Improving Multipath TCP for Latency Sensitive Flows in the Cloud

Wei Wang*†,Liang Zhou‡,Yi Sun*
*Institute of Computing Technology, CAS, †University of Chinese Academy of Science
‡University of California, Riverside
E-mail: {wangwei2012@ict.ac.cn}

***Abstract*—Recent cloud datacenter network usually has multiple transmission paths for any pair of servers to increase the aggregate bandwidth and to improve the robustness of failure recovery. Therefore, Multipath TCP (MPTCP) is becoming one of the most important transport protocols exploring the multipath resource to improve the performance of cloud network. It is obvious that MPTCP could significantly increase the bandwidth for the throughput-oriented long flows by using multiple paths concurrently. However, for many latency-sensitive applications that require low latency other than high bandwidth, MPTCP performs poor. In this paper, we propose a set of extensions for MPTCP (called MPTCP-L) to improve its performance on the latency. Particularly, MPTCP-L replicates packets of latency-sensitive flows on two MPTCP subflows to proactively reduce retransmission delay of lost packets. Besides, we propose a rapid packet retransmission scheme that takes only 1∼2 RTTs to recovery the lost packets for the TCP subflows of MPTCP. We conduct large scale simulations and testbed experiments based on real traffic load to evaluate MPTCP-L, and the results prove that MPTCP-L outperforms the standard MPTCP for latency-sensitive flows, and the induced redundancy overhead is negligible.**

Keywords*—Cloud Network, Multipath TCP, Latency, Packet Retransmission

I. Introduction

Cloud computing has become one of the most important infrastructures in the Internet, and large numbers of applications, such as web search, video streaming, file sharing, *etc.*, have been hosted in the cloud [1]. Since most of the applications rely on distributed computing and communicating inside the cloud [27], the performance of cloud network directly affect the user's perceived experience. Generally, the traffic of cloud applications can be classified into two categories: throughput-oriented long flows and latency-sensitive short flows [19] [9]. Long flows usually have large amount of data bytes to transmit, which are often generated by applications like VM migration or data synchronization *etc.*; Short flows are short-lived with little data to transmit, which are often generated by interactive applications like web search. Short flows are latency-sensitive and usually have strict completion time deadline, while long flows only require high bandwidth.

To improve the performance and scalability of cloud network, recent cloud datacenter architectures usually rely on large numbers of commodity switches and are organized as multi-rooted tree (such as Fat-tree [3],VL2 [16]), so there exist multiple shortest paths between any source and destination pairs. Therefore, Multi-Path TCP (MPTCP) [21] has been proposed and widely used inside the cloud network to increase the aggregate bandwidth and robustness by sending data on multiple paths in parallel.

For the throughput-oriented applications, such as VM migration or data synchronization, MPTCP could obviously optimize their aggregate throughput by using multiple available paths. In particular, MPTCP starts up multiple subflows to transmit the traffic concurrently across different paths by ECMP (Equal-Cost Multipath) [18], and loads balancing the traffic from congested paths to idle paths.

However, for the latency-sensitive applications, such as web search, MPTCP contributes little, and even performs worse than the traditional TCP. The traffic of latency-sensitive applications are usually short flows and have moderate bandwidth requirements. Therefore, even the bandwidth of a single path in current cloud network is enough for these applications, thus the aggregate bandwidth increment brought by MPTCP is useless for short flows. On the contrary, each subflow of MPTCP may only send several packets for the short flow, but still at the cost of subflows setup delay and packet re-ordering problem. Besides, the packet loss on any one path will cause the whole flow missing its deadline. In summary, current MPTCP could not benefit latency-sensitive short flows to improve their completion time (detailed analysis are presented in section II).

In this paper, we design a set of extensions to MPTCP (called MPTCP-L) to reduce the latency for short flows in the cloud network. In particular, since the key problem that increases the completion time of short flows is the packet loss [5]. Therefore our main contribution is to reduce the negative effect of packet loss on short flows by improving MPTCP in the cloud network. Firstly, instead of splitting a short flow on MPTCP subflows, we replicate all packets of a short flow and transmit them on the subflows concurrently. By adding a little redundancies into the network, the stalls of MPTCP caused by packet loss will be significantly reduced. Besides, we propose for MPTCP a rapid packet loss detection mechanism that only takes about one RTT (at most two RTTs) to retransmit the lost packets. We also introduce a more aggressive retransmission timer to avoid waiting for the long time RTO and resulting throughput degradation. We implement the proposed extensions to MPTCP in the simulator and Linux kernel, and evaluate them on large scale simulations and small-scale testbed using real cloud traffic traces. The results show that the completion time of short flows are significantly reduced compared with the standard MPTCP.

The rest of the paper is organized as follows: section II analyses the reasons why MPTCP performs poor for latency-

Double retransmission	the retransmitting packet is lost again
Tail retransmission	the lost packet is at the tail
Small cwnd	congestion window is small, *e.g.*, < 3
Small rwnd	receive window is small, *e.g.*, < 3
Continuous loss	all the in-flight packets are lost
ACK loss/delay	the following ACKs are lost or delayed by the delayed ACK mechanisms

TABLE I: Cases when the fast retransmission is ineffective

sensitive flows. Section III presents the detailed design of the MPTCP-L to reduce the latency of short flows, and in section IV the proposed scheme is evaluated and the results are analyzed. The related works are reviewed in section V, and the paper is concluded at last.

II. Background and Motivation

MPTCP utilizes multiple subflows to explore the multipath resource, while each subflow is actually a TCP flow. However, the traditional TCP has been proved to be suboptimal in terms of latency in the cloud network [5]. The sticking point is related to the packet loss. Particularly, the packets loss will prevent the following packets in the receive window from being sent to the applications (*i.e.*, the Head of Line blocking problem). Besides, the loss packet detection and retransmission scheme is also inefficient in the cloud network.

In particular, typical TCP implementation uses 3-duplicated ACKs (fast retransmit) and RTO to indicate packet loss. However, as showed in Table I, there are several situations when the fast retransmit is ineffective [28]. And totally more than 75% of packet loss in short flows are retransmitted until the RTO is triggered, as reported in [15]. However, the default value of RTO in TCP is usually hundreds of millisecond, which is too long to meet the deadline of short flows defined by SLA (Service Level Agreement) in the cloud. An alternative approach is to dynamically calculate the RTO based on the measured RTT, but the RTT in the cloud network varies frequently due to the bursty nature of traffic so that the calculated RTO is often inaccurate and unstable.

As in MPTCP the responsibility of packet loss detection is belong to each TCP subflow, thus MPTCP also performs poor to deal with packet loss. Therefore, in this paper we will propose a novel packet loss detection mechanism for MPTCP to rapidly detect the packet loss within one RTT (at most two RTTs).

When MPTCP detects the packet loss on any subflow, it will retransmit the lost packet on other available subflows to avoid the packet loss again. However, as we know, retransmission is a post-loss approach, and it still needs at least one extra RTT to recover the lost packets. Since MPTCP is able to delivery packets on multiple paths concurrently, it motivates us to actively replicate every packet of short flows on each subflow. Since the probability that all packet replicates are loss is relatively small, the short flow's completion could be obviously reduced. In our design, we propose that working with our rapid packet loss detection scheme, only 2 subflows for replicating the packets are enough to obtain the desired performance.

III. Design of MPTCP-L

We have shown in the previous section that current MPTCP performs poor on the latency-sensitive flows. The key problem can be concluded in two aspects: (1) splitting short flows' packet on multiple subflow is counter-productive in terms of reducing latency. (2) the subflows of MPTCP, *i.e.*, actually TCP flows, have been proved to perform poor to handle packet loss. Therefore, we design a set of extensions to MPTCP aiming to solve above problems. Our scheme MPTCP-L includes 3 step: in first step we propose to replicates packets of short flows on two MPTCP subflows; in step 2 and 3, we present a rapid packet loss detection and retransmission mechanism for MPTCP subflows to reduce the delay of packet loss recovery.

A. Replicate packets on two subflows

Current implementation of MPTCP is designed to schedule the packets on different subflows to increase the aggregate throughput. This is useless to latency-sensitive short flows, because they only have moderate bandwidth requirements. Many studies show that packet loss is the key point that increases the flow's completion time and misses the deadline. Therefore we propose to actively enhance the reliability by adding redundancy into the network. In the semantics of MPTCP, every packet of short flow is replicated on two subflows (as showed in Fig. 1). Thus even the packets are lost on any one subflow, the MPTCP could still receive the packets by the other subflows.

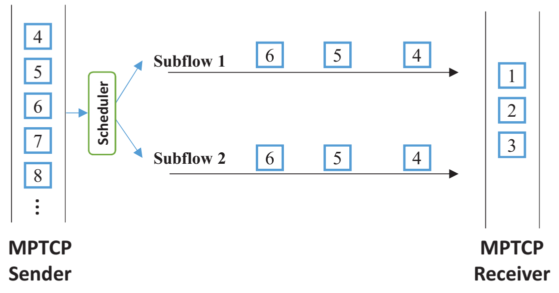

Fig. 1: Replicate packets on two subflows of MPTCP

Obviously, it will be better if there are larger numbers of subflows to replicate the packets. However, its drawbacks include that: (1) too large amount of redundant packets will waste much bandwidth. Even through the data volume of short flows are small, but the number of short flows in the cloud is extreme large, accounting for more than 80% of the total flows numbers. Based on the Long Tail theory, the total data bytes of replicated packets is non-negligible. (2) too large amount of redundant packets will aggravate the Incast problem that has been a critical problem in the cloud network [12]. (3) the redundant packets will make MPTCP be busy with dealing with the replicate packets. Therefore, in MPTCP-L, we only start up two subflows to replicate the short flow's packets, and working with our rapid packet loss retransmission scheme (that is detailed later), it is enough to achieve the same performance improvement as using larger numbers of replication subflows.

The implementation of this extension to MPTCP is simple. We only need to modify the scheduler of MPTCP, *i.e.*, replicating each packet in MPTCP's sending buffer and scheduling them to each subflows. There is no change to the receiver side.

B. Near 1 RTT packet loss detection

In the situation that one replicated packet is lost on the first subflow but the other is received on the second subflow, even through the receive window of MPTCP will not be blocked, the lost packet in the receive window of the first subflow still have to be retransmitted. So the packet loss has to be rapidly detected. In the situation that both replicated packets are lost(small probability, but it may occur), both subflows should detect the packet loss and retransmit them rapidly. We propose a rapid packet loss detection scheme replacing the fast retransmit in TCP that takes near one RTT to detect the packet loss .

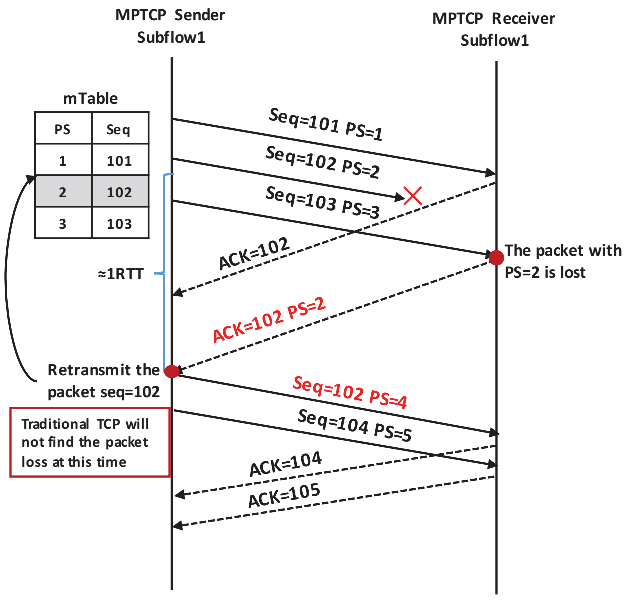

Fig. 2: An example of packet loss detection using Packet Sequence number. The packet with PS=2 is dropped in the network. When the receiver gets the packet with PS=3, it will find that the packet with PS=2 is lost. Then it sets PS=2 in the ACK packet to notify the sender the packet is lost. The sender will query the mTable to find the real packet sequence number (*i.e.*, seq=102), and then retransmit it.

In particular, for each subflow, we assign each packet a positive integer number that is monotone increasing, and we call this number the Packet Sequence (PS). Packet Sequence will not bind with the real sequence number of TCP, but it is only related to the number of packets a subflow has sent. For example, as showed in Fig. 2, firstly the packet with TCP seq=102 is assigned with Packet Sequence=2, but when retransmitting it, its Packet Sequence becomes 4 but the seq is still 102. The MPTCP sender will maintain a mapping table (mTable) for each subflow in which the key is the Packet Sequence number and the value is the real sequence number of TCP. Everytime a packet is sent out, the corresponding entry is added into the mTable.

The Packet Sequence is sent to the receiver by TCP option. The receiver will use it to detect the lost packets. The method is simple, if the received packets' Packet Sequence is not continuous, it indicates that there is packet loss. Then in the following ACK, the lost Packet Sequence are set in the TCP option to tell the sender which packet is lost. If a sequence of packets are lost, the Packet Sequence filed could be set as a range. Once the sender receives the ACK packet with the lost Packet Sequence number, it will firstly searching the mTable for the real sequence number, and then resend the corresponding packet. Fig 2 shows a simple example of the packet loss detection and retransmission using Packet Sequence.

This interaction takes near one RTT to notify the sender to retransmit the lost packets. Therefore, each entry of the mTable will be kept for only 2 RTTs, which guarantees that mTable will not use much memory space. Another advantage of using Packet Sequence for loss detection is that, if the retransmitting packet is lost again, it can also be detected in only one RTT, while in TCP it needs wait for the RTO. A side-affect is that if the packets are out of order, the proposed loss detection scheme can not distinguish this from real packet loss, and causes a spurious retransmission. However, the Packet Sequence number is used separately on each subflow along a single path, so the out of order situation rarely happens.

C. An aggressive timer

There are some situations when the proposed packet loss detection scheme is failed to notify the sender the lost packets. For example, all of the packets in the send window are lost, or the lost packet is the last one, so the receiver can't find the un-continuous Packet Sequence number. Another situation is that the ACK packet carrying the lost Packet Sequence number is lost.

Under these situations, the retransmission needs to wait for the long time RTO, while we introduce a more aggressive timer for retransmissions to reduce the long time TCP stalls of RTO. In particular, we setup a new aggressive timer (that is smaller than RTO) for each subflow (in our implementation, it is set to 2 RTTs). Once the aggressive timer is triggered, the first un-Acked in-flight packets should be retransmitted, but the congestion window will not be halved. Our intention is to aggressively retransmit the packet which is likely to be lost as soon as possible, but not to trigger the RTO. Note that simple reducing the value of RTO is counter-productive, because it will cause large numbers of spurious retransmission and make sender's sending window down to 1 MSS which significantly decreases the throughput and increases the latency.

D. Co-existing with long flows

1) At the end-host: The extensions to MPTCP described above are customized for latency-sensitive short flows. Long flows should still utilize the original MPTCP to load balancing the traffic on subflows and adopt non-aggressive packet-loss detection and retransmission schemes. A simple method to solve this problem is to setup two MPTCP instances: one is for the short flows with our proposed extensions, the other is original MPTCP for long flows. A better method is to implement MPTCP-L in MPTCP as a plugin, which could be dynamically (un)load for different flows. We leave this in our future work. Then another problem is how to detect long flows and short flows which is out of the scope of this paper, but any approach could be used in our scheme. In our implementation, we suppose that the long flows and short flows are a prior knowledge, which could be simply obtained from different types of applications, for example, the flows of VM migration and file synchronization are long flows while the flows of web search are short flows.

2) In the network: MPTCP utilize ECMP to load balancing the subflows across different paths simply based on the hashing of 4 tuples. Since long flows have relatively high flow rates, it is very likely to cause subflows collision on the same paths and to negatively affect the latency of short flows along those paths. Therefore, it is better to setup subflows as many as possible (but at most equal to the path number) for long flows. Thus each subflow of long flow has relatively low rate, so ECMP can works well with this. Besides, the packets of long flows are assigned with a lower priority in the IP TOS field, while packets of short flows are assigned with a higher priority. Then the latency-sensitive packets will not be blocked by long flows.

IV. Evaluation

In this section we present the evaluation results of MPTCP-L for the latency-sensitive flows compared with standard MPTCP. We firstly implement MPTCP-L in the original MPTCP simulator hitsim [2] and conduct large scale simulation experiments on it. Then we implement a prototype of MPTCP-L based on the latest MPTCP v0.90 [10] in the Linux kernel, and evaluate it on a small scale testbed.

A. Large Scale Simulation

The simulation experiments are conducted over an 8-pods Fat-tree topology, and each edge switch connects 10 servers, totally 320 servers. Our traffic load comes from the SWIM project traces [11], which covers 1.5 months of flow level information of Facebook's datacenter. In the workload, more than 80% of the flows are short flows (less than 1MB), and the sources and destinations are randomly set for each flow. In the simulation, for MPTCP-L's experiments, short flows uses the MPTCP-L while long flows utilize the standard MPTCP; for the compared MPTCP's experiments, both short flows and long flows use the MPTCP.

Firstly, we compare the flow completion time(FCT), which is the most important metric for the latency-sensitive flows. To clearly compare the FCT, we set in the first experiment all short flows with the same size. The results are showed in Fig. 3, where x-coordinate is the FCT, and y-coordinates shows the CDF. As we can see, more than 83% of short flows using MPTCP-L are completed within 50ms, while for MPTCP, only about 45% of short flows achieve this. And totally, MPTCP-L

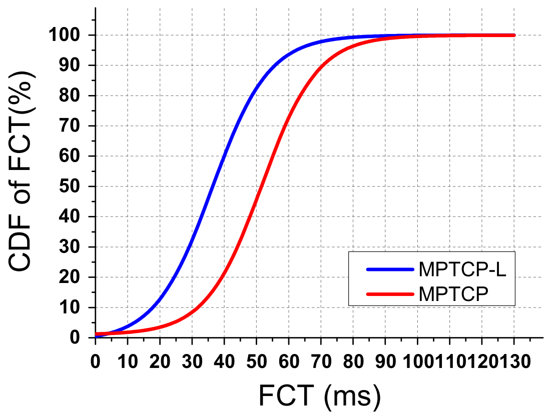

Fig. 3: CDF of short flows' FCT

reduces the average FCT by 23.5% compared with standard MPTCP. Besides, the worst case FCT is more important than the average FCT, because most of the applications in the cloud have the FCT deadline defined by SLA. Once a flow misses the deadline, the flow will be discarded by the upper layer applications. For the worst cases, the 99^th FCT of the flows are completed in about 80ms for MPTCP-L as showed in the Fig. 3, which is decreased by 29.8% compared with the MPTCP's 114ms. This experiment proves that MPTCP-L significantly reduces not only the average FCT but worst case FCT for short flows.

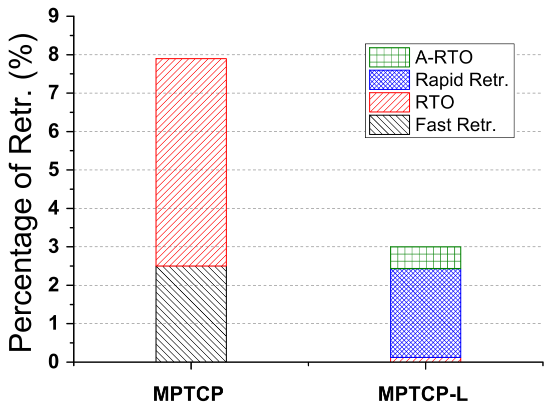

Fig. 4: Percentage of different types of retransmissions in total packets

To deeply explore how MPTCP-L outperforms the standard MPTCP, we present the detailed information of the retransmissions of MPTCP-L and MPTCP as showed in Fig. 4 and 5. Generally, there are only about 3% of total packets retransmitted in MPTCP-L in the experiments, while MPTCP increases this number to 8%, as showed in 4. As we know, retransmission is the key cause to increase the FCT. By replicating packets on two subflows, 97% of MPTCP-L packets are received without retransmission. This accounts for the major part of the improvement of MPTCP-L. Of these retransmissions in MPTCP-L, more than 2.61% of retransmission are triggered by the rapid packet loss detection scheme (denoted as *Rapid*

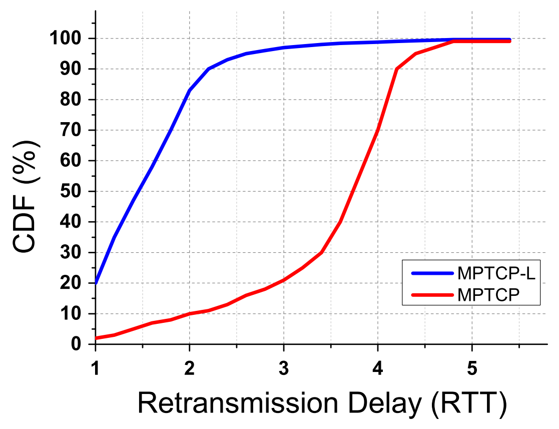

Fig. 5: CDF of retransmission delay

Retr.), and the left 0.27% are handled by the aggressive timer (denoted as *A-RTO*), only 0.02% are triggered by RTO. This means that even for the lost packets, it only takes 1∼2 RTTs to retransmit them. The detailed results are showed in the Fig. 5, in which we plot a CDF of the delay of the different types of retransmissions. It illustrates that about 90% of lost packets are retransmitted within 2 RTTs.

In terms of the MPTCP, of the retransmissions, as showed in Fig. 4, only 2.5% of packets are retransmitted by fast retransmit, while more than 5.4% are retransmitted until the RTO. This outcome also confirms what is reported in [15], that only 25% of retransmission are triggered by fast retransmit. As showed in Fig. 5, the large amount of the RTO based retransmission takes 4 RTTs or longer to retransmit the lost packets[1].

Then we evaluate the replication overhead of MPTCP-L, *i.e.*, the sacrificed bandwidth by the redundancy packets. The replication clearly wastes the bandwidth of the network. However, through calculating percentages of the extra data bytes with respect to all traffic volume, it shows the ratio is only 3∼5% (we repeat the experiments several times to get this value). It is not surprising because short flows only account for a very tiny fraction in the total traffic volume, and the packets are only replicated once.

B. *Testbed Experiments*

We implement MPTCP-L based on the MPTCP v0.90 in the Linux kernel. We build up a small scale testbed, as showed in Fig. 6. There are two servers (one sender and one receiver) connected by 4 different paths. We load background traffic on the 4 paths manually, while the servers only transmit the short flows. We measure mean and 99^th FCT for all short flows under different load of background traffic, as showed in Fig. 7.

It shows that MPTCP-L improves the average FCT by about 10% 26.7% across all traffic load compared with MPTCP. For the tail FCT, the improvement of MPTCP-L

[1]In this experiments, we set RTO of both MPTCP and MPTCP-L as 4RTTs. We also tries smaller RTO (*e.g.*2 RTTs) for the standard MPTCP, but it performs poorer in terms of FCT due to the spurious retransmission and resulting congestion window degradation.

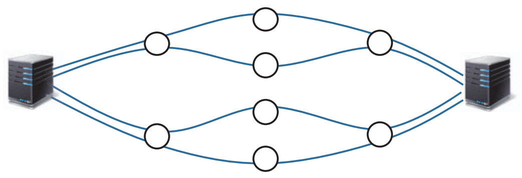

Fig. 6: The testbed topology

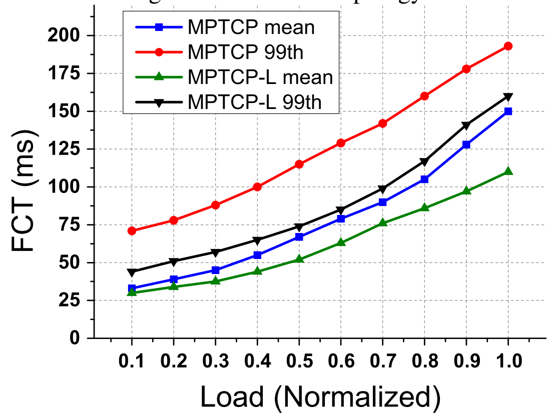

Fig. 7: Mean and 99th FCT of short flows

is more obvious, *i.e.*, the 99^th FCT is improved by about 17∼38% in most cases. And the performance improvement is more significant for the high load situation. The reason has two aspects. Firstly, the loss rate in the heavy-loaded network is high, so the probability of any one replicated packets surviving is much larger than that of only one packet. Secondly, a little more aggressive retransmission scheme proposed in MPTCP-L will be more affective to compete for the scarce bandwidth resource.

V. RELATED WORK

It is well known that TCP performs poor for latency sensitive short flows in cloud data center network [5]. Motivated by this, many approaches have been proposed to reduce the latency. We briefly review the most relevant works in this section.

1) Using redundancy for latency: Sacrificing a little bandwidth resource for improving the latency of short flows is a simple but efficient approach in cloud data center network. The general idea is to add some degree of redundant packets into the network to avoid in-effective retransmissions of TCP. Particularly, there are mainly two ways to achieve this. Firstly, the simplest way is to replicate the packets, which has gained increasing attention in both industry and academia. Google uses in their large-scale online services the request replications to reduce the tail response time [13]. In many distributed systems like MapReduce [14] and GFS [22], replications of the important messages are widely used to reduce the latency. In the academia, authors in [26] propose to use replications, but their approach relies on the applications to replicates the packets which is not feasible especially for the already deployed applications. As a multipath transport protocol, however, MPTCP can naturally and transparently achieve this.

The alternative way is called FEC (Forward error correction), which is initially used in the physical layer. The central idea is that the sender encodes the message in a redundant way by using an error-correcting code (ECC) [24]. And it allows the receiver to re-construct the lost packets using the ECC without needing an extra round to request retransmission of data, but at the cost of a fixed bandwidth sacrifice. Currently, many related works [20] [8], use FEC to improve the latency and reliability in the cloud network. FEC is orthogonal to packet replication used in our approach, and they can work together by replicating the FEC packets.

2) Low latency data center transport: A lot of specialized transport protocols are proposed for low latency and high throughput. DCTCP [5] and HULL [6] use ECN (explicit congestion notification) to pro-actively keep the buffer queue length of the switch at low level to reduce the latency. D2DCP [23], D3 [25], PDQ [17] use preemptive scheduling to complete firstly the flows with closer deadlines. pFabric [7] and Conga [4] propose clean-slate architectures to achieve the low latency. However, all of these works require the modifications to the network or end host, which is not deployable in the near future. MPTCP now currently has been implemented in the kernel, and can run on anywhere TCP can run. Our proposed rapid packet loss retransmissions for MPTCP is not only effective but also deployable.

VI. Conclusion

In this paper, we aim to reduce the FCT of latency sensitive flows by improving MPTCP in the cloud network. We propose to replicate the short flows' packets on two MPTCP subflows to proactively reduce the number of the packet retransmissions which is the key factor of increasing the FCT. Besides, to improve the packet loss detection and retransmission of MPTCP's TCP subflow, we present a rapid retransmission scheme which uses the monotone increasing packet sequence to detect the lost packets, and it takes only 1$\sim$2 RTTs to recovery the lost packets. Through large scale simulation and testbed experiments, the results demonstrate that the mean and tail FCT of short flows are significantly reduced compared with standard MPTCP, and the redundancy overhead induced by packet replication is negligible to the total traffic in the cloud.

VII. Acknowledgement

The project was sponsored in part by grants from National Basic Research Program of China (973) under Grant No. 2012CB315802, National Natural Science Foundation of China under Grant No. 61379133, joint project with RICOH, Natural Science Foundation of Inner Mongolia under Grant No.2015MS0601, and National Project of Culture and Technology Promotion Plan of China under Grant No.201201-02.

References

[1] Amazon aws products. In *http://aws.amazon.com/products/*.

[2] Mptcp simulators. In *http://nrg.cs.ucl.ac.uk/mptcp/implementation.html*.

[3] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. volume 38, pages 63–74, 2008.

[4] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, F. Matus, R. Pan, N. Yadav, G. Varghese, et al. Conga: Distributed congestion-aware load balancing for datacenters. In *ACM SIGCOMM*, volume 44, pages 503–514. ACM, 2014.

[5] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data center tcp (dctcp). *ACM SIGCOMM computer communication review*, 41(4):63–74, 2011.

[6] 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 *NSDI 12*, pages 253–266, 2012.

[7] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker. pfabric: Minimal near-optimal datacenter transport. *ACM SIGCOMM*, 43(4):435–446, 2013.

[8] M. Balakrishnan, T. Marian, K. P. Birman, and H. Weatherspoon. Maelstrom: Transparent error correction for communication between data centers. *Networking IEEE/ACM Transactions on*, 19(3):617–629, 2011.

[9] T. Benson, A. Akella, and D. A. Maltz. Network traffic characteristics of data centers in the wild. In *IMC 2010*, pages 267–280, 2010.

[10] e. a. C. Paasch, S. Barre. Multipath tcp in the linux kernel. In *http://www.multipath-tcp.org*.

[11] Y. Chen, A. Ganapathi, R. Griffith, and R. Katz. The case for evaluating mapreduce performance using workload suites. In *Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2011 IEEE 19th International Symposium on*, pages 390–399. IEEE, 2011.

[12] Y. Chen, R. Griffith, J. Liu, R. H. Katz, and A. D. Joseph. Understanding tcp incast throughput collapse in datacenter networks. In *ACM SIGCOMM 2009 Workshop on Research on Enterprise Networking, Wren 2009, Barcelona, Spain, August*, pages 6–20, 2009.

[13] J. Dean. Achieving rapid response times in large online services. In *Berkeley AMPLab Cloud Seminar*, 2012.

[14] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[15] N. Dukkipati, M. Mathis, Y. Cheng, and M. Ghobadi. Proportional rate reduction for tcp. In *ACM SIGCOMM*, pages 155–170. ACM, 2011.

[16] A. Greenberg, J. R. Hamilton, N. Jain, K. Srikanth, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. Vl2: A scalable and flexible data center network. In *In SIGCOMM*, 2009.

[17] C.-Y. Hong, M. Caesar, and P. Godfrey. Finishing flows quickly with preemptive scheduling. *ACM SIGCOMM*, 42(4):127–138, 2012.

[18] C. E. Hopps. Analysis of an equal-cost multi-path algorithm. 2000.

[19] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken. The nature of data center traffic: measurements & analysis. In *IMC 2009*, pages 202–208, 2009.

[20] H. Lundqvist and G. Karlsson. Tcp with end-to-end forward error correction. pages 152–155, 2004.

[21] C. Raiciu and S. e. a. Barre. Improving datacenter performance and robustness with multipath tcp. *ACM SIGCOMM*, 41(4):266–277, 2011.

[22] S. Saritha. Google file system. 2010.

[23] B. Vamanan, J. Hasan, and T. Vijaykumar. Deadline-aware datacenter tcp (d2tcp). *ACM SIGCOMM*, 42(4):115–126, 2012.

[24] C. Wang, D. Sklar, and D. Johnson. Forward error-correction coding. *Crosslink*, 3(1):26–29, 2001.

[25] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron. Better never than late: Meeting deadlines in datacenter networks. In *ACM SIGCOMM*, volume 41, pages 50–61. ACM, 2011.

[26] H. Xu and B. Li. Repflow: Minimizing flow completion times with replicated flows in data centers. In *INFOCOM, 2014 Proceedings IEEE*, pages 1581–1589. IEEE, 2014.

[27] D. Zats, T. Das, P. Mohan, D. Borthakur, and R. Katz. Detail: reducing the flow completion time tail in datacenter networks. *ACM SIGCOMM*, 42(4):139–150, 2012.

[28] J. Zhou, Q. Wu, Z. Li, S. Uhlig, P. Steenkiste, J. Chen, and G. Xie. Demystifying and mitigating tcp stalls at the server side.