$$P_{sum} = \max(P_1, P_2) \tag{8}$$

If the value of P_{sum} approaches to 1, then senders should send packets as many as possible. If the value of P_{sum} approaches to 0.1, senders need to reduce the number of packets sent so that higher-priority connections can allocate more bandwidth.

B. RECEIVE WINDOW SIZE SETTING

Much recent work [22]–[24] have shown that the queuing delay is an important characteristic that reflects the state of

network congestion. They focus on minimizing the queuing delay according to the accurate feedback such as ECN. Therefore, we use the queuing delay to estimate the current network congestion. When the receiver detects the queuing delay increasing, the receiver notifies senders decreasing its sending rate by ACK.

1) MEASURING THE QUEUING DELAY

The queuing delay is estimated as the difference between the instantaneous delay and base delay. The measurement of delay is the one-way delay instead of the round-trip time. The reason is that the one-way delay can prevent uncorrelated traffic on the reverse path from interfering with data transmission. The measurement of the queuing delay can be summarized in the pseudocode in Algorithm 1.

Algorithm 1 Measuring the Queuing Delay

```
1:  Parameters:
2:  base_delay: an initial one-way delay value
3:  current_delay: current measured one-way delay value
4:  current_time: current time the packet is received at the
    receiver
5:  send_time: the time packet is sent
6:
7:  Calculating the queuing delay at the receiver:
8:  if timestamp == True then
9:      current_delay ← current_time − send_time
10:     base_delay ← min(base_delay, current_delay)
11:     queuing_delay ← current_delay − base_delay
12: else
13:     timestamp ← True
14: end if
```

2) ESTIMATING THE NETWORK CONGESTION

Majority of existing work use congestion signals to reduce the sending rate of workers, e.g., DCTCP, D^{2}TCP, and D-SRTF. They all leverage ECN which has been well supported in switches, to deliver the congestion information. If the queue length is larger than the threshold, packets in the switch will be marked by ECN. When a receiver receives the marked packet, the ACK will also be marked. Then if senders receive the ACK, they will calculate the proportion of ACK packets marked in the previous window, and estimate the network congestion. Different from the ECN scheme, we use the queuing delay, as mentioned above, to estimate the network congestion. When the queuing delay exceeds the threshold, the network is in a state of congestion. Algorithm 2 presents the steps of the network congestion estimation performed at the receiver. First, we set the *Target* as the threshold for the queuing delay. Second, we count the number of packet only when the queuing delay exceeds the threshold (12-14). Finally, we calculate the network congestion and smooth it using the exponential filter (16-22). And the g is the sliding average coefficient (set value 0.0625). We set the *Target* and *interval* to 120μs, 20 in 1G network, and 100μs,65 in 10G network, respectively. Here the interval, 20 and 65, means the threshold for the number of packets received per round.

Algorithm 2 Estimating Network Congestion α

```
1:  Parameters:
2:  Target: an threshold for the queuing delay
3:  packet_count: number of packets received by the receiver
4:  packet_num: number of packets exceeding the threshold
5:  interval: calculate α every few packets
6:
7:  Initialization for global variable:
8:  packet_count ← 0
9:  packet_num ← 0
10:
11: Receive a packet at the receiver:
12: if queuing_delay ≥ Target then
13:     packet_num ← packet_num + 1
14: end if
15:
16: if packet_count mod interval == 0 then
17:     F ← packet_num/interval
18:     α ← (1 − g) ∗ α + F ∗ g
19:     Reset global information:
20:     packet_num ← 0
21:     packet_count ← 0
22: end if
23:
24: send the ACK packet
```

3) CALCULATING THE RECEIVE WINDOW

PTCP controls the receive window and informs the sender how many packets should be sent, which can be inferred from the ACK. Our strategy is to allocate the large window size for small flows or tight-deadline flows. The window size is only adjusted at the receiver by the priority. Inspired by DCTCP, we set the receive window called *rwnd* as follows:

$$rwnd = \begin{cases} (1 - \alpha^{P_{sum}}/2) * rwnd & \alpha > 0 \\ rwnd + 1 & \alpha = 0 \end{cases} \tag{9}$$

In the case of $\alpha = 0$, the window size adds a segment similar to TCP. And when the queuing delay of all packet feedback exceed the threshold, $\alpha = 1$, the window size is reduced to half. Similarly, if $0 < \alpha < 1$, the window size is modulated by P_{sum}.

C. ACK REGULATION

The receiver uses the ACK control to adjust their ACK interval. Our goal is to avoid the switch buffer overflow through the ACK adjustment when the network encounters congestion. To some extent, the queuing delay reflects the data of the switch buffer backlog, which represents the mismatching between the congestion window size and the

pipeline capacity. So we use the queuing delay as the indicator of ACK delay. However, it is obviously unreasonable to delay the entire queuing delay at any time, because it may cause the buffer underflow and under-utilization of the link capacity [25]. Thus, we use α and P_{sum} to determine the degree of ACK delay. The receiver ACK adjustment can be defined by ACK_{delay} as follows:

$$ACK_{delay} = \alpha^{P_{sum}} * Queue_{delay} \tag{10}$$

where $Queue_{delay}$ represents the queuing delay, which is given by Algorithm 1. In the case of $\alpha = 0$, there is no congestion in the network, the value of ACK delay is 0. If $\alpha = 1$, the network is in serious congestion and then we extend entire queuing delay by Eq. (5). And for α between 0 and 1, the value of ACK delay is modulated by α and P_{sum}, which can ensure that the switch queue length is always greater than 0.

V. SIMULATION RESULTS

We use NS-2(v2.35) [26] to evaluate the proposed PTCP by comparing its performance with DCTCP, IATCP, ICTCP, and M21TCP. The simulations are performed in both 1G and 10G network. We use the traffic generator described in Figure 5, to run the experiments with realistic traffic workloads.

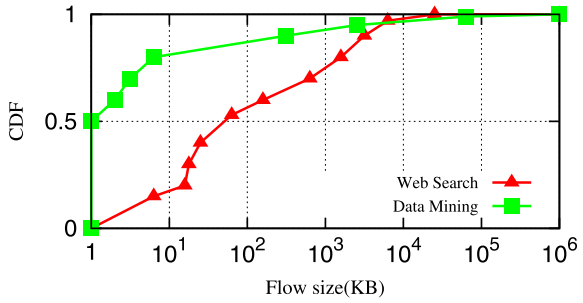


FIGURE 5. Traffic distributions used for evaluation.

All parameters of the simulations are listed in Table 1. For PTCP, the threshold *Target* is set to 120μs in 1G network, and 100μs in 10G network, respectively. For DCTCP, we set the marking threshold equal to the base BDP. For ICTCP, IATCP and M21TCP, the best parameters are set according to the simulation. The value of RTO_{min} is all set to 200ms.

TABLE 1. Simulation settings.

Parameter	1G Network	10G Network
MSS	1460 Bytes	1460 Bytes
Switch	96KB	375KB
Base RTT	100μs	100μs
Link delay	25μs	25μs
RTO_{min}	200ms	200ms
DCTCP	Initial window=2MSS $g = 0.0625$ $K = 10$	Initial window=2MSS $g = 0.0625$ $K = 65$
ICTCP	Initial window=2MSS $\gamma_1 = 0.1$ $\gamma_2 = 0.5$	Initial window=2MSS $\gamma_1 = 0.1$ $\gamma_2 = 0.5$
PTCP	$Target = 120\mu s$ $interval = 20$ Initial window=1MSS	$Target = 100\mu s$ $interval = 65$ Initial window=1MSS

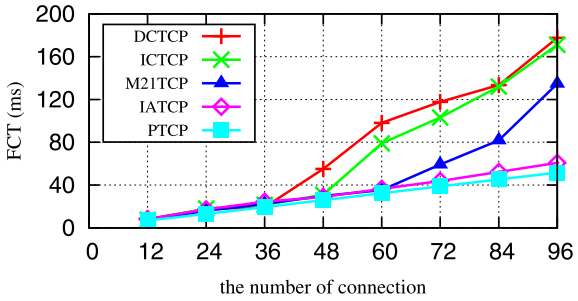


FIGURE 6. The FCT of different schemes in many-to-one scenario without background traffic.

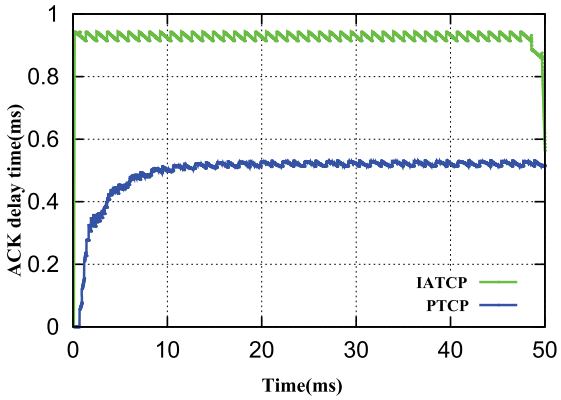


FIGURE 7. The ACK delay of PTCP, IATCP in many-to-one scenario without background traffic.

We evaluate the performance of the PTCP in the following three aspects:

- The Many-to-one scenario without background traffic and with background traffic, respectively.
- The background traffic scenario without incast.
- The all-to-all scenario.

A. MANY-TO-ONE SCENARIO WITHOUT BACKGROUND TRAFFIC

We evaluate the performance of PTCP compared to IATCP, ICTCP, DCTCP and M21TCP under fixed volume per server workload. The network topology is similar to Figure 2. One receiver and multiple senders are connected to the same ToR switch. We measure the FCT by setting the fixed volume to 64KB per round. The experiment results are the average of 20 rounds.

The FCT of DCTCP, IATCP, ICTCP, M21TCP, and PTCP are shown in Figure 6. When the number of connection is small, all algorithms perform well. However, the FCT of DCTCP, ICTCP, and M21TCP increase rapidly as the number of connection increases. Especially DCTCP shows the good performance in mixing workloads [5] and achieves the low average FCT. However, the performance of DCTCP is not very satisfactory in the many-to-one scenario because of TCP

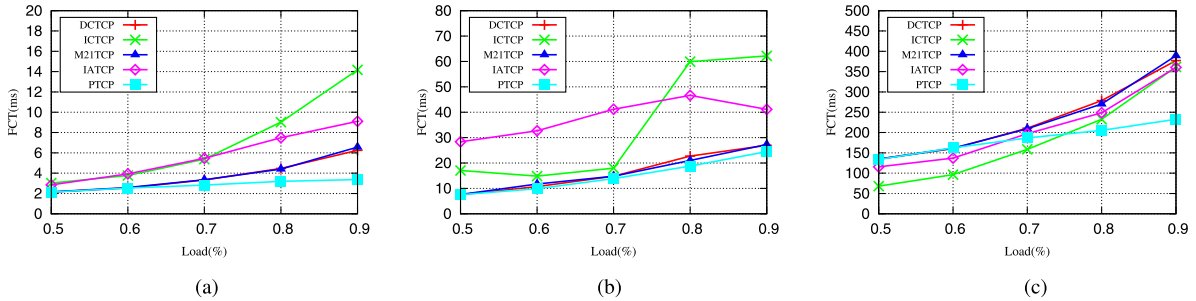


FIGURE 8. [Websearch Load]: The FCT of different schemes in many-to-one scenario with background traffic. (a) Small flow:Average. (b) Small flow:99th Percentile. (c) All flows: Average.

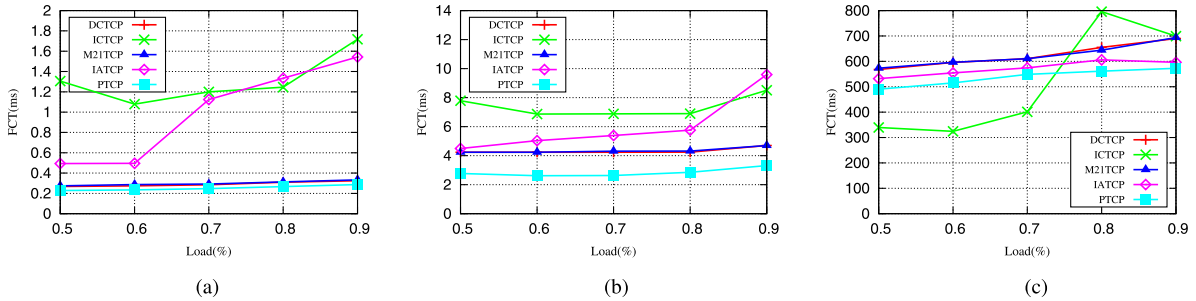


FIGURE 9. [Datamining Load]: The FCT of different schemes in many-to-one scenario with background traffic. (a) Small flow: average. (b) Small flow: 99th percentile. (c) All flows: average.

incast when the number of connection is more than 36. TCP incast causes delays of hundreds of milliseconds, which will dramatically increase the flow completion time. IATCP and PTCP show better performance in this scenario compared to several other schemes. They both leverage ACK control to avoid incast. However, the performance of PTCP is slightly better than IATCP because IATCP does not reasonably estimate the capacity of the network link, which causes excessive ACK delay. The unreasonable ACK delay will result in low link utilization [16].

To verify this, we conduct a small-scale simulation consisting of 96 senders. Each sender sends 64KB data to the same receiver. Figure 7 shows the ACK delay over time. The result shows that the delay time of IATCP is almost twice that of PTCP. Our algorithm uses conservative ACK delay, which can avoid the increasing of FCT.

B. MANY-TO-ONE SECNARIO WITH BACKGROUND TRAFFIC

We set one receiver and 96 senders connected to the same ToR switch and run 1000 flows in each setting. We evaluate the performance of the 99th percentile FCT for small flows, average FCT for small flows and average FCT for all flows, respectively. We apply two realistic workloads, a websearch workload and a datamining workload, based on measurements from production data centers.

Figure 8 and Figure 9 show the average FCT for small flows, 99th percentile FCT for small flows, and average FCT for all flows, respectively; for the websearch and

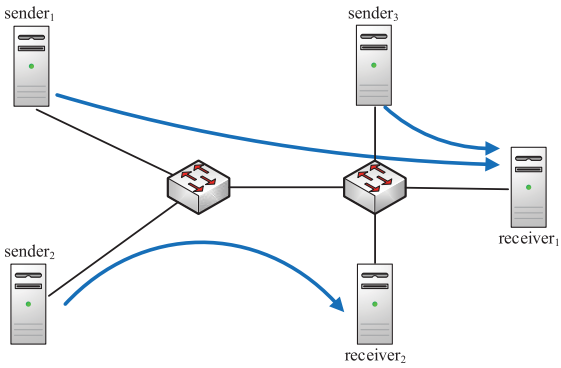


FIGURE 10. The scenario for no incast with background traffic.

datamining workloads, respectively. Three characteristics can be observed from Figure 8 and 9. First, PTCP achieves the best performace for small flows. Compared to DCTCP, ICTCP, M21TCP, and IATCP, PTCP reduces the average FCT of small flows by $\sim$ (0.5-46%, 30-76%, 1-48%, 25-63%) for websearch workload and $\sim$ (12-15%, 78-83%, 14-17%, 54-81%) for datamining workload. Second, the performance improvement of PTCP on the 99th percentile FCT of small flows is also obvious: $\sim$ (0.2-10%, 23-61%, 10-16%, 40-60%) for websearch workload and $\sim$ (30-34%, 62-65%, 29-39%, 38-65%) for datamining workload. Third, from Figure 8(c), we observe that the performance of PTCP is slightly better than DCTCP and M21TCP, while it is worse than ICTCP and IATCP when the network load is less than

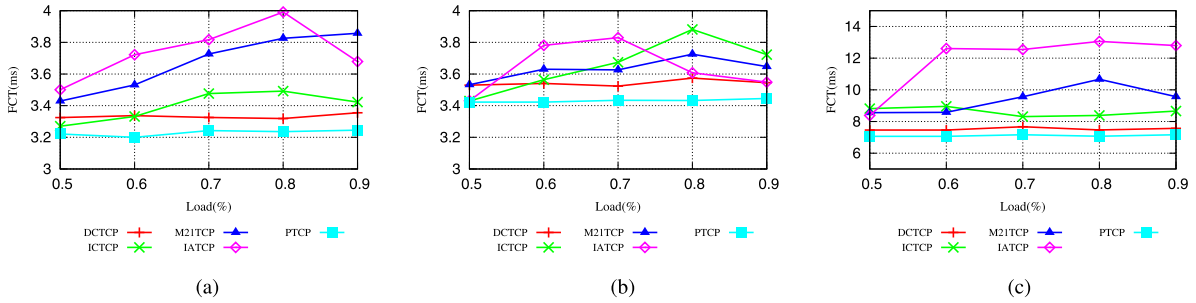


FIGURE 11. [Websearch Load]: The FCT of different schemes in the background traffic scenario without incast. (a) Small flow: average. (b) Small flow: 50th percentile. (c) Small flow: 99th percentile.

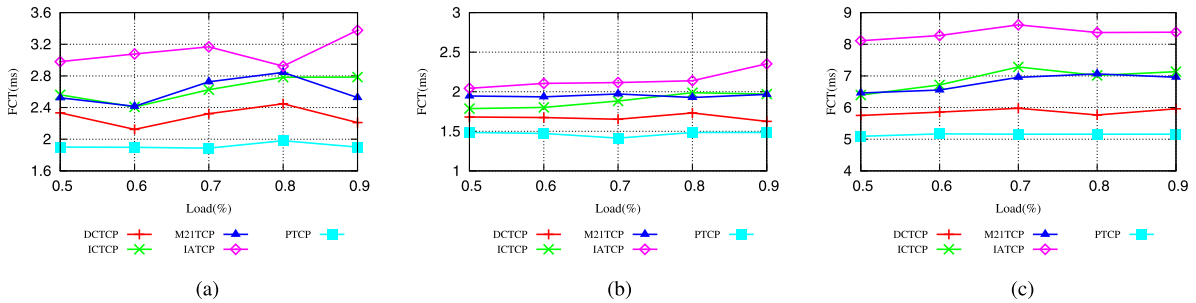


FIGURE 12. [Datamining load]: The FCT of different schemes in the background traffic scenario without incast. (a) Small flow: average. (b) Small flow: 50th percentile. (c) Small flow: 99th percentile.

0.7. In Figure 9(c), the overall FCT of PTCP is only slightly worse than ICTCP in datamining workload. This is because PTCP prioritizes small flows over long flows. Since small flows in data centers are more sensitive to delay, it is meaningful to appropriately reduce the priority of large flows.

C. THE BACKGROUND TRAFFIC SCENARIO WITHOUT INCAST

To verify the influence of background traffic on the Partition/Aggregate pattern, we conduct a test experiment in the background traffic scenario without incast. The network topology is similar to Figure 10. The sender$_2$ and sender$_3$ send small data to the receiver$_2$ and receiver$_1$, respectively. And sender$_1$ sends large data to receiver$_1$. The traffic generator is described in Figure 5 and comparison results are shown in Figure 11 and Figure 12.

The experiment results show that PTCP performs better than DCTCP, and greatly outperforms ICTCP, IATCP and M21TCP for small flows. PTCP improves the average FCT of small flows by $\sim$ (2-8%, 8-19%, 6-16%) for websearch workload, and $\sim$ (21-29%, 32-44%, 21-30%) for datamining workload when compared to ICTCP, IATCP, and M21TCP. In addition, the performance of PTCP over ICTCP, IATCP, and M21TCP in the 50th percentile FCT, and 99th percentile FCT of small flows is also obvious. For example, from Figure 11(c) and Figure 12(c), we can see that PTCP improves the 99th percentile FCT by $\sim$ (14-21%, 16-43%, 17-33%) for websearch workload and $\sim$ (20-29%, 37-40%, 21-27%) for datamining workload. The reason is that ICTCP, IATCP, and M21TCP cannot react to the background traffic when it pours into the link. The background traffic continues to occupy the link bandwidth causing the increasing of the completion time for small flows. However, for websearch and datamining workloads, the performance improvement is less significant when compared to DCTCP. For example, for the websearch workload, PTCP improves the average FCT for small flows by 3-4%, the 50th percentile FCT by 2-4%, and the 99th percentile FCT by only 1%. Because PTCP and DCTCP both estimate the network congestion to control the sending rate of workers. DCTCP uses ECN to maintain a low queue occupancy for improving the FCT. Therefore, they show similar performance in small flows.

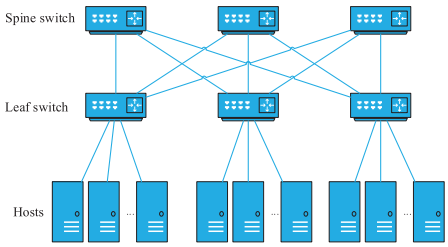


FIGURE 13. DCN architecture: leaf-spine topology.

D. ALL-TO-ALL SCENARIO

To estimate the performance of PTCP in today's data centers, we setup a leaf-spine topology with 9 leaves, 4 spines [27]. Each leaf is connected to 16 hosts using 10Gbps links and

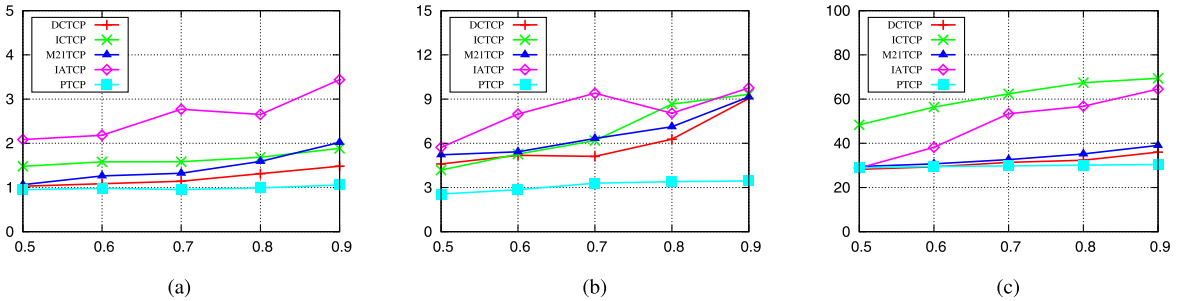


FIGURE 14. [Websearch Load]: The FCT of different schemes in all-to-all scenario. (a) Small flow: average. (b) Small flow:99th percentile. (c) All flows: average.

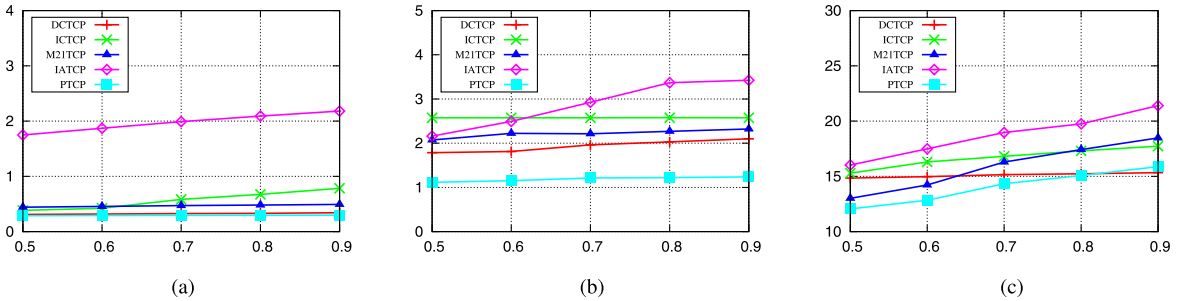


FIGURE 15. [Datamining Load]: The FCT of different schemes in all-to-all scenario. (a) Small flow: average. (b) Small flow: 99th percentile. (c) All flows: average.

4 spine switches using 40Gbps links. The default parameters are listed in Table 1 and the network topology is shown in Figure 13. The flow size comes from websearch and datamining distribution in Figure 5. We run 10,000 flows for each simulation. Figure 14 and Figure 15 show the average FCT for small flows, 99th percentile FCT for small flows, and average FCT for all flows, respectively; for the websearch and datamining workloads, respectively. The experiment results show that PTCP still keeps its superiority both for websearch workload and datamining workload when compared to previous schemes.

VI. CONCLUSION

This paper has proposed a priority-driven congestion control algorithm called PTCP to improve TCP performance both for TCP incast scenario and mixing workloads in data center networks. Differing from the previous approaches, PTCP leverages priority to achieve the window adjustment and ACK controls to prevent packet drops which lead to timeouts. The previous schemes focus on addressing problems in a specific scenario such as incast scenario or mixing workloads, but they perform inefficiently in other scenarios. Simulation results show that PTCP achieves lower flow completion time and can support more connections in the incast scenario, when it is compared with previous schemes. What's more, our algorithm shows better performance in flow completion time in mixing workloads. In addition, PTCP requires little system modification, making it readily deployable in production data centers.

REFERENCES

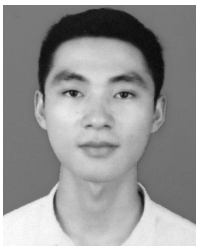
[1] A. Hammadi and L. Mhamdi, "A survey on architectures and energy efficiency in data center networks," *Comput. Commun.*, vol. 40, pp. 1–21, Mar. 2014.
[2] A. M. Abdelmoniem and B. Bensaou, "Curbing timeouts for TCP-incast in data centers via a cross-layer faster recovery mechanism," in *Proc. INFOCOM*, Honolulu, HI, USA, Apr. 2018, pp. 675–683.
[3] J. Hwang, J. Yoo, and N. Choi, "Deadline and incast aware TCP for cloud data center networks," *Comput. Netw.*, vol. 68, pp. 20–34, Aug. 2014.
[4] T. Benson, A. Anand, A. Akella, and M. Zhang, "Understanding data center traffic characteristics," in *Proc. ACM SIGCOMM Workshop*, Barcelona, Spain, Aug. 2009, pp. 65–72.
[5] M. Alizadeh *et al.*, "Data center TCP (DCTCP)," in *Proc. SIGCOMM*, New Delhi, India, Aug. 2010, pp. 63–74.
[6] M. Alizadeh *et al.*, "pFabric: Minimal near-optimal datacenter transport," in *Proc. SIGCOMM*, Hong Kong, Aug. 2013, pp. 435–446.
[7] W. Bai, L. Chen, K. Chen, D. Han, C. Tian, and H. Wang, "Information-agnostic flow scheduling for commodity data centers," in *Proc. NSDI*, Hong Kong, May 2015, pp. 455–468.
[8] L. Xu, K. Xu, Y. Jiang, F. Ren, and H. Wang, "Throughput optimization of TCP incast congestion control in large-scale datacenter networks," *Comput. Netw.*, vol. 124, pp. 46–60, Sep. 2017.
[9] V. Vasudevan *et al.*, "Safe and effective fine-grained TCP retransmissions for datacenter communication," in *Proc. SIGCOMM*, Barcelona, Spain, Aug. 2009, pp. 303–314.
[10] H. Wu, Z. Feng, C. Guo, and Y. Zhang, "ICTCP: Incast congestion control for TCP in data-center networks," in *Proc. CoNEXT*, Philadelphia, PA, USA, Nov. 2010, pp. 1–13.
[11] A. Adesanmi and L. Mhamdi, "M21TCP: Overcoming TCP incast congestion in data centres," in *Proc. IEEE Int. Conf. Cloud Netw.*, Niagara Falls, ON, Canada, Oct. 2015, pp. 20–25.
[12] E. Gonzalez, S. Mcclellan, and W. Peng, "RTOmin as a balancing parameter between fast retransmissions and timeouts within stream control transmission protocol (SCTP)," in *Proc. ICSAI*, Shanghai, China, Nov. 2014, pp. 687–691.
[13] P. Cheng, F. Ren, R. Shu, and C. Lin, "Catch the whole lot in an action: Rapid precise packet loss notification in data center," in *Proc. NSDI*, Seattle, WA, USA, Apr. 2014, pp. 17–28.

[14] J. Zhang, F. Ren, L. Tang, and C. Lin, "Taming TCP incast throughput collapse in data center networks," in *Proc. ICNP*, Goettingen, Germany, Oct. 2013, pp. 1–10.
[15] J. Hwang, J. Yoo, and N. Choi, "IA-TCP: A rate based incast-avoidance algorithm for TCP in data center networks," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Ottawa, ON, Canada, Jun. 2012, pp. 1292–1296.
[16] W. Bai, K. Chen, H. Wu, W. Lan, and Y. Zhao, "PAC: Taming TCP incast congestion using proactive ACK control," in *Proc. ICNP*, Washington, DC, USA, Oct. 2014, pp. 385–396.
[17] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in *Proc. USENIX OSDI*, San Francisco, CA, USA, Dec. 2004, pp. 137–149.
[18] D. Nagle, D. Serenyi, and A. Matthews, "The panasas activescale storage cluster: Delivering scalable high bandwidth storage," in *Proc. SC*, Washington, DC, USA, Nov. 2004, pp. 1–53.
[19] X. Hou, D. Pal, T. K. A. Kumar, J. P. Thomas, and H. Liu, "Privacy preserving rack-based dynamic workload balancing for Hadoop MapReduce," in *Proc. IEEE Int. Conf. Big Data Secur. Cloud*, New York, NY, USA, Apr. 2016, pp. 30–35.
[20] C. Y. Hong, M. Caesar, and P. Godfrey, "Finishing flows quickly with preemptive scheduling," in *Proc. SIGCOMM*, Helsinki, Finland, Aug. 2012, pp. 127–138.
[21] K. K. Ramakrishnan, S. Floyd, and D. L. Black, *The Addition of Explicit Congestion Notification (ECN) to IP*, document RFC 3168, RFC Editor, 2001.
[22] B. Vamanan, J. Hasan, and T. N. Vijaykumar, "Deadline-aware datacenter TCP (D^{2}TCP)," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 42, pp. 115–126, Oct. 2012.
[23] C. Lee, C. Park, K. Jang, S. Moon, and D. Han, "Accurate latency-based congestion feedback for datacenters," in *Proc. USENIX ATC*, Santa Clara, CA, USA, Jul. 2015, pp. 1–14.
[24] E. Ghazisaeedi and C. Huang, "EnergyMap: Energy-efficient embedding of MapReduce-based virtual networks and controlling incast queuing delay," in *Proc. ICCSN*, Beijing, China, Jun. 2016, pp. 698–702.
[25] D. Shan and F. Ren, "Improving ECN marking scheme with micro-burst traffic in data center networks," in *Proc. INFOCOM*, Atlanta, GA, USA, May 2017, pp. 1–9.
[26] *The Network Simulator NS-2*. [Online]. Available: http://www.isi.edu/nsnam/ns/
[27] M. Alizadeh *et al.*, "CONGA: Distributed congestion-aware load balancing for datacenters," in *Proc. SIGCOMM*, Chicago, IL, USA, Aug. 2014, pp. 503–514.

JIANHUI ZHUANG is currently pursuing the M.S. degree with the Faculty of Electrical Engineering and Computer Science, Ningbo University. His main research interests include data center networks, cloud computing, and wireless networking.

XIANLIANG JIANG received the B.E. degree from the University of Science and Technology of China, in 2009, the M.S. degree from Ningbo University, in 2012, and the Ph.D. degree in computer science and technology from Zhejiang University, in 2016. He is currently a Lecturer with Ningbo University. His research interests include protocol design, congestion control, and the Internet of Things.

GUANG JIN received the Ph.D. degree in computer science and technology from Zhejiang University. Since 2001, he has been with the Faculty of Electrical Engineering and Computer Science, Ningbo University, where he is currently a Full Professor. His research interests include wireless networking, network protocol, and the Internet of Things.



JIAHUA ZHU is currently pursuing the M.S. degree with the Faculty of Electrical Engineering and Computer Science, Ningbo University. His main research interests include network protocol design, congestion control, and the Internet of Things.

HAIMING CHEN (M'15) received the B.E. and M.E. degrees in computer engineering from Tianjin University, Tianjin, China, in 2003 and 2006, respectively, and the Ph.D. degree in computer science from the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China, in 2010. He is currently an Associate Professor with the Department of Computer Science, Ningbo University, Ningbo, China. His research interests include wireless, ad hoc, sensor networks, and networked embedded computing systems.