TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214 07/11 pp273-285
Volume 18, Number 3, June 2013

On Meeting Deadlines in Datacenter Networks

Li Chen, Baochun Li*, and Bo Li

Abstract: Datacenters have become increasingly important to host a diverse range of cloud applications with mixed workloads. Traditional applications hosted by datacenters are throughput-oriented without delay requirements, but newer generations of cloud applications, such as web search, recommendations, and social networking, typically employ a tree-based Partition-Aggregate structure, which may incur bursts of traffic. As a result, flows in these applications have stringent latency requirements, i.e., flow deadlines need to be met in order to achieve a satisfactory user experience. To meet these flow deadlines, research efforts in the recent literature have attempted to redesign flow and congestion control protocols that are specific to datacenter networks. In this paper, we focus on the new array of deadline-sensitive flow control protocols, thoroughly investigate their underlying design principles, analyze the evolution of their designs, and evaluate the tradeoffs involved in their design choices.

Key words: datacenter; transport protocols; design principles

1 Introduction

As large scale datacenters are increasingly used to host a diverse range of cloud applications, new requirements of datacenter applications have inspired new designs of datacenter network protocols. Traditionally, throughput oriented applications, such as bulk data transfers, are more demanding with respect to bandwidth. In contrast, a newer generation of cloud applications, such as web search and social networking, have stringent latency requirements, which is typically represented by *deadlines* associated with application flows. The performance of these deadline-sensitive applications may significantly affect the revenue of the enterprise running these applications, since they are designed to respond to user requests in a timely fashion. For example, for every 100 milliseconds of additional latency, Amazon will suffer from a 1% loss of sales[1].

These applications typically employ a tree-based Partition-Aggregate structure[2], which consists of a Top-Level Aggregator (TLA), a number of Mid-Level Aggregators (MLAs), and a number of workers, as shown in Fig. 1. The TLA receives requests from users and partitions the workload across the MLAs as his children nodes. In a similar vein, the workload is partitioned by the MLAs, all the way down to the workers, where computation is performed. The results are then sent back, aggregated by the MLAs and eventually by the TLA with a specific deadline. When the deadline arrives, the TLA will respond to users without waiting for the children who missed the deadline. Since the response is only based on flows from those children meeting their deadlines, the result will be incomplete, negatively affecting the service quality. To make matters worse, if flows fail to meet their deadlines, the bandwidth they used does not contribute to the correctness of the results.

A specific example of an application with the Partition-Aggregate structure is a typical Facebook page. The page consists of many components, such as instant messaging and event notification. The contents of these components are generated by a number of workers, and eventually assembled by the TLA to be

• Li Chen and Baochun Li are with the Department of Electrical and Computer Engineering, University of Toronto, Toronto M5S 3G4, Canada. E-mail: {lchen, bli}@eecg.toronto.edu
• Bo Li is with the Department of Computer Science, Hong Kong University of Science and Technology, Hongkong, China. E-mail: bli@cse.ust.hk
∗ To whom correspondence should be addressed.
Manuscript received: 2013-02-26; accepted: 2013-03-04

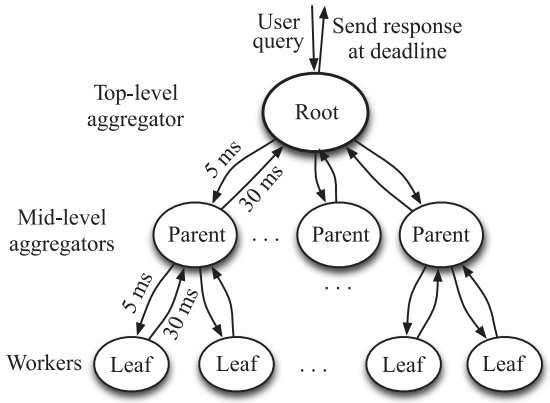


Fig. 1 The Partition-Aggregate structure with their associated deadlines.

presented to users. To achieve a satisfactory response time, the TLA is required to present the page to users within a specific deadline. If some flows are delayed when the deadline arrives, the TLA will have to present the incomplete contents it has aggregated. Hence, the quality of the page is degraded and the bandwidth consumed by the flows that have missed the deadline is wasted. Therefore, meeting flow deadlines is a critical objective for this type of applications.

Such a new objective of meeting flow deadlines has inspired researchers to reinvestigate the design of transport protocols in datacenter networks. The currently prevailing transport protocols like TCP, RCP[3], and ICTCP[4] are deadline agnostic. They strive to allocate bandwidth equally among flows to approximate fair sharing. In addition, flow scheduling mechanisms do not differentiate flows with different deadlines. The lack of awareness of flow deadlines causes a large number of flows to miss their deadlines, and the underlying reason is the tendency of treating flows equally to achieve fairness when congestion occurs.

Figures 2 and 3 have shown two examples to illustrate the necessity of deadline awareness. As shown in Fig. 2, flow 1 and flow 2 with the same size but different deadlines share a bottleneck link. Fair sharing allows them to finish at the same time, but flow 1 will miss its tighter deadline. Inverse prioritization of flows, i.e., scheduling flow 2 before flow 1, also causes flow 1 to miss its deadline. Only with the correct prioritization — by first allocating flow 1 with full bandwidth and then reallocating the bandwidth to the other flow — can these two flows both meet their deadlines. An extreme case, illustrated in Fig. 3, is when all flows sharing a bottleneck link have the same deadline. In

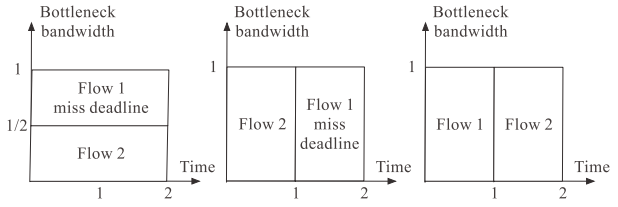


Fig. 2 Flow 1 and flow 2 have the (size, deadline) pairs (1, 1), (1, 2) respectively. Flow 1 has a tighter deadline and thus a higher priority. Fair sharing leads to flow 1 missing deadline. An inversion of priority also causes flow 1 to miss its deadline. Prioritizing flow 1 is the best solution that can meet the deadlines of both flows.

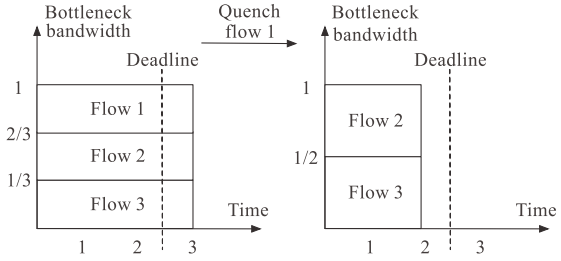


Fig. 3 Three flows have the same (size, deadline) pair (1, 1). Fair sharing results in all flows missing their deadlines. After eliminating a flow, the remaining flows can meet their deadlines.

this case, none of the flows will meet their deadlines when bottleneck bandwidth is fairly shared, which is a huge waste of bandwidth. A better way is to eliminate one flow and allow the remaining ones to meet their deadlines, so that a partial response can be generated.

Given the mismatch between the traditional design objective of fair sharing and the new flow deadline requirements, new transport protocols that are aware of flow deadlines need to be designed. The issue of meeting flow deadlines has drawn much attention from the literature, with deadline aware protocols proposed in the recent years[2, 5-8]. These new *deadline aware* protocols share a common design principle that gives priority to flows with tighter deadlines. When the network is congested, these deadline aware protocols will throttle those flows that can afford to wait, so that flows with tight deadlines can finish sooner and meet their deadlines.

In this paper, we survey existing network protocols in the literature that are designed for applications with a delay-sensitive nature, including both deadline aware protocols and deadline agnostic protocols, by illustrating their main design choices and ideas. In particular, we focus on analyzing the design principles and choices in the new array of *deadline aware* protocols. By discussing the advantages and

weaknesses of these protocols, by presenting their similarities and differences across-the-board, and by showing how these protocol designs evolve over time, we hope to bring forward some important insights and lessons learned in the context of deadline aware transport protocols in datacenter networks, so that better protocols can be designed in the future.

2 From Deadline Agnostic to Deadline Aware: Evolutions and Challenges

Before meeting flow deadlines becomes the primary design objective, network protocol designs have gone through two major stages. In the first stage, bandwidth efficiency is the primary concern in designing protocols for high-speed and long-distance networks. Due to the slow increase of the congestion window, conventional Additive Increase Multiplicative Decrease (AIMD) TCP fails to utilize bandwidth efficiently and causes substantial performance degradation. To address this issue, BIC[9], CUBIC[10], FAST[11], HSTCP[12], LTCP[13], and XCP[14] were proposed to mitigate the drastic reactions to packet losses in AIMD TCP, in order to achieve efficient bandwidth utilization and to maximize the network throughput.

In the second stage, the increasing popularity of delay-sensitive applications motivates a latency-oriented protocol design. Rate Control Protocol (RCP)[3] emulates processor sharing by assigning the same rate to all of the flows passing through a bottleneck link, with the attempt to minimize flow completion times. A faster response to TCP timeout is made possible by using high-resolution timers[15], to reduce flow latencies and to avoid incast collapse[16]. ICTCP[4] proactively adjusts the TCP receiver window to avoid packet drops, thus reducing flow completion times. DCTCP[17] and HULL[18] reduce flow completion times by controlling buffer occupancies and capping the link utilization below their capacities.

More recently, deadline requirements have become the primary design concern for transport protocols in the context of datacenter networks, since it is crucial to the performance of an important class of cloud applications with the Partition-Aggregate structure. In datacenter networks, flows without deadlines and flows with different deadlines contend for the bottleneck link when congestion occurs. A network protocol is deadline agnostic if it treats flows with deadlines and those without deadlines equally. In comparison, a deadline aware protocol gives higher priorities to flows with deadlines than those without, thus achieving better performance by meeting flow deadlines.

To design a network protocol with an awareness of deadlines, a number of new and important challenges need to be addressed in the context of datacenter networks. First, deadlines are associated with flows rather than with packets. Hence, traditional packet-level scheduling mechanisms such as Earliest Deadline First (EDF) can not be directly applied to deadline aware flow-level scheduling. Second, flows with deadlines are usually short flows (about 50 KB) with minimal Round-Trip Times (RTTs) (about 300 μs), which require scheduling decisions to be made very quickly. In this sense, reservation schemes proposed in IntServ[19] and DiffServ[20] are too heavy weighted. Also, traditional timeout mechanisms are not feasible for these short flows. Third, both flow deadlines and flow completion times can vary significantly due to multiple services and diverse traffic patterns. The long tailed distribution of flow completion times makes it difficult to meet flow deadlines. Finally, applications with the Partition-Aggregate structure are likely to cause fan-in bursts, since the children of a parent node may have the same deadline and send their responses to the parent at the same time. This will result in congestion and cause a large number of flows to miss their deadlines.

Faced with these challenges, we wish to investigate a number of fundamental design choices in the next section. Also, we will illustrate the solutions of existing works in subsequent sections, highlighting how they address these challenges.

3 Fundamental Choices

When designing deadline aware protocols, the first decision to be made is which mechanism to use and how to incorporate deadline awareness. We begin with an overview of the main mechanisms and strategies employed by deadline agnostic protocols, with the hope of offering some hints on deadline aware protocol designs.

Explicit rate control is employed in RCP[3] and XCP[14], where routers assign sending rates to senders in order to keep shorter queues and lower latencies. However, their practicality is a major limitation, since they have strict requirements on revising router hardware designs, which are quite expensive and not

practically feasible.

Vasudevan et al.[15] proposed a retransmission strategy to address the TCP incast problem[16] that causes flow completion times to increase. They used high-resolution timers to enable fine-grained TCP retransmissions, thus alleviating the negative impact of timeouts caused by the incast. However, it does not prevent the timeout. Hence, the high latency caused by the timeout has not been addressed.

RCP[3], ICTCP[4], XCP[14], and DCTCP[17] allocated link bandwidth equally among concurrent flows, approximating fair sharing. However, fair sharing is far from optimal with respect to minimizing flow completion times[21] and the number of flows whose deadlines can not be met[22].

Hedera[23] and MPTCP[24,25] provided a way to reduce latency by using multiple flow paths. Hedera performed flow scheduling by periodically remapping elephant flows to better paths. Taking a step further, MPTCP has made TCP multi-path aware, establishing multiple sub-flows on different paths and moving traffic off more congested paths to less congested ones. However, their operational timescales were coarse-grained, which required further improvement in order to reduce flow completion times.

Active Queue Management (AQM) schemes are employed in RED[26], PI[27], and E-TCP[28] to track congestion at a switch. When the switch buffer occupancy is high, packets will be randomly dropped, thus early warnings of congestion will be conveyed to TCP end hosts. In this way, senders can back off to avoid serious congestion and to achieve higher throughput and faster retransmission. Based on AQM, Explicit Congestion Notification (ECN)[3,29-31] enables congestion feedback through marking the Congestion Encountered (CE) bit in the IP header. Higher RTTs resulted from increased queuing delays are also viewed as congestion feedback by Vegas[32], CTCP[33], and FAST[11].

Proactive bandwidth reservation has been proposed in Ref. [34] to meet the latency requirement of real time multimedia traffic. There are also reactive schemes[35,36]. For example, TCP-RTM[35] improves latency related performance by minimizing packet reordering and packet losses in TCP.

Having gone through the mechanisms and strategies of deadline agnostic protocols, we now investigate the fundamental design choices involved in deadline aware protocol designs. The basic idea of deadline aware protocol designs is to build upon and incorporate deadline awareness into some of these mechanisms, in order to prioritize flows with deadlines. Hence, the first question we would like to answer is where and how to incorporate deadline awareness. We further present an overview of design choices within each mechanism. Given the alternatives, we will analyze and compare existing works in subsequent sections of this paper, to see what choices they have made, what benefits they have gained, and what disadvantages they have incurred.

3.1 Where and how to incorporate deadline awareness?

If a flow has multiple paths, it is more likely to meet its deadline by following the least congested path. This requires the senders to perform load distribution, and the routers to perform path selection.

On the other hand, if a flow is assumed to follow a single path, three alternative mechanisms can be used to design deadline aware protocols. One option is to make *flow scheduling* deadline aware. When flows contend for network bandwidth, a deadline aware scheduling should schedule flows with deadlines ahead of those without, and flows with closer deadlines should be scheduled ahead of those with deadlines that are farther away. This mechanism mainly involves operations at switches.

Another option is to incorporate deadline awareness into *congestion control* at senders. Receiving congestion feedback, senders will throttle flows differentially, according to the deadlines of flows they are sending. Flows without deadlines will back off aggressively to relinquish bandwidth, allowing flows with deadlines to complete sooner. In a similar vein, flows with closer deadlines should be throttled less than those with more relaxed deadlines.

The third option is to incorporate deadline awareness into *rate control*, which requires switches to allocate sending rates and senders to follow the allocated rates. While congestion control strives to react after congestion happens, rate control avoids congestion proactively through appropriate bandwidth reservations based on flow deadlines. The intent is to allow flows with deadlines to be sent at higher rates, so that they can complete sooner and meet their deadlines.

Both rate control and congestion control couple flow prioritization with the flow sending rate. In rate control, the flow sending rate is calculated by switches based

on states of concurrent flows. In congestion control, the flow sending rate is calculated by the sender based on congestion feedback, without explicit knowledge of other flows. In contrast, flow scheduling can decouple flow prioritization from the flow sending rate. When the switch buffer is full, it simply drops packets belonging to flows without deadlines or those with more relaxed deadlines, without the need to specify the flow sending rates.

3.2 How to generate feedback?

The basic idea of congestion control is that switches provide congestion feedback to senders and senders reduce their sending rates in response. Playing an important role in any congestion control mechanism, congestion feedback deserves to be designed carefully. There are three alternatives of feedback. First, upon detecting congestion, a switch may send a control message to one or more sources that are filling the queue, in order to stop these sources. However, transmitting control packets may add additional load to the network and exacerbate congestion. In addition, control packets are likely to be lost when congestion occurs. The second alternative is for the switch to mark forwarded packets when it notices congestion. Upon receiving these congestion signals, the senders can slow down to avoid the congestion. No additional packets are required in this alternative, but there may be a long delay before sender-side reactions take effect. The third alternative is to simply discard packets when congestion occurs. As a result, the senders of these packets will receive duplicated ACKs or experience a timeout. Though this alternative is very simple and reliable, the delay may again be quite long before senders notice the congestion and react to it.

3.3 Distributed or centralized?

For flow scheduling and rate control, whether to design a centralized controller or distributed controllers is an important choice to be made. With access to global states of the entire network, a centralized controller is able to obtain a nearly optimal solution for flow scheduling or rate allocation. However, it does not scale well. A centralized algorithm requires the maintenance of complete and up-to-date state information for all of the flows. Whenever new flows are started or existing flows terminate, there will be communication delays between the senders and the centralized controller. In datacenter networks where the majority of the flows are short and delay-sensitive, flow initiation overhead will be considerable and heavy weighted. With rapid flow arrivals and tight flow deadlines, there may not be sufficient time for the centralized controller to gather the up-to-date information of all the flows before making decisions. In addition, the centralized controller is a congested hot-spot. Controling messages to or from the centralized controller may congest the link that connects the controller to the network, and it introduces a single point of failure.

In contrast, a distributed algorithm is scalable and feasible, however, it is based on localized flow information. The key issue is to make sure that local decisions made separately are able to converge to achieve a global objective.

3.4 Fair or preemptive?

In traditional network protocol designs, fairness is an important metric. Ensuring fairness is of great significance in providing protection to well behaved users and discouraging ill behaved ones. However, in datacenter networks that carry delay-sensitive traffic, meeting flow deadlines becomes a primary concern, and fairness needs to be reinvestigated.

It is shown in an analysis[21] that at least 99% of jobs have shorter completion times under a preemptive scheduling discipline called Shortest Job First (SJF), than under fair sharing disciplines. Hence, unfairness actually improves the flow completion times. On the other hand, preemptive scheduling is unfair to flows with low priorities. If flows with high priorities frequently arrive, flows with lower priorities may be starved. It is necessary to evaluate such a tradeoff when designing a scheduling algorithm.

4 Reactive Sender-Side Back-Off: DCTCP vs. D^{2}TCP

Traditional AIMD TCP achieves remarkable success in the Internet, thanks to the simplicity and reliability of using packet drop as congestion feedback. However, TCP reacts to the presence of congestion, rather than its extent. This feature causes substantial underutilization of network bandwidth over high-speed long-distance networks. To better utilize network bandwidth, TCP variants are designed to make senders react to the extent of congestion, either indicated by packet drops or by increased RTTs. Recently, datacenters play an important role in hosting cloud applications with Partition-Aggregate structure, which typically

have stringent latency requirements. It is of great significance to customize TCP for datacenter networks.

Since datacenters host a diverse range of applications with mixed workloads, concurrent flows typically consist of delay-insensitive long background flows and delay-sensitive short flows. Long background flows demand a large amount of bandwidth and build up queues at switches, thus threatening the performance of short flows. Moreover, switches in datacenters have shallow buffers. If queues build up, there would not be sufficient buffer space to absorb bursts of traffic, resulting in increased delays.

In this context, DCTCP[17,37] and D^{2}TCP[6] were both proposed to maintain low queues through the sender-side back-off mechanism, to meet requirements of delay-sensitive applications. They are both TCP friendly, inheriting most features of TCP such as slow start, retransmission, and recovery from packet losses. The only change is the congestion window modification based on congestion feedback in every RTT.

DCTCP provides a congestion control scheme that utilizes the ECN feedback from congested switches to implement early control at sources. When congestion happens at a switch, represented by the queuing length reaching a specific threshold, the switch will mark the CE bit in the IP header of the arriving packet. At the receiver side, delayed acknowledgement packets (ACKs) are used to convey the sequence of marked packets to senders.

The sender modulates the size of the congestion window according to the extent of congestion. In every RTT, the fraction of the marked packets, represented by α, is updated by $\alpha = (1-g)\times\alpha + g\times F$, where F is the fraction of packets that were marked in the last RTT, and g $(0 < g < 1)$ is the weight given to new samples. This weighted-average metric indicates the extent of congestion, which is used to adjust the window size as follows: cwnd = cwnd $\times(1-\alpha/2)$. By throttling flows as soon as queue length exceeds the threshold, DCTCP reduces queueing delays in congested switch ports, and thus the latencies of delay-sensitive short flows.

Although it analyzed deadline requirements for applications with the Partition-Aggregate structure, DCTCP is a deadline agnostic solution, which does not give priority to flows with deadlines. Therefore, it may cause a large number of flows to miss their deadlines, when there are many concurrent flows with tight deadlines and a serious fan-in congestion.

In comparison, D^{2}TCP is a deadline aware congestion control protocol, which adjusts the size of the congestion window according to the extent of congestion and the urgencies of flow deadlines. The hope is to allow flows with closer deadlines to back off less than flows with deadlines that are more relaxed.

In D^{2}TCP, flow deadlines are specified by applications and passed to the transport layer. The switches are ECN capable as in DCTCP. When a buffer occupancy exceeds a specified threshold, the switch will mark the CE bit in the IP header, to inform the sender of the congestion. Observing the CE feedback, the sender updates the fraction of the marked packets averaged over time, denoted by α, similar to DCTCP. In addition, the deadline imminence factor d is calculated in each RTT as the estimated time for the flow to complete divided by the remaining time before the deadline. Based on both the extent of congestion α and the deadline imminence factor d, each sender modulates its congestion window W as follows:

$$x = \begin{cases} W\times\left(1-\dfrac{p}{2}\right), & p>0; \\ W+1, & p=0 \end{cases} \tag{1}$$

where $p = \alpha^d$, which is a Gamma-correction function shown in Fig. 4.

The Gamma-correction function elegantly combines the extent of congestion and the urgency of deadline, perfectly characterizing the properties that are desired. When the congestion is mild (i.e., α is near 0), the flow with a deadline that is far away (i.e., $d < 1$) will have a large p, shrinking its congestion window more effectively. In contrast, the flow with a close deadline (i.e., $d > 1$) will almost retain its congestion window since it has a small p. In this sense, flows with deadlines that are more relaxed relinquish their bandwidth to allow flows with closer deadlines to meet

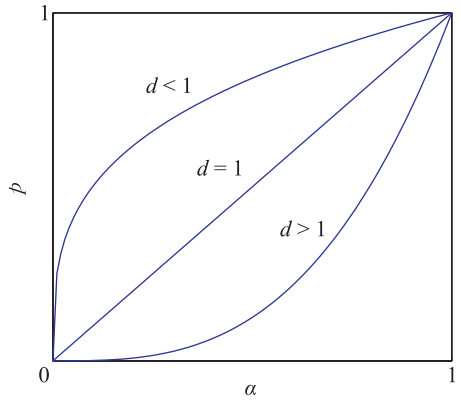


Fig. 4 Gamma-correction function for congestion avoidance (p=α^d).

their deadlines. When the congestion is serious (i.e., α is near 1), however, all of the flows will back off similarly to alleviate the congestion.

To summarize, both DCTCP and D^{2}TCP inherit the distributed and reactive nature of TCP: senders control their congestion windows without knowing the behaviours of other flows, and adjust their sending rates based on congestion feedback to avoid oversubscription of bandwidth in a reactive manner. In terms of practicality, DCTCP and D^{2}TCP show their great advantages, for they only demand ECN capable switches, which are cheap and accessible. Moreover, with the ability to coexist with TCP, both of these protocols can be deployed incrementally on demand, without incurring complete unavailability during an upgrade, and without unreasonably requiring all applications to abandon TCP. Despite many similar features, the essential difference between DCTCP and D^{2}TCP is deadline awareness. By incorporating deadline awareness into the sender-side back-off mechanism, D^{2}TCP substantially outperforms the deadline-agnostic DCTCP, allowing more flows to meet their deadlines.

5 Proactive Switch-Side Rate Allocation: D3 vs. PDQ

Compared with D^{2}TCP where reactive congestion control is distributed among senders, in D3[5] and PDQ[7], switches become the critical controllers that proactively allocate sending rates to flows.

The basic idea of D3 is that each sender calculates and requests its sending rate from routers along its flow path, while routers perform rate allocations, prioritizing the flows with deadlines to satisfy as many of them as possible.

The desired sending rate is defined as the remaining flow size divided by the remaining deadline. In every RTT, the desired rate is updated and piggybacked on one of the data packets to traverse the routers along the flow path. Upon receiving rate requests, each router allocates the rate and piggybacks the allocated rate on the packet header. A vector of allocated rates by each router along the path is fed back to the sender through the ACK from the receiver. The sender chooses the minimum of the allocated rates as the rate to send data packets. In this way, the bandwidth for each flow is reserved before the data transmission in every RTT.

Deadline awareness is incorporated into rate allocations in the routers. The overall idea of rate allocation is to first satisfy the rate requests of flows with deadlines and then allocate the spare capacity equally to all of the current flows. If the router does not have enough capacity, it greedily tries to maximize the number of flows that can meet their deadlines, while assigning a base rate to the remaining flows for them to request rates in the future.

The routers are centralized controllers for bandwidth allocations. Typically, centralized controllers require the maintenance of per-flow states to make decisions, which is expensive and not scalable. However, in the design of D3, the state of each flow is maintained by the sender rather than the router, and is conveyed to routers by piggybacking on a packet header. The flow state is used to require the router to assign the allocated rate for the next RTT and return the allocation for the current RTT. Each router simply keeps aggregate counters for all of the flows passing it. Without maintaining per-flow states, D3 successfully avoids the limitation of a naive centralized design.

The utilization of a router may be impacted by flows bottlenecked downstream and senders failing to return allocations. In addition, queuing may be resulted from a burst of new flows. D3 periodically adjusts the aggregated sending rate for each link to ensure high utilization and low queuing.

Although D3 is aware of deadlines, it is unaware of the urgencies of deadlines. In a first-come-first-served manner, D3 highly depends on the arriving orders of flows with deadlines. When the network is congested, if the flow with a closer deadline arrives slightly after the flow with a deadline that is farther away, the greedy approach will satisfy the flow that arrives earlier, causing the flow with the closer deadline to miss its own. However, there exists a possibility that both of these two flows are able to meet their deadlines, if D3 first satisfies the flow with the closer deadline. Due to the first-come-first-served nature, D3 fails to maximize the number of flows whose deadlines can be satisfied. The priority inversions will cause many flows to miss their deadlines, especially in fan-in bursts.

With respect to its practicality, D3 requires switches to handle requests at line rates, incurring high costs and a long time for deployment. To make matters worse, since the bandwidth allocation of D3 is not recognized by TCP, D3 can not coexist with TCP. Therefore, there will be a long period of unavailability when a datacenter is being upgraded.

Preemptive Distributed Quick (PDQ) flow scheduling is another deadline aware protocol that fixes the priority inversion problem in D3 by enabling flow preemption. Whenever a new flow arrives or an existing flow leaves, link bandwidth will be reallocated to flows in a decreasing order of their criticality. If the flow with a closer deadline is considered more critical than the flow with a deadline that is farther away, PDQ emulates the scheduling discipline of Earliest Deadline First (EDF) to achieve the objective of meeting flow deadlines. Alternatively, if the flow with a shorter expected completion time is considered more critical, PDQ approximates SJF to minimize the mean flow completion time. EDF and SJF are centralized algorithms, which require complete knowledge of flow states and zero communication delay. PDQ avoids the limitations by implementing them in a distributed manner.

Similar to D3, senders in PDQ request their sending rates from switches before sending data packets, which is quite different from D^2TCP where senders reduce their sending rates upon receiving the congestion feedback, in a reactive manner. Along with the desired sending rate, other flow state information is piggybacked on the scheduling header to be sent to the network and viewed by the switches. When receiving an ACK packet, the sender updates its flow state information and sends packets at the rate allocated by the switches along the path.

In contrast to D3, a PDQ sender makes an improvement by terminating a flow if it is doomed to miss its deadline, when any of the following conditions is satisfied: the deadline has passed; the remaining flow transmission time is larger than the remaining time before the deadline; the flow is paused; and the remaining time before the deadline is smaller than an RTT. In this way, PDQ avoids wasting bandwidth on the unproductive flows.

When flows are contending for bandwidth, a PDQ switch accepts flows and allocates bandwidth to flows based on their criticality, allowing critical flows to preempt less critical flows. To compare flow criticality and resolve flow contentions, switches share a common flow comparator and maintain states about flows on each link. Hence, compared with D3, PDQ is expensive and not scalable if the number of concurrent flows increases.

Switches along the path of a flow exchange their separate decisions on flow pause and flow acceptance by tagging the scheduling header, to reach a consensus on the global decision. A flow will be paused if any switch along the path pauses it, while it will be accepted only if all switches along the path accept it. The coordination between switches in PDQ is a major difference from D3. Since the actual flow sending rate in D3 is set as the minimum of allocations by all of the switches, the network resource is not efficiently utilized. In contrast, the sender in PDQ sends data at the rate agreed by all switches, hence making better use of the bandwidth.

Along with the benefits of the preemption is the challenge of low link utilization when switching between flows. When a flow finishes, the link will be idle for one or two RTTs before the next flow receives the feedback and starts. To address the problem, PDQ starts the next set of flows slightly before current flows finish. Nevertheless, this will generate more concurrent flows temporarily, resulting in increased queues. Similar to D3, PDQ controls the per-link aggregated sending rate to drain the queues after flow switching. Further, the rate control allows PDQ to be friendly to other transport protocols, while D3 can not coexist with TCP. However, its coexistence with other protocols requires static bandwidth partitioning between PDQ and non-PDQ flows, which may result in a waste of bandwidth.

Furthermore, there is a multi-path version of PDQ, designed to increase the transmission reliability and explore the path diversity. The basic idea is that a sender splits a PDQ flow into sub-flows with the equal size. The switches are kept the same as in standard PDQ. The only difference is that a sub-flow becomes the scheduling unit, and there may be paused sub-flows and sending sub-flows at the same time. A sender periodically shifts the load from the paused sub-flows to the sending one with the minimum remaining load. Due to the flow split, there may be packets arriving out of order. To solve this problem, per-flow buffers are maintained at the receiver side to reorder the packets.

6 Exploiting Path Diversity: DeTail

According to the analyses in DeTail[2], the challenge with the objective of creating and rendering a page within a specific deadline is the high variance of flow completion times, which form a long-tailed distribution. With the long tail, Facebook engineers are only provided with two poor options. They may set tight timeouts for resending requests to increase the chance

of rendering a complete page, at the cost of increasing the server load. Or they may use conservative timeouts to avoid unnecessary request retransmissions, at the risk of rendering an incomplete page at the deadline. Neither of these options are satisfactory. However, if the long tail is reduced, a better result can be achieved that applications can use tighter timeouts to render a complete page without increasing the server load.

This observation motivates the design of DeTail to meet deadlines from the perspective of reducing the long tail of flow completion times. The long tail is caused by timeouts after packet losses, uneven load balancing, and the absence of traffic prioritization. To reduce the long tail, DeTail constructs a lossless fabric to prevent congestion-related packet losses, and quickly detects congestion at lower layers, to drive upper layer routing decisions in order to find alternative paths with lower congestion.

Switches in DeTail employ the Combined Input/ Output Queue (CIOQ)[38,39] architecture, consisting of both ingress queues and egress queues. These queues perform strict priority queueing among eight different priorities. Each queue maintains the counters of per-priority occupancies. Whenever a counter exceeds a specific threshold, the switch will quickly notify the higher layers of congestion.

At the link layer, DeTail employs Priority Flow Control (PFC)[40], a hop-by-hop push-back mechanism (available on newer Ethernet switches) that reacts to congestion by sending control messages to the previous hop. Congestion is indicated by the ingress queue occupancy. When the occupancy exceeds a threshold, the switch sends a Pause message, informing the previous hop to stop transmitting packets of the specific priority. When the queue length reduces, the switch informs the previous hop to resume packet transmissions through sending an Unpause message. In a serious congestion, with the queues of previous hops building up, Pause messages will be generated by each hop along the path back to the source end host and finally quench the source.

With the indication of congestion provided by the link layer, DeTail performs a per-packet congestion-based load balancing at the network layer, forwarding packets to the shortest path that is the least congested. The switch monitors egress queue occupancies which indicate the congestion downstream. As shown in the link layer, the egress queue will build up after receiving the Pause message generated by the next hop where congestion occurs. To react to the congestion, the switch forwards the arriving packet to the least congested shortest path based on a bitmap of favored ports (F) which are lightly loaded and a bitmap of acceptable ports (A) that lead to shortest paths. As illustrated in Fig. 5, an arriving packet obtains the associated bitmap of A from the IP lookup process, as well as the bitmap of F based on its priority. Then a bitwise AND (&) of these two bitmaps is performed, to obtain a set of usable ports ($U = A\&F$), which are lightly loaded ports leading to the shortest paths. DeTail randomly chooses a port from U and forwards the packet to it.

The transport layer protocol in DeTail needs to be redesigned since the switching fabric is load-balanced and lossless. Due to load balancing, packets arriving out of order will be the normal case. Thus, the transport protocol should be robust to packet reordering. As the fabric is lossless, packets are seldom dropped. Hence, the traditional fast recovery and fast retransmission based on packet drop should be abandoned. Since congestion can no longer be indicated by packet drops, DeTail monitors output queues and uses ECN to indicate congestion, similar to DCTCP and D^{2}TCP. If the buffer occupancy of a low priority queue exceeds a specific threshold, the ECN flags of the packets will be marked. By this way, the low priority flows without deadlines will back off to alleviate the congestion.

DeTail fails to address fan-in congestion. The bottleneck in fan-in congestion is the egress port of the Top-of-Rack switch connected to the root node. Since there are no alternate paths, the fan-in congestion can not be handled by exploiting the path diversity. Another limitation of DeTail is that the number of priority levels (8-16) supported by hardware (with respect to PFC) is far from enough, unable to satisfy the demands of flows with various deadlines.

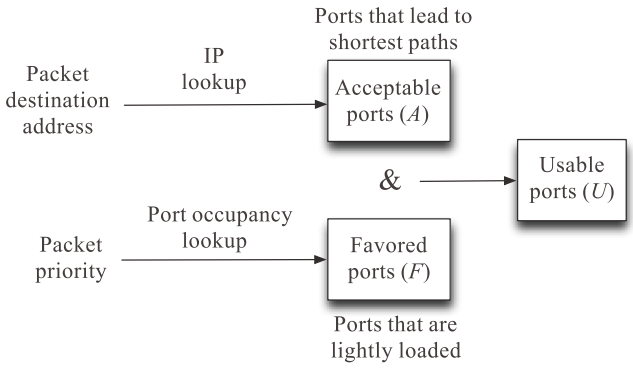


Fig. 5 Adaptive load balancing in DeTail.

7 Priority Dropping: pFabric

pFabric[8] is a datacenter fabric design that significantly simplifies buffering and congestion control. It performs greedy packet scheduling and dropping at switches, according to the priorities of packets, indicated in packet headers. The key design insight of pFabric is the decoupling of flow prioritization from rate control, which is quite different from previous works.

Traditionally, flow prioritization is coupled with the sending rate. In DCTCP and D^{2}TCP, flow prioritization is indicated by the extent of window size reduction. In D3 and PDQ, a flow with a higher priority will be assigned a higher sending rate than others. This kind of flow prioritization requires a cross-layer cooperation, which is rather complex. Decoupling flow prioritization from rate control, pFabric designs a simple datacenter fabric. It views the entire fabric as a giant switch and employs greedy scheduling across the fabric, significantly simplifying rate control.

According to the design, the priority of a flow is specified by the sender, based on the deadline and the size of the flow. In the packet header, there is a field, called the priority number, that represents the flow priority. Switches accept and schedule packets according to their priority numbers. When a buffer is full, if the incoming packet has the lowest priority among all of the buffered packets, it will be dropped. Otherwise, it will be accepted to replace the packet that has the lowest priority in the buffer. When a port is idle, the packet with the highest priority is always the first to be transmitted. In this sense, the scheduling is preemptive, since a flow arriving later may be scheduled ahead of the earlier ones, if its priority is the highest among concurrent flows. The rate control required by pFabric can be easily implemented, since it only supplies the basic function of throttling flows when the network is approaching the congestion collapse.

pFabric significantly simplifies the design by eliminating the need for complex congestion control at the senders, feedback generation from the network, flow state maintenance, high speed buffers, and rate assignments at switches. Of course, there is a performance tradeoff in pursuing such simplicity. Further, pFabric has not yet been evaluated in a testbed with real traffic.

8 Concluding Remarks

In retrospect, the design objective of network protocols in datacenter networks has evolved form maximizing network throughput to meeting flow deadlines. The rise of research interests in deadline aware protocols for datacenter networks is largely fuelled by the performance requirements of cloud applications with the Partition-Aggregate structure, as well as the characteristics of datacenter networks hosting these applications.

In this context, we have surveyed a number of new protocols with deadline awareness, and summarized key mechanisms employed in traditional network protocols, illustrating how recently proposed protocols are able to incorporate deadline awareness into these mechanisms.

Among the deadline aware protocols, D^{2}TCP[6] applies reactive congestion control at senders to control their sending rates. Compared with the original TCP which uses packet drop as congestion feedback, D^{2}TCP applies the piggybacked ECN technique to inform senders of congestion. Moreover, deadline awareness in D^{2}TCP is represented by flows with different deadlines backing off to different extents. Two alternative deadline aware protocols, D3[5] and PDQ[7], avoid congestion by proactively specifying appropriate flow sending rates. They incorporate deadline awareness into rate allocations at switches, which make allocation decisions either independently[5] or cooperatively[7], and convey the allocations to senders. DeTail[2] utilizes flow path diversity, i.e., selecting a less congested path, to reduce the long tail and increase the chance for a flow to meet its deadline. From the perspective of packet scheduling, pFabric[8] makes packet drop and packet transmission deadline aware. A comparison among these protocols is shown in Table 1.

In general, the design of network protocols in datacenter networks is mainly constrained by hardware requirements and backward compatibility. In this sense, D^{2}TCP outperforms the other protocols with its simplicity, backward compatibility, and ready deployability. However, Software Defined Networking (SDN)[41] has recently been proposed to build a "clean slate" network architecture. In such a new architecture, we expect that the factors of hardware and compatibility would no longer be design constraints. In addition, with a global network view provided by SDN, control decisions can be made by a centralized controller with more accuracy. Recently, Ghobadi et al.[42]

Table 1 Deadline aware protocols in datacenter networks: A comparison.

Work	Principles of meeting deadlines	Mechanisms and features	Switches	End hosts
D3	Greedily satisfy deadline flows as many as possible	Proactive rate control, request and set	Determine allocations independently with FCFS discipline	Specify flow deadlines, request and set sending rates
D^2TCP	Flows with tighter deadlines back off less when congestion occurs	Reactive congestion control, try and back off	Monitor Buffer Occupancy and provide ECN feedback	Specify flow deadlines, calculate and modulate sending rates
PDQ	Flow with the earliest deadline preempts other flows	Proactive rate control, request and set	Determine allocations cooperatively with EDF or SJF discipline	Specify flow deadlines, request and set sending rates
DeTail	Prevent packet losses and select less congested paths	Multi-path load balancing	Monitor Buffer Occupancy and quench source, decide the suitable path	Specify flow priorities
pFabric	Schedule and drop packets according to priorities	Simple priority dropping	Schedule and drop packets according to priorities independently	Specify flow priorities

have taken the initiative to propose a TCP adaptation framework in SDN. They tuned transport protocols according to the network and traffic conditions viewed by the SDN controller. Facilitated by SDN, centralized network protocols can be designed and implemented to optimize the performance of applications with deadline requirements with finer granularity. We look forward to a variety of more effective solutions in the near future.

References

[1] T. Hoff, Latency is everywhere and it costs you sales — How to crush it, http://highscalability.com/blog/2009/7/25/latency-is-everywhere-and-it-costs-you-sales-how-to-crush-it.html, 2012.

[2] D. Zats, T. Das, P. Mohan, D. Borthakur, and R. Katz, DeTail: Reducing the flow completion time tail in datacenter networks, in *Proc. ACM SIGCOMM*, 2012.

[3] N. Dukkipati and N. McKeown, Why flow-completion time is the right metric for congestion control, *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 1, pp. 59-62, Jan. 2006.

[4] H. Wu, Z. Feng, C. Guo, and Y. Zhang, ICTCP: Incast congestion control for tcp in data center networks, in *Proc. CoNEXT*, 2010.

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

[6] B. Vamanan, J. Hasan, and T. N. Vijaykumar, Deadline-aware datacenter TCP (D^2TCP), in *Proc. ACM SIGCOMM*, 2012.

[7] C. Hong, M. Caesar, and P. Godfrey, Finishing flows quickly with preemptive scheduling, in *Proc. ACM SIGCOMM*, 2012.

[8] M. Alizadeh, S. Yang, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, Deconstructing datacenter packet transport, in *Proc. ACM SIGCOMM HotNets Workshop*, 2012.

[9] L. Xu, K. Harfoush, and I. Rhee, Binary increase congestion control (BIC) for fast long-distance networks, in *Proc. IEEE INFOCOM*, 2004.

[10] S. Ha, I. Rhee, and L. Xu, CUBIC: A new tcp-friendly high-speed TCP variant, *ACM SIGOPS Operating Systems Review*, vol. 42, no. 5, pp. 64-74, 2008.

[11] D. Wei, C. Jin, S. Low, and S. Hegde, FAST TCP: Motivation, architecture, algorithms, performance, *IEEE/ACM Trans. on Networking*, vol. 14, no. 6, pp. 1246-1259, 2006.

[12] S. Floyd, HighSpeed TCP for large congestion windows, RFC 3649, 2003.

[13] S. Bhandarkar, S. Jain, and A. Reddy, LTCP: Improving the performance of TCP in highspeed networks, *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 1, pp. 41-50, 2006.

[14] D. Katabi, M. Handley, and C. Rohrs, Congestion control for high bandwidth-delay product networks, in *Proc. ACM SIGCOMM*, 2002.

[15] V. Vasudevan, A. Phanishayee, H. Shah, E. Krevat, D. Andersen, G. Ganger, G. A. Gibson, and B. Mueller, Safe and effective fine-grained TCP retransmissions for datacenter communication, in *Proc. ACM SIGCOMM*, 2009.

[16] Y. Chen, R. Griffith, J. Liu, R. Katz, and A. Joseph, Understanding TCP incast throughput collapse in datacenter networks, in *Proc. ACM Workshop on Research on Enterprise Networking*, 2009.

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

[18] 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 Networked Systems Design and Implementation (NSDI)*, 2012.

[19] R. Braden, D. Clark, and S. Shenker, Integrated services in the internet architecture: An overview, RFC 1633, 1994.

[20] K. Nichols, S. Blake, F. Baker, and D. Black, Definition of the differentiated services field (DS field) in the IPv4 and IPv6 Headers, RFC 2475, 1998.

[21] N. Bansal and M. Harchol-Balter, Analysis of SRPT scheduling: Investigating unfairness, in *Proc. ACM SIGMETRICS*, 2001.

[22] V. Sivaraman, F. M. Chiussi, and M. Gerla, End-to-end statistical delay service under GPS and EDF scheduling: A comparison study, in *Proc. IEEE INFOCOM*, 2001.

[23] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, Hedera: Dynamic flow scheduling for data center networks, in *Proc. USENIX Networked Systems Design and Implementation (NSDI)*, 2010.

[24] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley, Improving datacenter performance and robustness with multipath TCP, in *Proc. ACM SIGCOMM*, 2011.

[25] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley, Design, implementation and evaluation of congestion control for multipath TCP, in *Proc. USENIX Networked Systems Design and Implementation (NSDI)*, 2011.

[26] S. Floyd and V. Jacobson, Random early detection gateways for congestion avoidance, *IEEE/ACM Trans. Networking*, vol. 1, no. 4, pp. 397-413, 1993.

[27] C. Hollot, V. Misra, D. Towsley, and W. Gong, On designing improved controllers for AQM routers supporting TCP flows, in *Proc. IEEE INFOCOM*, 2001.

[28] Y. Gu, D. Towsley, C. Hollot, and H. Zhang, Congestion control for small buffer high speed networks, in *Proc. IEEE INFOCOM*, 2007.

[29] Y. Xia, L. Subramanian, I. Stoica, and S. Kalyanaraman, One more bit is enough, in *Proc. ACM SIGCOMM*, 2005.

[30] I. Qazi, T. Znati, and L. Andrew, Congestion control using efficient explicit feedback, in *Proc. IEEE INFOCOM*, 2009.

[31] K. Ramakrishnan and R. Jain, A binary feedback scheme for congestion avoidance in computer networks, *ACM Trans. Computer Systems*, vol. 8, no. 2, pp. 158-181, 1990.

[32] L. Brakmo and L. Peterson, TCP vegas: End to end congestion avoidance on a global internet, *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 8, pp. 1465-1480, 1995.

[33] K. Tan, J. Song, Q. Zhang, M. Sridharan, A compound TCP approach for high-speed and long distance networks, in *Proc. IEEE INFOCOM*, 2006.

[34] D. Ferrari, A. Banerjea, and H. Zhang, Network support for multimedia a discussion of the tenet approach, *Computer Networks and ISDN Systems*, vol. 26, no. 10, pp. 1267-1280, 1994.

[35] S. Liang and D. Cheriton, TCP-RTM: Using TCP for real time multimedia applications, in *Proc. IEEE ICNP*, 2002.

[36] C. Zhang and V. Tsaoussidis, TCP-real: Improving real-time capabilities of TCP over heterogeneous networks, in *Proc. Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, 2001.

[37] K. Tan, J. Song, Q. Zhang, and M. Sridharan, A compound TCP approach for high-speed and long distance networks, in *Proc. IEEE INFOCOM*, 2006.

[38] Cisco Nexus 5000 Series Architecture, http://www.cisco.com/en/US/prod/collateral/switches/ps9441/ps9670/white_paper_c11-462176.html, 2012.

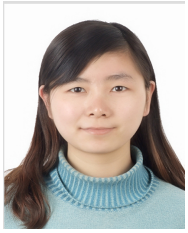
[39] N. McKeown, White paper: A fast switched backplane for a gigabit switched router, http://www.cs.cmu.edu/ srini/15-744/readings/McK97.pdf, 2012.

[40] Priority flow control: Build reliable Layer 2 infrastructure, http://www.cisco.com/en/US/prod/collateral/switches/ps9441/ps9670/white_paper_c11-542809.pdf, 2012.

[41] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, OpenFlow: Enabling innovation in campus networks, *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69-74, 2008.

[42] M. Ghobadi, S. Yeganeh, and Y. Ganjali, Rethinking end-to-end congestion control in software-defined networks, in *Proc. ACM HotNets Workshop*, 2012.

Baochun Li received his PhD degree from the Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, in 2000. Since then, he has been with the Department of Electrical and Computer Engineering at the University of Toronto, where he is currently a professor. He holds the Bell Canada Endowed Chair in Computer Engineering since August 2005. His research interests include large-scale distributed systems, cloud computing, peer-to-peer networks, applications of network coding, and wireless networks. He is a member of the ACM and a senior member of the IEEE.



Li Chen is currently pursuing her Master's of Applied Science degree at the Department of Electrical and Computer Engineering, University of Toronto. She received her BEng degree from the Department of Computer Science and Technology, Huazhong University of Science and Technology, China, in 2012. Her research interests include cloud computing and datacenter networking.

Bo Li is a professor in the Department of Computer Science and Engineering, Hong Kong University of Science and Technology. He received his PhD degree from the Department of Electrical and Computer Engineering, University of Massachusetts at Amherst. He holds the Cheung Kong Chair Professor in Shanghai Jiao Tong University. Prior to that, he was with IBM Networking System Division, Research Triangle Park, North Carolina. He was an adjunct researcher at Microsoft Research Asia-MSRA and was a visiting scientist at Microsoft Advanced Technology Center (ATC). He has been a technical advisor for ChinaCache Corp. (NASDAQ CCIH) since 2007. He is an adjunct professor in Huazhong University of Science and Technology, Wuhan, China. His recent research interests include large-scale content distribution in the Internet, peer-to-peer media streaming, the Internet topology, cloud computing, and green computing and communications. He is a fellow of IEEE for "contribution to content distributions via the Internet". He received the Young Investigator Award from the National Natural Science Foundation of China (NSFC) in 2004. He served as a distinguished lecturer for IEEE Communications Society (2006-2007). He was a co-recipient for three Best Paper Awards from IEEE, and the Best System Track Paper in ACM Multimedia (2009).