Receive Buffer Pre-division Based Flow Control for MPTCP

Jiangping Han[1,2], Kaiping Xue[1,2(✉)], Hao Yue[3], Peilin Hong[1], Nenghai Yu[1], and Fenghua Li[4]

[1] Department of EEIS, University of Science and Technology of China, Hefei 230027, Anhui, China
kpxue@ustc.edu.cn
[2] Science and Technology on Communication Networks Laboratory, Shijiazhuang 050081, Hebei, China
[3] Department of Computer Science, San Francisco State University, San Francisco, CA 94132, USA
[4] State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China

Abstract. Multipath TCP (MPTCP) enables terminals utilizing multiple interfaces for data transmission simultaneously, which provides better performance and brings many benefits. However, using multiple paths brings some new challenges. The asymmetric parameters among different subflows may cause the out-of-order problem and load imbalance problem, especially in wireless network which has more packet loss. Thus it will significantly degrade the performance of MPTCP. In this paper, we propose a Receive Buffer Pre-division based flow control mechanism (RBP) for MPTCP. RBP divides receive buffer according to the prediction of receive buffer occupancy of each subflow, and controls the data transmission on each subflow using the divided buffer and the number of out-of-order packets, which can significantly improve the performance of MPTCP. We use the NS-3 simulations to verify the performance of our scheme, and the simulation results show that RBP algorithm can significantly increase the global throughput of MPTCP.

Keywords: MPTCP · Receive buffer · Pre-division · Flow control
Wireless · Out-of-order

1 Introduction

Nowadays, the Internet is developing rapidly. Various network access technologies have been developed and used, and one terminal is always equipped with multiple network interfaces. However, traditional TCP only makes use of one interface at a time, which neither takes full advantages of the network resources nor meets the increasing demand on data transmission. Researchers have proposed a number of protocols [1–3] that utilize multipath transmission to solve this problem. The solutions on transport layer are hot topics of the discussion [4–6],

L. Zhu and S. Zhong (Eds.): MSN 2017, CCIS 747, pp. 19–31, 2018.
https://doi.org/10.1007/978-981-10-8890-2_2

since the transport layer is the lowest layer to maintain the end-to-end connection and no change needs to be made at the intermediate nodes. Multipath TCP (MPTCP) [7] has become a new standard supported by IETF MPTCP working group. It can utilize multiple network interfaces simultaneously while is also compatible with existing network systems. Thus, it has received much attention from both academia and industries.

MPTCP could aggregate bandwidth resources and improve overall throughput. However, different from TCP which only uses one path to transmit data, using multiple paths will cause new problems, like the out-of-order problem and load imbalance problem. These problems will lead to the degradation of overall network performance. Especially in today's heterogeneous networks, which has many wireless links and different path parameters. Wireless networks lead to more packet loss and different paths lead to asymmetrical scenarios, will make it harder to collaborate different paths, and the impact will be even severer.

Specifically, out-of-order packets will cause buffer bloat in receive buffer, which will block the data transmission and decrease the throughput. This problem can be solved using scheduling algorithm [8]. The main idea is to schedule the packets and make the packets arrive at receiver in-order. However, there are still some limitations in the scheduling algorithms. Scheduling algorithms are suitable for the situation where there are obvious differences among different subflows' round-trip time (RTT). If the difference is within two times, it will be hard for scheduling algorithms to achieve good performance. On the other hand, scheduling algorithms always reserve a block of sending buffer for the subflow with smaller RTT. If the size of the buffer is limited and insufficient for every subflow, less data will be sent on the subflow with larger RTT, which will then cause the load imbalance problem.

The other method to reduce the out-of-order packets is controlling the traffic flow on each subflow independently according to the characteristics of subflow. In the original flow control of MPTCP, subflows share one receive buffer. The subflows compete against each other, which will cause unreasonable distribution of receive buffer among subflows and make the out-of-order problem even worse. MPTCP uses multiple paths for parallel transmission, where each subflow can easily implement separate flow control. Meanwhile, an independent control for each subflow will lead to better use of the different characteristics among subflows and a more reasonable data distribution. The idea of flow control with evenly divided buffer has been mentioned in CMT-SCTP [9]. But this mechanism does not consider the path difference between subflows.

In this paper, we propose a Receive Buffer Pre-division based flow control For MPTCP (RBP). RBP enables flow control on each subflow separately according to buffer pre-divition, which can control the data distribution among subflows reasonably and solve the above-mentioned problems effectively. The main contributions of our work can be summarized as follows:

– We propose a new flow control mechanism based on receive buffer predivision, which can distribute data according to the performance of subflows. This scheme decreases the influence of bufferbloat and improves the performance of MPTCP.

– We propose a scheme estimating the buffer occupancy of each subflow which dynamically adjusts to the actual situation, thus is more adaptable to the real network.
– Simulation results show that the proposed scheme can effectively improve the throughput of MPTCP.

The rest of this paper is organized as follows. In Sect. 2, we will introduce the flow control of TCP and MPTCP. The details of our algorithm will be described in Sect. 3. In Sect. 4, we evaluated the performance of the proposed algorithm. Finally, we will conclude our work in Sect. 5.

2 Related Work

Scheduling algorithm is an approach to enhance the performance of MPTCP. Scheduling algorithm can solve out-of-order problem caused by asymmetry of paths, and try to ensure the packets to reach receiver in order. When a subflow is under scheduling, sender estimates N, the number of packets which can arrive at sender before the first packet at this subflow, and skips these N packets to send. These N packets will be reserved for other subflows.

There are many recearches on scheduling algorithm. Linux-MPTCP scheduling algorithm [10] is a predictive scheduling algorithm supported in Linux-MPTCP kernel code. Linux-MPTCP scheduling algorithm ignores the change of congestion and other factors and just makes $N = \sum_{j,RTT_j<RTT_i} \left(\frac{RTT_i}{RTT_j} \cdot cwnd_j\right)$. Forward Prediction Scheduling (FPS) [11], Fine-grained Forward Prediction based Dynamic Packet Scheduling (F^2P-DPS) [12], Offset Compensation based Packet Scheduling (OCPS) [13] are the enhancements of Linux-MPTCP scheduling algorithm, which consider the change of congestion window, wireless packet loss and feedback information respectively, and gradually increase the accuracy of schedule algorithms.

Although the modelling of scheduling algorithm is becoming better designed, there are still some limitations as we mentioned before. The scope of application is limited, and sometimes it needs a large buffer. So we can think about this question from another perspective.

In the original MPTCP, receive buffer is shared, and receiver notifies the overall buffer allowance ($rwnd$), which controls the data flow of the total MPTCP connection. When sender receives an ACK from $subflow_i$, it will change the send window of $subflow_i$ as $Send\_Window_i = min(cwnd_i, rwnd)$, where $cwnd_i$ is the congestion window of $subflow_i$, and $rwnd$ is the total advertisement window carried in ACK. If $Send\_Window_i - Outstanding_i > 0$, $subflow_i$ is able to send new data, where $Outstanding_i$ is the unACKed data of $subflow_i$ on subflow level.

In the original MPTCP protocol, $cwnd_i$ is the congestion window of subflow level, and $rwnd$ is the receive window of connection level. Sender makes use of $cwnd_i$ and $rwnd$ to control the data sent on $subflow_i$, which will lead to a mess in different subflows, and cause a decline in the global throughput. If sender can

keep all of the packets arriving at receiver in order or receiver has an infinite buffer, it will be unnecessary to keep flow control on each subflow independently. However, the capacity of different subflows is always different, and it is impossible to make all the packets arriving at receiver in order because of different RTTs and packet loss. In this case, to distinguish each subflow and make an individual flow control will become a better choice.

Concurrent Multipath Transfer using SCTP (CMT-SCTP) [14] is also a multipath protocol on transport layer. Although the discussion of SCTP is gradually weakening, MPTCP could also draw lessons from the thought of flow control scheme. CMT-SCTP came up with a flow control scheme based on buffer allocation [9]. The basic idea is to evenly divide the buffer space into N parts (where N is the number of subflows), and make an independently flow control on each subflow.

However, subflows are always asymmetric in the real station, evenly dividing the buffer space does not consider the inconsistency between parameters of different subflows. Therefore it will not adapt to the actual network well. Sender should consider the capacity of different subflow, and adjust with the actual situation.

3 Receive Buffer Pre-division Based Flow Control Algorithm

To address the problems caused by asymmetric paths of subflows and improve the global throughput of MPTCP in heterogeneous networks, we propose a Receive Buffer Pre-division based flow control algorithm (RBP) for MPTCP. The basic idea of RBP is to distribute the overall $rwnd$ into $rwnd_i$ on each subflow according to the capacity of different subflows. To achieve this goal, we make some modification on the sender. The receiver do not need to make any changes. The receiver still notice the overall receive window to the sender. Then the sender estimates the buffer occupancy of each subflow, and then divides the receive window according to the estimation.

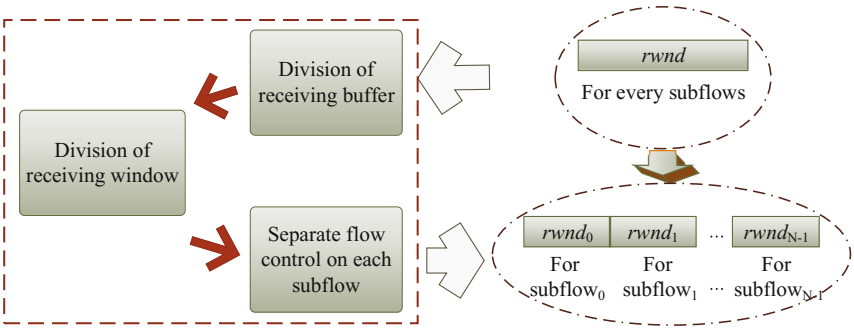

Fig. 1. Receive buffer pre-division based flow control algorithm

Figure 1 shows the basic idea of RBP. The sender first estimates the average maximum buffer occupancy of each subflow in MPTCP, and then divides the receive buffer according to the estimation. After that, the sender counts the unACKed packets on connection level of each subflow, and sets the receive window to the remaining buffer capacity. Finally, the variable *rwnd* will be distributed among all the subflows and the results will be used for flow control on each subflow.

RBP consists of three parts: (1) division of receive buffer; (2) division of receive window; and (3) separate flow control on each subflow. Next, we will describe them in details.

3.1 Division of Receive Buffer

In RBP, the sender first estimates the average maximum buffer occupancy of each subflow, and distributes the buffer based on the estimation results. Buffer occupancy depends on the congestion window and RTT. Notice that congestion window will be changed by congestion control algorithms according to varying path condition, especially in wireless networks with serious packet loss. In order to estimate the congestion window, we assume that the path condition is stable during the estimation.

We use $acwnd_i$ to denote the short-time average size of the congestion window of $subflow_i$. When $cwnd_i$ changes, $acwnd_i$ will be updated as follows:

$$acwnd_i \longleftarrow (1-\beta) \cdot acwnd_i + \beta \cdot cwnd_i, \tag{1}$$

where β is the weight between 0 and 1. Here, we take $\beta = 1/16$, which refers to the update of congestion degree in [15].

Figure 2 illustrates an example of two subflows. The round-trip times of $subflow_0$ and $subflow_1$ are denoted as RTT_0 and RTT_1, respectively. Here,

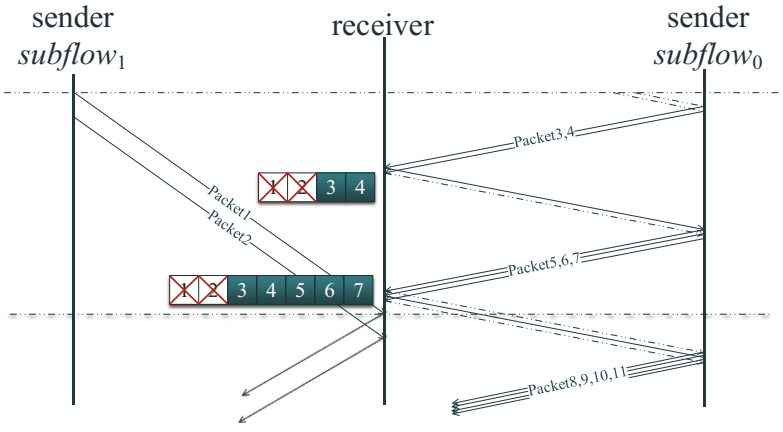

Fig. 2. Estimation of buffer occupy

we assume that $RTT_0 < RTT_1$. The sender estimates the average size of the congestion window for $subflow_0$ as $acwnd_0$.

As shown in Fig. 2, $subflow_1$ will cause out-of-order packets received from $subflow_0$ in the receive buffer. If $subflow_1$ sends its first packet before $subflow_0$, there will be $\left\lceil \frac{RTT_1}{2 \cdot RTT_0} + \frac{1}{2} \right\rceil \cdot acwnd_0$ out-of-order packets in the receive buffer. Otherwise, there will be $\left(\left\lceil \frac{RTT_1}{2 \cdot RTT_0} + \frac{1}{2} \right\rceil - 1 \right) \cdot acwnd_0$ out-of-order packets in the receive buffer. The probabilities of these two cases are both $\frac{1}{2}$. Also, there will be $acwnd_0$ packets that are in transmission from the sender to the receiver, which should also be considered. Therefore, the average buffer occupancy of $subflow_0$ is $acwnd_0 \cdot \left(\left\lceil \frac{RTT_1}{2 \cdot RTT_0} + \frac{1}{2} \right\rceil + \frac{1}{2} \right)$.

When there are more than two subflows in a MPTCP connection, the calculation on the number of out-of-order packets and the average buffer occupancy is similar to that in the above example. For $subflow_i$, every $subflow_j (j \neq i)$ will cause $acwnd\_i \cdot \left(\left\lceil \frac{RTT_j}{2 \cdot RTT_i} + \frac{1}{2} \right\rceil + \frac{1}{2} \right)$ out-of-order packets in the receive buffer. The number of out-of-order packets from $subflow_i$ depends on the maximum value of them, i.e., $\max_{j \neq i} acwnd\_i \cdot \left(\left\lceil \frac{RTT_j}{2 \cdot RTT_i} + \frac{1}{2} \right\rceil + \frac{1}{2} \right)$.

Then, the average maximum buffer occupancy of the $subflow_i$ can be calculated as follows:

$$Buf_i = acwnd\_i \cdot \left(\left\lceil \frac{\max_{j \neq i} RTT_j}{2 \cdot RTT_i} + \frac{1}{2} \right\rceil + \frac{1}{2} \right), \tag{2}$$

where Buf_i is the estimation of the average maximum buffer occupancy of $subflow_i$.

After estimating the average maximum buffer occupancy, the receive buffer will be allocated among all the subflows and the distribution is proportional to the estimated average maximum buffer occupancy, which is shown as follows:

$$B_i = recvBuffer \cdot \frac{Buf_i}{\sum_{i=0}^{N-1} Buf_i}. \tag{3}$$

Suppose there are N subflows in total, B_i is the allocation of available receive buffer to $subflow_i$ and $recvBuffer$ is the variable that contains the size of the receive buffer and will be transmitted to the sender from the receiver at the beginning of the transmission.

3.2 Division of Receive Window

In this part, the sender records the amount of unACKed data on the connection level, and calculates the size of residual available buffer for each subflow. Then, it will allocate $rwnd$ based on the residual available buffer size of each subflow. Each subflow $subflow_i$ obtains $rwnd_i$, which is the receive window on subflow level. Then, the send window of $subflow_i$ will be determined by $rwnd_i$ as follows:

$$rwnd_i = \begin{cases} 0 & B_i \leq unordered_i, \\ \dfrac{rwnd \cdot (B_i - unordered_i)}{\sum\limits_{B_i > unordered_i} (B_i - unordered_i)} & else. \end{cases} \tag{4}$$

Here, $rwnd_i$ is the resudual buffer size distributed to $subflow_i$, $rwnd$ is the size of the total available buffer noticed by the receiver, and $unordered_i$ is the amount of unACKed data on connection level of $subflow_i$. We can observe that the sender allocates $rwnd$ based on the ratio of $(B_i - unordered_i)$. If $(B_i - unordered_i) \leq 0$, which indicates there are too many out-of-order packets from $subflow_i$, the sender will temporarily stop sending data on $subflow_i$.

3.3 Separate Flow Control on Each Subflow

After the above two steps, the shared receive window $rwnd$ will be divided into a set of receive windows $rwnd_i$ for each subflow $subflow_i$. The amount of data transmitted on each subflow will be controlled by the $rwnd_i$. The send window on subflow level slides according to the congestion window and the receive window of each subflow. The send window for a subflow cannot exceed the overall send window for the connection.

When $subflow_i$ is able to send data, the send window of $subflow_i$ will be restricted by the congestion window of the subflow as follows

$$Send\_Window_i = min(cwnd_i, rwnd_i), \tag{5}$$

where $cwnd_i$ is the congestion window of $subflow_i$ and $rwnd_i$ is the receive window of $subflow_i$. The send window of $subflow_i$ cannot exceed the minimum of $cwnd_i$ and $rwnd_i$.

Then, the amount of data that $subflow_i$ can send will be controlled by the send window and the size of unACKed data from $subflow_i$ as follows

$$Send\_data_i = Send\_Window_i - Outstanding_i, \tag{6}$$

where $Outstanding_i$ is the size of unACKed data from $subflow_i$, which is different from $unordered_i$. If $Send\_data_i > 0$, $subflow_i$ is able to send new data.

It can be observed that subflows influence each other when they share the same receive buffer. RBP enables independent flow control on each subflow, and restricts the rate of subflow with too many out-of-order packets. Then, the sender could transmit more data on the subflow with higher throughput, which reduces the number of out-of-order packets and achieves load balancing. RBP does not change the congestion window size. When the number of out-of-order packets on the connection level decreases, it will resume the normal throughput quickly, which can achieve good adaptability to the network. The RBP algorithm is described in Algorithm 1.

Algorithm 1. RBP Algorithm Description

Input:

The receive window: $rwnd$;
The congestion window of $subflow_i$: $cwnd_i$;
The average congestion window of each subflow: $acwnd_1, acwnd_2, ..., acwnd_{N-1}$.

Output:

The amount of new data which is able to send on $subflow_i$: $Send\_data_i$

$acwnd_i \longleftarrow (1-\beta) \cdot acwnd_i + \beta \cdot cwnd_i$
for $j = 0 \rightarrow N-1$ **do**
$Buf_j = acwnd\_j \cdot \left(\left\lceil \frac{\max_{k,k \neq j} RTT_k}{2 \cdot RTT_j} + \frac{1}{2} \right\rceil + \frac{1}{2} \right)$
end for
for $j = 0 \rightarrow N-1$ **do**
$B_j = recvBuffer \cdot \frac{Buf_j}{\sum_{j=0}^{N-1} Buf_j}$
end for
if $B_i \leq unordered_i$ **then**
$rwnd_i = 0$
else

$$rwnd_i = rwnd \cdot \frac{B_i - unordered_i}{\sum_{B_i > unordered_i} (B_i - unordered_i)}$$

end if
$Send\_Window_i = min(cwnd_i, rwnd_i)$
$Send\_data_i = Send\_Window_i - Outstanding_i$

4 Performance Evaluation

In this section, we evaluate the efficiency of the RBP algorithm on NS-3 [16] simulator. The basic MPTCP code is provided by Google MPTCP Group [17]. We use the original MPTCP and TCP on each subflow as comparisons at the same time. We evaluate the performance of the RBP algorithm in terms of the overall throughput, the throughput of subflows, and the number of out-of-order packets in receive buffer.

Table 1. Flow parameters of simulation scenario

Parameters	$subflow_0$	$subflow_1$
Path delay	10 ms – 50 ms	50 ms
Maximum bandwidth capacity	5 Mbps	10 Mbps
Packet loss rate	0.1% – 5%	0.1%
Maximal Segment Size (MSS)	1400 Bytes	1400 Bytes
Congestion control algorithm	TCP-Reno	TCP-Reno

The simulation scenario is shown in Fig. 3. There are two subflows in the MPTCP connection, each of which is routed along a separate path. The middle link on each path is the bottleneck, and therefore the maximum bandwidth capacity of $subflow_0$ and $subflow_1$ is 5 Mbps and 10 Mbps respectively. Each subflow has a wireless link for the last hop to the MPTCP receiver, which suffer from random packet loss. In addition, there is a UDP background flow that is transmitted along each path. The traffic of the UDP flows is uniformly distributed and we set the interval between packets to 5 ms.

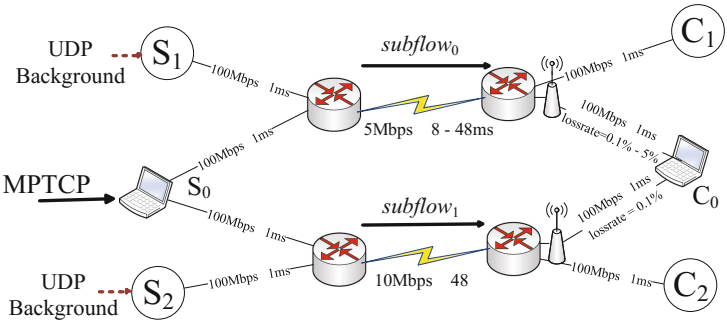

Fig. 3. Simulation scenario

Each subflow has different parameters, such as path delay, packet loss, and so on. Table 1 shows the parameter configuration of $subflow_0$ and $subflow_1$. In addition, the MSS of each subflow is set to 1400 Bytes, and the size of shared receive buffer is 2×65536 Bytes. The receive buffer of single TCP is 65536 Bytes, and the MSS of UDP is set to 1024 Bytes. For each scenario, we take the average values of 50 simulation runs as results.

4.1 Asymmetric Scenario

The different path delay among subflows is an important factor that causes the decrease of overall throughput in asymmetric scenarios. In a network, if two subflows have the same path delay, it will result in the best performance. However, it is difficult to achieve the best performance since different subflows often have different path delay in real network.

In the first scenario, we change the path delay while maintain the other parameters as constants. Specially, the path delay of $subflow_0$ changes from 10 ms to 50 ms, and the delay of $subflow_1$ is always 50 ms. We also set the loss rate of $subflow_0$ to 0.1% in the simulations.

Figure 4 shows the result of overall throughput. It can be observed that the global throughput of these two schemes decrease simultaneously as the path delay of $subflow_0$ increases. Because the throughput of a subflow will be affected by the path delay, and large delay may lead to a decrease in throughput. However,

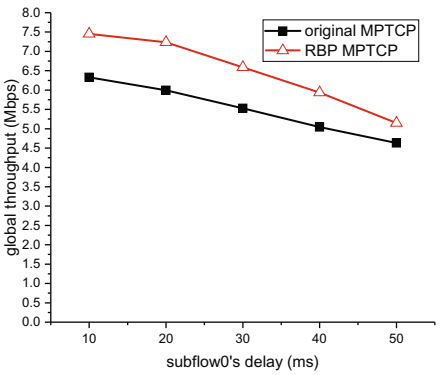

Fig. 4. Comparison of overall throughput with the change of path delay

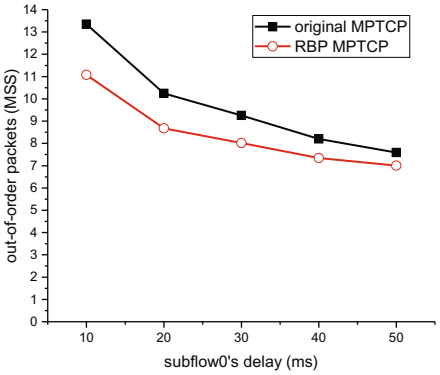

Fig. 5. Comparison of out-of-order packets with the change of path delay

from Fig. 4 we can see that RBP brings a transmission gain on MPTCP, which will result in a better performance. In the best case, RBP can achieve a gain nearly 20%. However, transmission gain of RBP will be a little lower if the path delay of $subflow_0$ and $subflow_1$ are similar, because the symmetric paths will cause fewer out-of-order packets.

The results on the number of out-of-order packets in the receive buffer is shown in Fig. 5. With the path delay increasing, the difference between the delays of $subflow_0$ and $subflow_1$ becomes smaller, and the number of out-of-order packets will significantly decrease, since the difference of path delay is the most important factor that leads to the out-of-order packets in the receive buffer. In addition, when RBP with independent control on each subflow is used, the number of out-of-order packets will also decrease. At the beginning, when the RTT of $subflow_0$ is much smaller than $subflow_1$, the number of out-of-order packets decreases a lot with RBP algorithm. The gain will become smaller as the difference between two subflows decreasing.

Figure 6 shows the throughput of different subflows with original MPTCP and RBP algorithm. As shown Fig. 4, RBP brings a gain on the total throughput. Figure 6 shows more details of the gain. The throughput of $subflow_0$ does not decrease too much with RBP algorithm, but the throughput of $subflow_1$ increases significantly, which will improve the overall throughput.

4.2 Wireless Scenario

In practice, random packet loss mostly occurs in wireless networks, which may lead to the decrease of throughput. We change the packet loss rate of $subflow_0$ from 0.1% to 5%, and maintain the other parameters as constants. The delay of $subflow_1$ is always 50 ms. We also set the path delay of $subflow_0$ to 20 ms in this scenario.

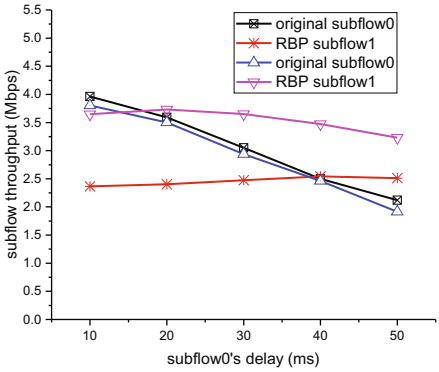

Fig. 6. Comparison of subflow throughput with the change of path delay

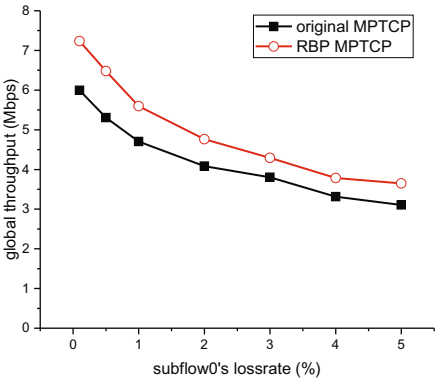

Fig. 7. Comparison of overall throughput with the change of loss rate

Figure 7 shows the simulation results of overall throughput. With the random packet loss rate on $subflow_0$ increasing, the throughput of these two schemes decreases. The reason is that a majority of congestion control algorithms are based on packet loss, and large packet loss rate will cause poor throughput. Moreover, large packet loss rate will lead to asymmetric situation of data sending between subflows, which causes more out-of-order packets in connection level, and hence the overall throughput will be further reduced. It can also be observed that RBP still outperforms the original MPTCP.

The results on the number of unordered packets are shown in Fig. 8. Since RBP controls data transmitted on each subflow independently and reasonably distributes data among subflows, it can effectively decrease the number

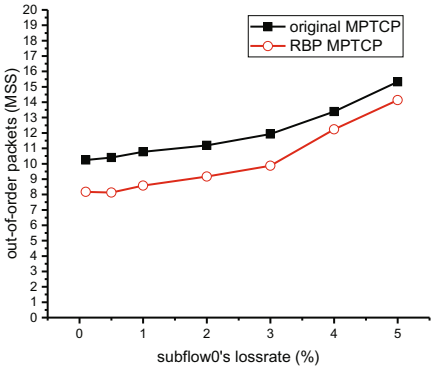

Fig. 8. Comparison of out-of-order packets with the change of loss rate

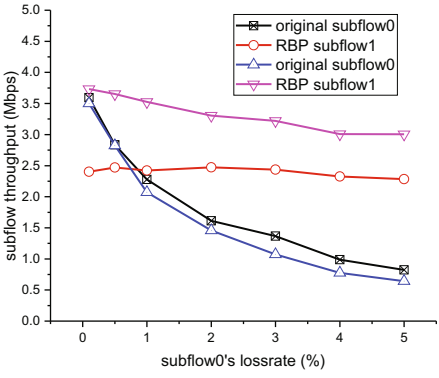

Fig. 9. Comparison of subflow throughput with the change of loss rate

of out-of-order packets. We can see RBP works better on the scenario with less packet loss, because the packet loss will lead to more out-of-order packets and make it difficult to estimate the transmission state. However, from Fig. 8 we can see RBP still brings a better performance on out-of-order packets in lossy scenarios.

Figure 9 shows the throughput of subflows. When the RBP algorithm is used, the throughput of $subflow_1$ is higher than that with original MPTCP. At the same time, the throughput of $subflow_0$ does not significantly decrease. Thus, RBP still improves the overall throughput. Likewise, we can see that RBP allocates more data on $subflow_1$ than original MPTCP, which indicaets RBP allocates more data on the best subflow, so as to achieve load balance between subflows.

5 Conclusions

MPTCP uses multiple paths for data transmission at the same time, which needs to be more precisely controlled. However, each subflow has its own congestion window, but there is only one receive window on connection level. If there is nothing different between subflows, and each packet can arrive at receiver in order, it will be unnecessary for keeping separate flow control on each subflow. However, the wireless network and asymmetrical paths lead to a degradation of MPTCP, which is caused by misallocation of data among subflows.

RBP scheme makes independent flow control on each subflow and adjusts to the actual situation. Thus it can regulate and control the data sent on each subflow in detail. If there are too many out-of-order packets on one subflow, which is beyond the permitted scope, sender will limit the data on the subflows, so as to reduce the amount of out-of-order packets, and promote throughput of the correct subflows, thus could also reduce bufferbloat and provide a better performance. Therefore RBP can achieve the purposes of improving the overall throughput and balancing traffic load between subflows, which can improve the network performance greatly.

Acknowledgment. This work is supported by the National Natural Science Foundation of China under Grant No. 61379129 and No. 61671420, the Fund of Science and Technology on Communication Networks Laboratory under Grant No. KX162600024, Youth Innovation Promotion Association CAS under Grant No. 2016394, and the Fundamental Research Funds for the Central Universities.

References

1. Habak, K., Harras, K.A., Youssef, M.: Bandwidth aggregation techniques in heterogeneous multi-homed devices: a survey. Comput. Netw. **92**, 168–188 (2015)
2. Lee, W., Koo, J., Park, Y., Choi, S.: Transfer time, energy, and quota-aware multi-RAT operation scheme in smartphone. IEEE Trans. Veh. Technol. **65**(1), 307–317 (2016)

3. Zheng, X., Cai, Z., Li, J., Gao, H.: Scheduling flows with multiple service frequency constraints. IEEE Internet Things J. **4**(2), 496–504 (2017)
4. Amer, P., Becke, M., Dreibholz, T., Ekiz, N., Iyengar, J., Natarajan, P., Stewart, R., Tuexen, M.: Load sharing for the stream control transmission protocol (SCTP). IETF Personal Draft, draft-tuexen-tsvwgsctp-multipath-13 (2016)
5. Li, M., Lukyanenko, A., Ou, Z., Ylä-Jääski, A., Tarkoma, S., Coudron, M., Secci, S.: Multipath transmission for the internet: a survey. IEEE Commun. Surv. Tutor. **18**(4), 2887–2925 (2016)
6. Shailendra, S., Bhattacharjee, R., Bose, S.K.: MPSCTP: a simple and efficient multipath algorithm for SCTP. IEEE Commun. Lett. **15**(10), 1139–1141 (2011)
7. Ford, A., Raiciu, C., Handley, M., Bonaventure, O.: TCP extensions for multipath operation with multiple addresses. IETF RFC, RFC6824 (2013)
8. Xue, K., Han, J., Ni, D., Wei, W., Cai, Y., Xu, Q., Hong, P.: DPSAF: forward prediction based dynamic packet scheduling and adjusting with feedback for multipath TCP in lossy heterogeneous networks. IEEE Trans. Veh. Technol. **67**(2), 1521–1534 (2017)
9. Adhari, H., Dreibholz, T., Becke, M., Rathgeb, E.P., Tüxen, M.: Evaluation of concurrent multipath transfer over dissimilar paths. In: Proceedings of 2011 IEEE Workshops of International Conference on Advanced Information Networking and Applications (WAINA 2011), pp. 708–714. IEEE (2011)
10. Barré, S., et al.: Implementation and assessment of modern host-based multipath solutions. Ph.D. dissertation, UCL (2011)
11. Mirani, F.H., Boukhatem, N., Tran, M.A.: A data-scheduling mechanism for multi-homed mobile terminals with disparate link latencies. In: Proceedings of the 72nd IEEE Vehicular Technology Conference Fall (VTC 2010-Fall), pp. 1–5. IEEE (2010)
12. Ni, D., Xue, K., Hong, P., Shen, S.: Fine-grained forward prediction based dynamic packet scheduling mechanism for multipath TCP in lossy networks. In: Proceedings of the 23rd International Conference on Computer Communication and Networks (ICCCN), pp. 1–7. IEEE (2014)
13. Ni, D., Xue, K., Hong, P., Zhang, H., Lu, H.: OCPS: offset compensation based packet scheduling mechanism for multipath TCP. In: Proceedings of 2015 IEEE International Conference on Communications (ICC 2015), pp. 6187–6192. IEEE (2015)
14. Iyengar, J.R., Amer, P.D., Stewart, R.: Concurrent multipath transfer using SCTP multihoming over independent end-to-end paths. IEEE/ACM Trans. Netw. **14**(5), 951–964 (2006)
15. Kühlewind, M., Wagner, D.P., Espinosa, J.M.R., Briscoe, B.: Using data center TCP (DCTCP) in the internet. In: Proceedings of 2014 IEEE Globecom Workshops (GC Wkshps), pp. 583–588. IEEE (2014)
16. NS3 simulator. www.nsnam.org/
17. MPTCP NS3 code. http://code.google.com/p/mptcp-ns3/