Preventing Passive TCP Timeouts in Data Center Networks with Packet Drop Notification

Yu Xia, Ting Wang, Zhiyang Su and Mounir Hamdi
Department of Computer Science and Engineering
Hong Kong University of Science and Technology
Hong Kong, China
emails: rainsia@gmail.com, {twangah, zsuab, hamdi}@cse.ust.hk

***Abstract*—Data centers are considered as the indispensable infrastructure for cloud services nowadays. However, large-scale data centers using the traditional TCP protocol may experience some performance issues, such as TCP Incast and long query completion times. It is shown that the fundamental cause of these problems is that the existing TCP protocols are unable to effectively detect packet losses in the network, which always causes timeouts in data centers. In this paper, we present the packet drop notification, a mechanism that can accurately inform senders about packet losses. So the passive timeouts can be avoided, and the packets can be retransmitted immediately. This mechanism is simplified when used with the software-defined networking technique, which is a promising technology to build a more customizable control plane, in data center networks. Simulation results show that the precise packet drop notification can improve the goodput of high fan-in applications, and therefore effectively reduce the task completion times. Further, we evaluate the fairness of the TCP with notification, and show that it can work well with other TCP variants.**

***Keywords*-TCP Incast; retransmission timeout; data center; packet loss; source notification**

I. Introduction

Data centers have become a key component of today's cloud computing and various other services that depend on intensive server-side computing. For example, most of the mobile applications need server-side computing because of the limitation of the computing and storage resources on the mobile devices. As an increasing number of applications moving to the cloud, the requirement of modern large-scale data centers is much higher than traditional data centers, e.g., the shorter round-trip time (RTT), higher bandwidth, highly variable flow characteristics, and very low latency [1].

The TCP/IP protocol stack, as the de facto standard for computer communications, is widely used in Internet and proved its success. The TCP protocol, which is one of the key protocols in the Internet stack, provides reliable data transfer services. TCP has proven to be very successful in Internet, and is deployed in almost all the data centers; however, traditional TCP is imperfect in data center networks (DCNs). Contrary to Internet, the end-hosts are near to each other in a data center. Besides, the data rates of the DCNs are high. As a result, the round-trip time (RTT) is very short, around several hundreds microseconds [2]. On the other hand, the commodity switches used in DCNs have small buffers. The purpose of using small buffers is to lower the queuing delay as well as the cost; however, when a switch is heavily loaded, packet losses are usually unavoidable. TCP's packet loss detection mechanism is found to be inefficient and leads to severe link under-utilization, and thus the goodput collapse [2–4] for Incast applications, which use high fan-in, many-to-one traffic patterns, such as cluster-based storage systems [5] and MapReduce [6] applications.

In this paper, we present a simple and practical solution to the TCP Incast problem in DCNs. As the TCP Incast problem is caused by the packet loss as well as the slow detection of the packet loss in the traditional TCP protocol, we propose to use explicit packet drop notifications (PDNs) to allow the accurate notification of the lost packets and fast retransmission of them. In the solution, switches extract the essential information, called packet drop notification data (PDND), from the packets before discarding them and store the PDNs locally. When a frame moving to the next hop towards the original sender passes by, the switch then attaches the PDN to the frame. The PDN is then carried along the path to the original sender of the lost packet. When the sender receives the notification, it gets the precise information of which packet is lost. The sender then retransmits the packet immediately. The solution needs slight modifications to switches. However, as the increasing popularity and the continuous development of the software-defined networking (SDN) technology, it will be possible to customize the actions that a switch can take in a programmable way. Thus, the solution is practical in the near future and gives the device vendors a guideline to the useful actions of the SDN-enabled switches. Finally, we show the effectiveness of our solution and its compatibility with the traditional TCP protocol through extensive simulations.

The rest of this paper is organized as follows. Section II describes the necessary background of the paper, such as the data center networks, the TCP Incast problem and the TCP timeout, as well as the software-defined networking. Section III briefly summarizes the related work on the TCP Incast problem and methods to prevent the timeout. Section IV introduces our solution of using the explicit packet drop notification to notify the senders about the packet losses to prevent the passive

This research is partially supported by HKUST Research Grants Council (RGC) under the grant number 613113.

978-1-4799-2730-2/14/$31.00 ©2014 IEEE

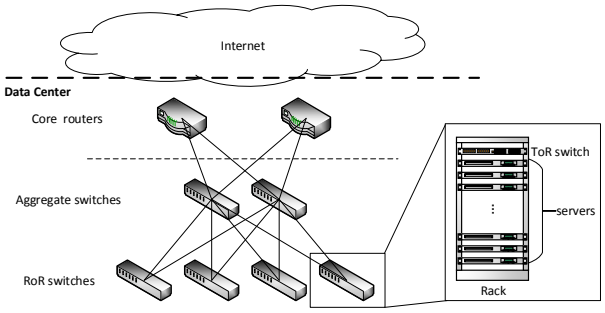

Fig. 1. A common data center architecture and the detailed rack

TCP timeouts. In Section V, we evaluate the performance of the proposed solution through extensive simulations in NS-3. Section VI draws conclusions.

II. Background

A. Data center networks

Data centers are centralized repositories of the computation, storage, management, dissemination of information and data, and many other resources. A typical data center usually consists of computers, switches/routers, racks of servers, e.g., Web servers, application servers and database servers, etc. All these infrastructures within a data center are orchestrated by the data center network (DCN) to work as an organic cohesive whole. The network architecture in a data center greatly affects its performance.

Fig. 1 shows a typical DCN architecture. Servers are usually grouped into racks. The servers in the same rack are connected through a switch, called the top-of-rack (ToR) switch. In a typical setup, the number of servers in a single rack is around 40 to 48, as the most available commodity switch usually has 48 Gigabit ports and one more more 10-Gigabit ports. The ToR switches from racks are connected by higher level switches/routers. There are usually two additional levels of switches/routers on top of the ToR switches, namely the aggregate switches and core switches/routers.

The traditional data center has over-subscription on different levels, and as traffic moves up to the higher levels, the over-subscription ratio increases rapidly. Servers under the same ToR switch always has 1:1 over-subscription, i.e., they can communicate with each other at the full rate (e.g., 1 Gbps). However, from the ToR switches to the aggregate switches, the over-subscription is increased to 1:5 to 1:20, i.e., 20 severs have to share 1 to 4 Gbps of the uplink bandwidth. From the aggregate switches to core switches/routeres, the over-subscription can be as high as 1:240, which largely limits the performance of the whole data center. Architectures, such as Portland [7] and VL2 [8], are able to provide full bandwidth for all the servers with more aggregation and core switches/routers, but they pose high wiring complexity on the DCNs.

B. TCP Incast and timeout

In a typical TCP Incast scenario, a host, called the client, requests for a chunk of data, usually called the server request unit (SRU), from several other hosts, called the servers. After these servers get the data ready, they send the data to the client almost in parallel, as shown in Fig. 2. In most of the cases, only when the client successfully received all the SRUs from all the servers, can it send out new requests for another round of SRUs. This traffic pattern is common in storage networks and the MapReduce applications in DCNs. Modern data center networks usually have high bandwidth and low latency. For such high fan-in traffic patterns in data centers, the small buffer space on the commodity switches becomes the critical resource. When a large number of flows destined for the same output port arrive at a switch, the buffer will soon overflow and severe packet losses occur. For the flows, if there are not enough subsequent packets to trigger three duplicate ACKs at the sender (this could happen at the end of the TCP transmission window), retransmission timeout (RTO) is then triggered at one or more TCP senders. However, the minimum retransmission timeout (RTO_{min}) is usually limited to 200 ms on most of the OSs. However, in the data center networks, the typical round-trip time (RTT) is in the order of 100 μs [2]. Therefore, if a packet loss occurs and the RTO has to be triggered, there will be a 200 ms idle time for that pair of sender and receiver, and the goodput of the application will collapse.

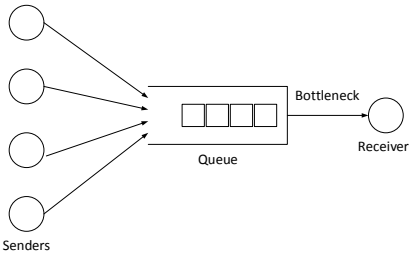

Fig. 2. Multiple sender send data to a single receiver concurrently

C. Software-defined networking

Software-defined networking (SDN) is an emerging technology which gains significant attention from both academia and industry. The decoupling of the control plane and data plane enables unprecedented network control flexibility and programmability. A logically centralized controller connects to all switches via secure channels and maintains the network states across the network. When a new flow (an unknown packet) comes to the network, a table miss is generated and , according to the configuration, the packet header or even the whole packet will be forwarded to the controller to decide for the specific actions for the switch (e.g., forward, drop, modify the header, etc.). Forwarding rule is usually installed along the path bidirectionally to save the communication cost between the controller and switches [9]. SDN allows network operators to easily deploy innovative protocols and algorithms in a closed network. SDN is currently being supported by major network device vendors, and has become an open standard that enables network operators to easily deploy existing and experimental innovative protocols in real networks. Besides, traffic engineering in SDN is much easier as different granularity of flow statistics can be collected from the switches on demand. OpenFlow [10] is the first and the actual standard of

SDN. It has been deployed in commercial data centers [11] and experimental data centers [12, 13]. SDN was originally designed to allow easy innovations in networks. Although only the control plane is programmable at the current stage, a more customizable data plane will later be possible, as the further development of the SDN technology, to allow easy and cheap innovations in the data plane.

III. Related Works

The researchers have tried to solve the TCP Incast problem from different network layers.

Efforts have been devoted to solve the problem on the data link layer. Reference [3] proposed to use Ethernet flow control (EFC) to solve the TCP Incast problem. However, it suffers from the head-of-line blocking problem. Quantized Congestion Notification (QCN) [14], is a congestion control mechanism designed for data center Ethernet. However, QCN itself cannot completely solve the TCP Incast problem. Thus, in [15], the authors proposed to modify QCN by increasing the sampling frequency at the Congestion Point (CP) and to make the link rate increase adaptively to the number of flows at the Reaction Point (RP). However, QCN-based protocols have to monitor the packet arrival rate at the flow level, which incurs high overhead given the large number of flows in modern data centers.

Some researchers tried to solve the problem from the source, i.e., the transport layer, by modifying the TCP protocol. Reference [2] proposed to reduce the RTO_{min} to match the RTT, e.g., 200 μs, in the DCNs. However, it is impossible to implement such fine-grained timers in most of the OSs. If one of the servers does not support such a fine-grained timer, the whole Incast group will be waiting for it, which still causes goodput collapse. ICTCP [16], in a cross-sockets manner, controls the transmission rates of the senders by adaptively adjusting the advertising window (aWnd) at the receiver side. The receiver estimates the available bandwidth and RTT to calculate an appropriate aWnd value for each flow to allow them fairly inject proper number of packets to the network. However, the estimation of the real-time available bandwidth and RTT is difficult, and even harder if they need to be accurate. Furthermore, if the bottleneck link is not directly connected to the receiver, e.g., the scenario shown in Fig. 5, ICTCP cannot work properly. DCTCP [1] employs the explicit congestion notication (ECN) technology and proportionally reduces the congestion window (cWnd) at the senders to keep the buffer occupancy at the switches low and to avoid packet losses, and therefore the number of timeouts. However, it cannot completely solve the packet loss problem, especially when the number of senders is large. D3 [17] allows the senders of delay-sensitive flows compute their expected sending rates and pass these rates to switches. Each switch then assigns a proper rate for each flow based on the collected rate requirements to avoid traffic congestion and thus solve the TCP Incast problem. However, it requires the switches to keep tracking of the current rate for each existing flow and to calculate a proper rate for each new flow, which is not practical.

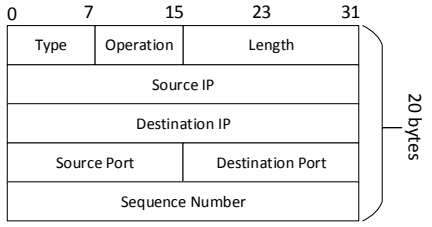

Fig. 3. The format of the packet drop notification

Other researchers tried to solve the problem on the application layer. For example, reference [18] proposed to request only a subset of servers or data each time to limit the number of parallel transmitting flows. However, it is hard to decide how many servers and data to request, especially when there is background traffic besides the Incast traffic.

IV. Explicit packet drop notifications

To prevent the aforementioned TCP problems, we propose to use the explicit, accurate packet drop notification (PDN) to inform the sender to retransmit the lost packets to avoid the passive retransmission timeouts.

When a switch drops a packet due to the congestion, the switch extracts the 5 essential fields from the packet, such as source and destination IP addresses and port numbers, sequence number as well as the length of the payload. These are the most essential information needed for a sender to recover the lost data. The extraction should be easy and fast since the offsets of these fields are fixed in all the packets. The switch then sends the 20-byte PDN to inform the sender to retransmit the packet. The format of the PDN is shown in Fig. 3, where there are two extra 1-byte fields: type and operation. The type field indicates the types of the notification. In this paper, we only discuss the packet drop notification. But other notifications such as accurate congestion notification will be very helpful. The operation field indicates the receiver of the notification what action or actions it should take. In this paper, we ask the sender to retransmit the packet and then reduce the cWnd size by half. Other actions and cWnd reduction could be easily customized especially with the support of the SDN technology.

The switch should reserve some buffer space to store these notifications, and finally forward them towards the senders. The switch will know the next hop switch to forward these notifications according to the essential information they contain. The notifications are queued towards the output ports as the time they are received. The notification queues are separate from packet queues. For an SDN-based data center, the routing of the notifications is simpler. When the connection of a TCP flow was set up by the controller, each switch are also required to set up a reverse path to route the notifications to the sender. As soon as a switch receives a notification, it matches the source and destination addresses and port numbers in the notification with a flow table entry, then it finds out which port to forward the notification and enqueues notification to the notification queue accordingly.

When a notification is enqueued into a notification queue, it waits until it becomes a head-of-line (HoL) notification. At that

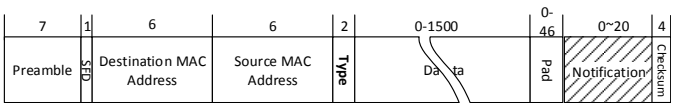

Fig. 4. A notification is added at the end of the frame as the attachment

time, if the output has frames to send, then the switch attaches the notification at the end of the next frame as shown in Fig. 4. This can help reduce the overhead incurred by the notifications. If the packet queue for the output port is empty, which means the link will be idle, then the switch creates an empty frame and attaches the notification to the new frame. To tell a frame whether it carries a notification or not, we use the type field in the Ethernet frame header. Usually, when the frame carries an IPv4 packet, the type field should be 0x0800, and 0x86DD for an IPv6 packet. We assign two unused values, e.g., 0xA800 and 0xA6DD, for the IPv4 and IPv6 packet with a notification at the end, respectively. If the frame is fully loaded, then the notifications cannot be attached, therefore, we should set the maximum segment size (MSS) 20 bytes smaller than the normal value in the OSs to leave some space for the notifications.

When a switch receives a frame, it then checks the type field. If the value is 0x0800 or 0x86DD, then the frame contains a normal IPv4 or IPv6 packet, and they do not need any additional processing. If the value is 0xA800 or 0xA6DD, then there is a notification at the end of the frame. The switch then detaches the notification, and match it with an entry in the flow table to find out the output port. Finally, the switch enqueue the notification to the corresponding notification queue. Then the above process is repeated at every hop, until the notification arrives at the sender of the original packet.

When the sender receives the notification, the protocol stack knows that there is a packet loss in the network. A TCP socket can be determined based on the source and destination address and port fields carried in the notification. The protocol stack then forwards the sequence number and the length of the packet up to the corresponding TCP socket. The TCP socket then retransmits the packet from the TCP transmitting buffer.

Usually, traditional TCP variants detect the packet loss by three duplicate ACKs (totally four ACKs with the same acknowledgment number). After detecting the packet loss, the TCP will reduce the congestion window (cWnd) and also the slow start threshold (ssThresh) accordingly. More specifically, for the TCP New Reno variant, if three duplicate ACKs are triggered, New Reno will halve cWnd, set ssThresh to the halved cWnd, perform a fast retransmit, and enter a phase called fast recovery. When New Reno enters the fast recovery phase, it records the highest outstanding unacknowledged sequence number. When this sequence number is acknowledged, New Reno returns to the congestion avoidance state; or if there is a timeout, New Reno will set cWnd to 1 maximum segment size (MSS) and enters the slow start phase. Similarly, for our TCP with notifications, when a PDN is received, the socket realizes that the network is experiencing congestion, then socket halve the cWnd, set ssThresh, and enters a phase similar to the fast recovery phase. However, in order not to reduce the cWnd too much when a continuous packet drop happens, we only halve the cWnd once in each RTT. Different from the traditional TCP, as soon as the packet losses occur, the sender will receive the PDN, and the sending rate will reduce immediately. Thus, the packet loss rate can be well controlled, which ensures that only a small notification queue is needed. This is also validated through simulations in Section V.

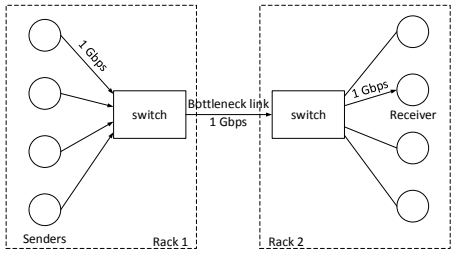

Fig. 5. The topology of the simulation

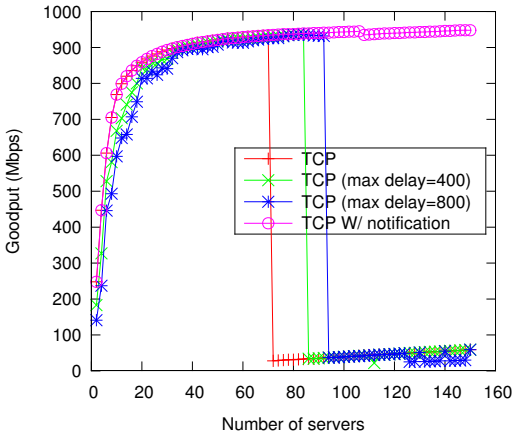

Fig. 6. The goodput of various protocols

V. Performance Evaluation

In this section, we evaluate the performance of TCP with notifications through simulations by using the network simulation 3 (NS-3). The topology used for the simulations is shown in Fig. 5, where we assume that the servers in one rack is communicating with a server in another rack. Each rack has a switch connecting all the servers. The two switches are connected directly through a link to eliminate the interference of routing algorithms in the network, so we can concentrate on the transport protocol itself. All the links are 1 Gbps and the propagation round-trip delay of the network is set to 100 µs. Commercial commodity switches used in data center networks, such as the Cisco Catalyst 4948, usually has at least 16 MB of buffer memory for its 48 ports; in other words, each port has a buffer of about 300 KB. Thus, we set the buffer size of each port in the switches to 300 KB. The minimum retransmit timeout (RTO_{min}) is set to 200 ms, which is the default value in most of the OSs.

In the first simulation, we evaluate the application level goodput of different protocols: TCP New Reno, TCP New Reno with deliberate random starting delays, and our TCP with PDN. We set the SRU to 10 KB and vary the number of servers

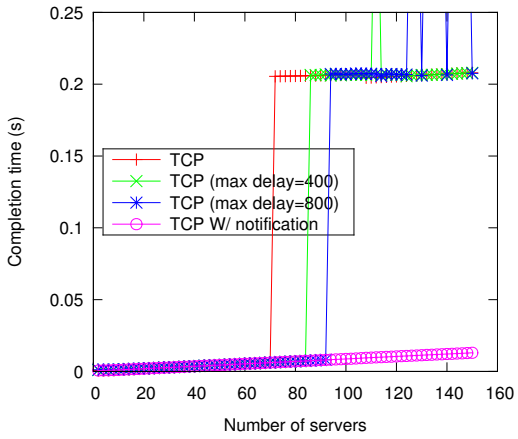

Fig. 7. The work completion time of various protocols

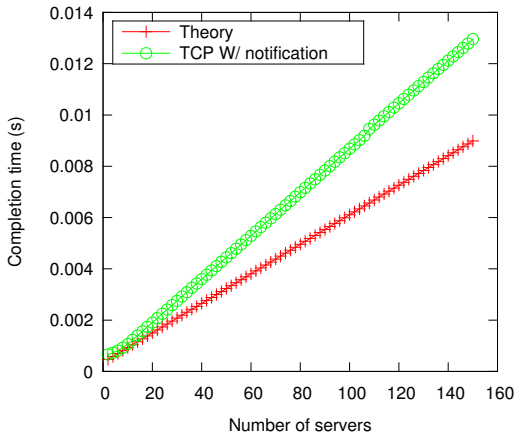

Fig. 8. The comparison of TCP/notification with the best-case theoretical completion time

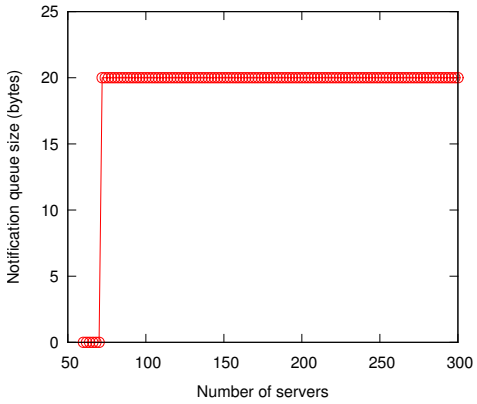

Fig. 9. The maximum size of notification queues in the bottleneck switch

from 2 to 150. The goodput is measured from the application layer of the receiver, which only counts the effective data excluding the headers and duplicated packets. As we can see from Fig. 6, the goodput of the standard TCP protocol collapses when the number of servers reaches 72. If we deliberately delay the transmission starting time with a value uniformly distributed between 0 to 400 µs, then the goodput collapse can be alleviated to 86 servers, and when we increase the maximum delay to 800 µs, the goodput collapse can be relieved to 94 servers. This is because the starting times of the data transmission from the senders are deliberately desynchronized, and the congestion time is delayed. However, if the number of servers increases, the maximum delay should also be increased, which makes the task completion time longer. With the PDN, the sender can retransmit the lost packet immediately before three duplicate ACKs or even retransmission timeouts. Thus, the goodput is greatly improved when the number of servers is large.

Fig. 7 shows the task completion time of the mentioned protocols. Before the number of senders reaching 72, the task completion time of the TCP protocol is around a few milliseconds. However, when the number of senders exceeds 72, the packet losses occurred, and no three duplicate ACKs can be triggered in the TCP protocol; thus, the flows will experience 200 ms timeouts. As a result, the flow completion time is a multiple of 200 ms, which is very long compared with the RTT. If we deliberately delay the flow starting times of different senders randomly, the flow completion time can be maintained low. However, when the number of servers continues to increase, the packet loss will still occur, and the flows will experience the large timeouts again, which degrades the performance. With the PDN, the lost packet can be retransmitted immediately, and no timeout is experienced; thus, the task completion time is very short when compared with other protocols, and this short completion time can be kept as the number of senders increases.

Fig. 8 shows the best-case theoretical task completion time, where we assume that all the flows are coordinated well to avoid congestion, thus the transition delay is only due to the bandwidth, and the propagation delay, which is 100 µs. We compare the task completion time of TCP with PDN with the theoretical task completion time, we found that they are close. However, since there are still congestions, therefore, packet losses and retransmissions, the time spent to complete the data transmission cannot reach the theoretical value. The difference is only 3 ms when the number of servers is 150.

In the next simulation, we measure the maximum notification queue size in the bottleneck switch in rack 1. As shown in Fig. 9, when the number of servers increases to 72, the first packet drop occurs; however, for each notification queue, the maximum number of queued notifications is only 1, which is 20 bytes. Even when we increase the number of senders to 300, the maximum size of each notification queue is still 20 bytes. This is because when the first packet loss occurs, the sender will reduce the cWnd immediately, thus the sending rate will be decreased to prevent further packet losses. Given the notifications are very small (only 20 bytes) compared with data packets (about 1KB), only a small buffer space should be reserved for the notification queues.

In the next simulation, we evaluate the fairness of the TCP with PDN. We first let a server in rack 1 send packets (flow 1) to another server in rack 2 at its maximum rate starting from time 1 second. After we start flow 1, we add an additional flow from a new server in rack 1 to the same server in rack

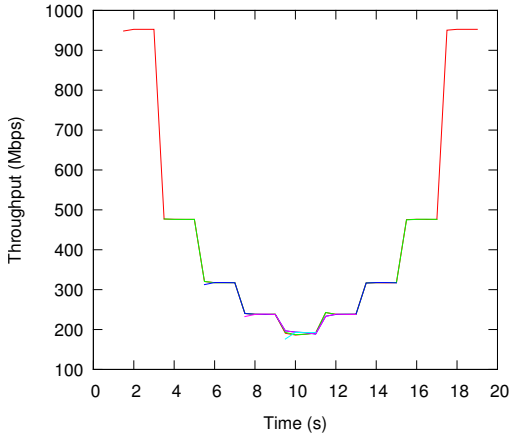

Fig. 10. The fairness between TCP/PDN and the traditional TCP New Reno

2 every 2 seconds, until we have 5 flows in the network. Flow 1 and flow 2 are TCP with PDN flows and flow 3 through 5 are normal TCP New Reno flows. Through this special configuration, we can check the compatibility of TCP with PDN and the traditional TCP protocol (i.e., New Reno). All the flows send packets at their maximum data rate, i.e., 1 Gbps in this simulation, after they are started. Then, we remove a flow every 2 seconds, until we have only 1 flow left. We measure the throughput of each flow at the uplink of the switch in rack 1 to see if the flows can share the bandwidth fairly. The simulation result is shown in Fig. 10. Between the 1st and the 3rd seconds, we have only 1 flow in the network; thus it uses the full bandwidth. After the 3rd second, another flow is started, the bandwidth is now shared by two flows, and each flow can get about 500 Mbps. After the 5th second, three flows share the bandwidth, each flow gets about 300 Mbps of the bandwidth. After the 7th second, the fourth flow is started, each flow now gets about 250 Mbps. After the 9th second, all 5 flows are started, each of them gets about 200 Mbps of the bandwidth. We keep the 5 flows transmitting for 2 seconds. Then, we start removing the fifth flow at the 11th second, then we keep removing the flows every 2 seconds, until we have only the first flow left. As we can see in Fig. 10, the flows all get their fair share of the bandwidth during the whole simulation period, when we increase or decrease the number of flows.

VI. Conclusion

Experiments and analysis indicate that TCP problems in data centers are due to packet losses and retransmission timeouts. To address these imperfections, we proposed to use explicit packet drop notifications (PDNs) to rapidly notify a sender about the packet loss with accurate packet loss information to allow the sender to retransmit the packet immediately. To reduce the overhead of the notifications, we propose to attach the notifications to the existing frames. The simulation results verified that the PDN can effectively solve the problems with the traditional TCP in data centers. In addition, TCP with PDN is compatible with the traditional TCP protocol when they coexist in the same data center, which enables the incremental deployment and simplifies the management of the public data centers.

References

[1] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center tcp (dctcp)," *ACM SIGCOMM CCR*, vol. 41, no. 4, pp. 63–74, 2011.

[2] V. Vasudevan, A. Phanishayee, H. Shah, E. Krevat, D. G. Andersen, G. R. Ganger, G. A. Gibson, and B. Mueller, "Safe and effective fine-grained tcp retransmissions for datacenter communication," in *ACM SIGCOMM CCR*, vol. 39, no. 4, 2009, pp. 303–314.

[3] A. Phanishayee, E. Krevat, V. Vasudevan, D. G. Andersen, G. R. Ganger, G. A. Gibson, and S. Seshan, "Measurement and analysis of tcp throughput collapse in cluster-based storage systems." in *FAST*, vol. 8, 2008, pp. 1–14.

[4] J. Zhang, F. Ren, and C. Lin, "Modeling and understanding tcp incast in data center networks," in *Proc. IEEE INFOCOM*, 2011, pp. 1377–1385.

[5] D. Nagle, D. Serenyi, and A. Matthews, "The panasas activescale storage cluster: Delivering scalable high bandwidth storage," in *Proc. ACM/IEEE conference on SC*, 2004, p. 53.

[6] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[7] R. Niranjan Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat, "Portland: a scalable fault-tolerant layer 2 data center network fabric," in *ACM SIGCOMM CCR*, vol. 39, no. 4, 2009, pp. 39–50.

[8] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "Vl2: a scalable and flexible data center network," in *ACM SIGCOMM CCR*, vol. 39, no. 4, 2009, pp. 51–62.

[9] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "Devoflow: scaling flow management for high-performance networks," in *ACM SIGCOMM CCR*, vol. 41, no. 4, 2011, pp. 254–265.

[10] 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 CCR*, vol. 38, no. 2, pp. 69–74, 2008.

[11] "OpenFlow @ google," http://www.opennetsummit.org/archives/apr12/hoelzle-tue-openflow.pdf.

[12] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown, "Elastictree: Saving energy in data center networks." in *NSDI*, vol. 10, 2010, pp. 249–264.

[13] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks." in *NSDI*, vol. 10, 2010, pp. 19–19.

[14] R. Pan, B. Prabhakar, and A. Laxmikantha, "Qcn: Quantized congestion notification," *IEEE802*, vol. 1, 2007.

[15] P. Devkota and A. N. Reddy, "Performance of quantized congestion notification in tcp incast scenarios of data centers," in *IEEE proc. MASCOTS*. IEEE, 2010, pp. 235–243.

[16] H. Wu, Z. Feng, C. Guo, and Y. Zhang, "Ictcp: incast congestion control for tcp in data-center networks," *IEEE/ACM Trans. on Networking (TON)*, vol. 21, no. 2, pp. 345–358, 2013.

[17] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron, "Better never than late: Meeting deadlines in datacenter networks," in *ACM SIGCOMM CCR*, vol. 41, no. 4, 2011, pp. 50–61.

[18] E. Krevat, V. Vasudevan, A. Phanishayee, D. G. Andersen, G. R. Ganger, G. A. Gibson, and S. Seshan, "On application-level approaches to avoiding tcp throughput collapse in cluster-based storage systems," in *Proc. workshop on Petascale data storage: in conjunction with SC*. ACM, 2007, pp. 1–4.