Fine-grained Forward Prediction based Dynamic Packet Scheduling Mechanism for Multipath TCP in Lossy Networks

Dan Ni[1], Kaiping Xue[1]*, Peilin Hong[1], Sean Shen[2]*
[1] The Department of EEIS, University of Science and Technology of China, Hefei, Anhui 230027 China
[2] DNSLAB, China Internet Network Information Center, Beijing 100190 China
*kpxue@ustc.edu.cn, sean.s.shen@gmail.com

***Abstract*—Nowadays, multi-interface terminals in heterogeneous network may access Internet through various access technologies, and further aggregate network resources from multiple paths as much as possible. Multipath TCP exploits multiple paths simultaneously by stripping data of a connection over multiple TCP flows, each of which is through a disjoint path. However, it encounters the problem caused by the great number of out-of-order packets at receiver due to dissimilar path characteristics, i.e. latency, bandwidth, packet loss rate, etc. The previous intelligent scheduling mechanisms to keep in-order delivery all ignored packet losses and became fragile in lossy networks. In this paper, we present Fine-grained Forward Prediction based Dynamic Packet Scheduling Mechanism(F^{2}P-DPS) for multipath TCP. It utilizes the idea of TCP modeling to estimate the latency on the path under scheduling and the data amount sent on the other paths simultaneously, which takes packet loss rate into consideration, and then decides which packets to send on the under-scheduling path. From the simulation, we can see that our mechanism obviously improves throughput and reduces cache occupancy at receiver in lossy networks.**

I. Introduction

As various radio access technologies(RATs) overlap and form heterogeneous networks, it is common that mobile terminals equipped with multiple network interfaces, which make data transmission through different interfaces simultaneously become possible. Unfortunately, currently used protocols utilize only a single interface at a time even though multiple interfaces are connected. Theoretically, reasonable data-stripping solutions to exploit multiple interfaces can aggregate the available network resources of different RATs to provide better service, such as higher bandwidth, better connectivity and so on. However, if each interface is relative to a different path, simultaneous transmission over different paths for one application stream leads to packet reordering due to the dissimilar path characteristics, i.e. latency, bandwidth, packet loss rate, etc. It can adversely affect the performance of any real-time applications.

In recent years, there are solutions of data stripping [1] on multi-interface terminals to enhance concurrent transfer across multiple paths for different purpose, such as load sharing, in-order delivery, fairness and so on. These solutions can be implemented at different layers to efficiently schedule packets. Firstly, we talk about some typical scheduling schemes in link layer and network layer. Link layer solutions of traffic distribution over multi-radio networks has been studied in [2]. It is based on the MAC-layer measurement, dispersing traffic across the links in proportion to their available capacities. The most notable network layer solution is the earliest delivery path first (EDPF) scheduling [3]. It transmits the packet through the path that delivers it the earliest. However, these lower layer scheduling solutions are transparent to transport layer, which means the transport layer doesn't aware of these multiple paths. Regarding to TCP, it can't distinguish out-of-order from packet loss, which leads to lots of unnecessary retransmission.

Application layer solutions [4], [5] are also proposed, which can use either TCP or UDP to transmit. [4] adopts an approach that works in the case of using HTTP, which issues a set of range queries each using a separate connection on a different interface. MuniSocket [5] describes the design and implementation of an UDP-based socket that utilizes multiple network interfaces connected through heterogeneous networks. However, the applications need modifications to be aware of multiple interfaces.

Consequently, transport layer solution better suits multi-interface terminals for the reason not only it is the lowest layer to keep end-to-end semantics among peers, but also it can provides transparency for applications. Since TCP has good features, such as providing reliable delivery and guaranteeing fairness, multipath transmission protocols based on TCP have been studied [6], [7], [8], [9] and some are further specified by IETF WG, such as Load Sharing for SCTP(LS-SCTP) [10], which is improved on Stream Control Transmission Protocol(SCTP) [11], and Multipath TCP(MPTCP) [12]. In TCP-based multipath transmission, a connection can be composed of multiple TCP flows and the scheduling is packet-oriented, which means packets of a connection are scheduled individually and sent over different TCP flows. It can exploit multiple paths simultaneously if each TCP flow is relative to a disjoint path. When disjoint paths have significantly different latencies, the scheduling function should be carefully designed because the packets are strictly ordered and large number of out-of-order packets at receiver will increase the end-to-end delay and decrease the throughput heavily.

In this paper we propose a new scheduling algorithm for MPTCP: Fine-grained Forward Prediction based Dynamic Packet Scheduling(F^{2}P-DPS). It allocates some specific packets to under-scheduling TCP subflow by estimating the amount of packets that will be transmitted on other TCP subflows simultaneously. The estimation is done by utilizing TCP characteristics of each TCP subflow within a connection. It adopts the idea of TCP modeling and takes account of packet loss, which is more adaptive and suitable in lossy networks.

Remainder of this paper is organized as follows. In section II, we discuss related works of the scheduling algorithms for multipath transmission based on TCP. Section III elaborates the idea of F^{2}P-DPS by an example of two-path MPTCP scenario. F^{2}P-DPS adopts the idea of TCP modeling. Furthermore, in section IV, we use NS-3 to verify that the scheduling algorithm we propose can reduce the number of out-of-order packets at receiver and enhances total throughput. Section IV concludes the paper.

II. Related Work

There are several multipath transmission protocols based on TCP, such as LS-SCTP(improved on SCTP) and MPTCP. They support to establish a connection with multiple TCP flows through different paths, thus scheduling algorithm is essential to distribute data efficiently over multiple TCP flows.

Stream Control Transmission Protocol(SCTP) [11] is a transport layer protocol and supports multi-homing, serving in a similar role to TCP and UDP. Multi-homed terminal with SCTP assigns a different IP address for each interface and uses them in a single ”association”, which is similar to ”connection”. SCTP supports the sender to establish an association with a primary path and reserves alternative paths for retransmission or back-up. If the primary path fails, the alternative paths can be used.

Concurrent Multipath Transfer using SCTP(CMT-SCTP) [13] extends SCTP to support simultaneously use multiple paths within a SCTP association. The sender simply adopts round robin(RR) manner without intelligence, where it just schedules data from the sending buffer in sequence to the available congestion window space of the next path. However, It can’t alleviate the effects brought up by heterogeneous path characteristics, such as packet reordering. Packets with larger sequence number may arrive at receiver earlier than expectation, and have to wait until the sequence numbers are continuous. The number of out-of-order packets aggregated from multiple paths arises due to the dissimilar and timely changed path characteristics(e.g, latency, bandwidth, packet loss rate). A large receive buffer is required to cache out-of-order packets, which leads to large waiting delay and heavily degrades the throughput.

Just as mentioned above, in-order delivery in a single connection over multiple paths is important. Load Sharing for SCTP(LS-SCTP) [10] supports weighted round robin and distributes data to each path in proportion to the ratio cwnd/RTT(congestion window/round trip time). However, it is coarse-grained and can’t ensure in-order delivery for each packet. WestwoodSCTP(W-SCTP) [14] performs a more intelligent bandwidth aware scheduling at sender, which is named as BAS. It scores for each path, and the path with the lowest score has the highest priority to transmit packets. It tries to provide in-order delivery but still suffers from serious performance degradation if the paths in an association have significantly different latencies.

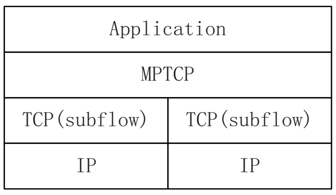

Fig. 1. MPTCP Internet stack

Forward Prediction Packet Scheduling(FPS) for multi-interface terminals with disparate latencies is introduced in [15], which is verified in SCTP. When a path under scheduling frees congestion window space to pull new data from sending buffer, it estimates the duration of this new transmission. Then it estimates the number of packets(N) that can be delivered simultaneously in other paths during this given duration. The estimation is based on the assumption that no loss will occur, thus the congestion window size will increase for every RTT. Then the under-scheduling path chooses $(N + 1)$th packet and the following ones from the sending buffer to fill its congestion window. If the packets experience no loss, the estimation is precise and FPS can keep windows sliding smoothly at both sides and enhances throughput. However, in lossy networks where losses should be taken into consideration, it becomes fragile.

For SCTP isn’t compatible with regular TCP, it is difficult to implement SCTP in the current network. Recently, Multipath TCP(MPTCP) [12] is raised to conquer the problems in SCTP. MPTCP can establish a MPTCP connection with multiple regular TCP subflows, each may be on a different path, and it can just fall back to regular TCP when there is only a TCP subflow. As shown in Fig.1, it adds MPTCP layer above TCP layer to the networking stack, with the original transport layer called subflow layer. Subflow layer can just adopt regular TCP. In MPTCP layer, it divides data of a connection into several portions and schedule them on parallel TCP subflows. Since the packets are strictly ordered by sequence number, scheduling algorithm at MPTCP layer is essential and needs more intelligence to improve the chances of in-order delivery at connection level.

In current MPTCP specification [12], it simply uses round robin(RR) manner to schedule data. MPTCP employs a large receive buffer shared by all the subflows to hold out-of-order packets. Linux-MPTCP scheduler [16] is an intelligent scheduler implemented in Linux MPTCP kernel [17]. The amount of data scheduled on each TCP subflow is in proportion to the estimated bandwidth of the path, calculated by $BW = cwnd/RTT$. Besides, it has the intelligence to choose which packet to allocate from the shared sending buffer. Nonetheless, it doesn’t utilize the TCP characteristic of each subflow and becomes fragile in lossy networks.

Above all, in-order delivery among different paths is a big challenge for multipath transmission, some packets arrive much earlier than others through quicker paths will cause head-of-line blocking problem. What's more, when multiple paths have significantly different latencies, packet reordering will become critical. The intelligent scheduling algorithms at sender have been proposed in former researches to minimize the chance of out-of-order, however they all ignore packet losses and provide less robustness in lossy networks.

III. Fine-grained Forward Prediction based Dynamic Packet Scheduling Mechanism(F^{2}P-DPS)

In this section, we propose a new intelligent scheduling mechanism named Fine-grained Forward Prediction based Dynamic Packet Scheduling(F^{2}P-DPS) for MPTCP. The scheduling function at sender intelligently distributes packets in a connection over multiple TCP subflows, each of which is through a different paths.

F^{2}P-DPS is closer to FPS, but more fine-grained and provides more robustness in lossy networks. When a subflow is under scheduling, the sender predicts the variation of TCP window size for each faster subflows in the same connection, and estimates the data amount N_{total} sent on them during one successful delivery time on the under-scheduling subflow. The under-scheduling subflow then select $(N_{total} + 1)$th packet and the following ones to fill its congestion window. The estimation model is the key issue to be solved. FPS ignores packet loss, while F^{2}P-DPS takes losses into consideration. F^{2}P-DPS adopts the idea of TCP modeling, which models TCP's behavior for each TCP subflow by averaging all the possible packet loss events. In this section, we yield a expressions for N_{total}, as a function of RTT and packet loss rate.

We adopt most of our terminology from TCP modeling [18], [19], which develops the characteristics of steady-state throughput and latency. We assume each subflow adopts TCP Reno and model TCP Reno in terms "round", the same as that in [19]. A round starts with transmission of the packets in current congestion window. The first ACK reception marks the end of the current round and the beginning of the next round. The duration of a round is equal to the RTT. We make some other assumptions as follows:

1) On every TCP subflow, the packet arrives at the receiver side in order if the packets are not lost in transmission.

2) On every TCP subflow, loss indication is triggered by triple duplicated ACKs or timeout.

3) The modeling of TCP Reno adopts independent loss model, which means the loss rate of a packet is independent of any loss rate of other packets.

4) The time needed to send all the packets in a round is smaller than RTT.

We elaborate the main idea of F^2-DPS through a two-path MPTCP scenario illustrated in Fig.2. Both endpoints(e.g.client and server) support MPTCP and a connection with two TCP subflows(e.g.subflow0, subflow1) has been established between them. These two subflows are transmitted on dissimilar paths, where path1 experiences larger delay than path0, which means $RTT_1 > RTT_0$. The packet loss rates of these two paths are p_0 and p_1 separately. The packets of the connection generated at client can go through either path0 or path1. Besides, it is allowed that more subflows through different paths join the connection. In this text, we just use the two-path scenario for the simplification of analysis.

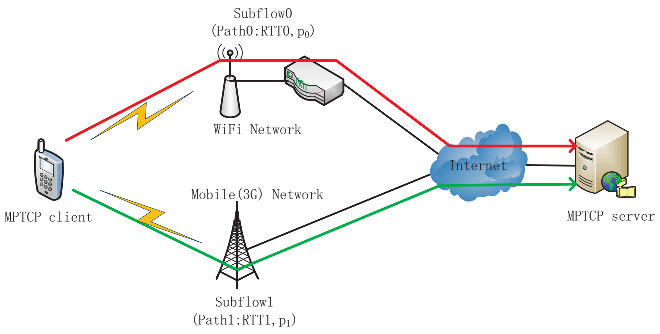

Fig. 2. An example of two-path MPTCP scenario

Fig.3 shows packet transmission process of the two-path MPTCP scenario. At time instance t, subflow1 is ready to send new packets, the expected receipt time of these new packets is t'. In Fig.4, we model subflow0's behavior during $[t, t']$. We assume the first loss indication occurs at packet i_1 on subflow0, and the number of lost packets in the same round is $nloss_1$. These i_1 packets are transmitted within r_1 rounds, and the total number of packets transmitted in r_1 rounds is $n_{max}(i_1)$. When these $nloss_1$ losses are all recovered, the number of packets have already been transmitted except for the retransmitted ones is N_1. Finally, the number of packets transmitted on subflow0 during $[t, t']$ is N_{total}, which is derived by averaging all the possible packet loss events. The detailed modeling steps are as follows:

A. Expected t'

In Fig.3, at time instance t, subflow1 has free congestion window space, $cwnd1$, and is ready to send new packets. The expected transfer time of these $cwnd1$ packets is derived, which is similar with the derivation of latency in TCP modeling [18]. And the expected receipt time of these $cwnd1$ packets is t'.

If all these $cwnd1$ packets are transmitted successfully on subflow1, the successful delivery time is $RTT_1/2$ and the possibility is $\hat{p}1 = (1 - p_1)^{cwnd1}$.

Otherwise, if $nloss$ packets are lost, there are two cases to be discussed. In the first case, $cwnd1 - nloss < 3$, the sender will not receive enough ACKs and resort to a timeout. The average duration of a timeout, accounting for the exponential backoff of the retransmission timer on the loss of a retransmitted packetis given by

$$E(TO) = TO\frac{1 + p^2 + 2p^2 + 4p^3 + 8p^4 + 16p^5 + 32p^6}{1 - p} \tag{1}$$

where TO is the duration of time the sender waits before retransmitting the first lost packet on the specific subflow. With

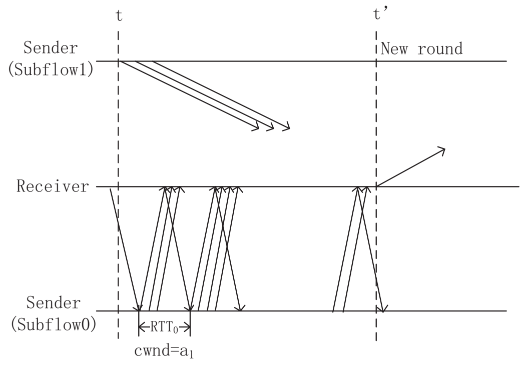

Fig. 3. Packet transmission process of two-path MPTCP scenario

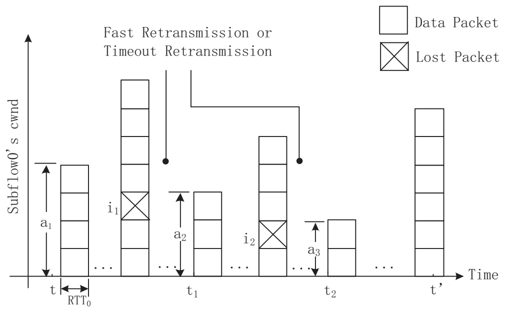

Fig. 4. Modeling TCP's behavior on subflow0 during [t,t']

p_1 the packet loss rate and TO_1 the waiting time for a timeout on subflow1, $E(TO_1)$ is obtained. Thus, the successful delivery time in this case is $E(TO_1)$ and the possibility is $\hat{p}2 = cwnd1 * (C^1_{cwnd1-1}(1-p_1) * p_1^{cwnd1-2} + C^2_{cwnd1-1}(1-p_1)^2 * p_1^{cwnd1-3})$. In the second case, $cwnd1 - nloss \geq 3$, the sender will receive enough ACKs and enter fast retransmission to recover, the successful delivery time is $1/2RTT_1 + RTT_1 = 3/2RTT_1$ and the possibility is $\hat{p}3 = 1 - \hat{p}1 - \hat{p}2$. By averaging all these possibilities, t' satisfies

$$t' = t + \frac{RTT_1}{2}\hat{p}1 + \frac{3RTT_1}{2}\hat{p}2 + E(TO_1)\hat{p}3 \tag{2}$$

B. Calculation of r_1 and $n_{max}(i_1)$

Fig.4 shows TCP's behavior on subflow0 during $[t, t']$. The $cwnd$ of the first round on subflow0 is a_1, and the first loss indication occurs at packet i_1. Let $cwnd^0_{i_1}$ denote the congestion window of the round when the i_1th packet is transmitted. These i_1 packets may be transmitted in slow start phase or congestion avoidance phase within r_1 rounds, and the total number of packets transmitted in r_1 rounds is $n_{max}(i_1)$. We derive the expression for r_1 and $n_{max}(i_1)$, which are the function of i_1 and a_1. Considering the different phases these i_1 packets may be in, there are three cases as follows:

1) Slow start phase: If $cwnd^0_{i_1} < ssthresh$, these i_1 packets are all transmitted in slow start phase and cwnd increases exponentially. $ssthresh$ is the slow start threshold. With a_1 the cwnd of the first round, r_1 is obtained by solving $\sum_{n=1}^{n=r_1-1} a_1 * 2^{(n-1)} \leq i_1 \leq \sum_{n=1}^{n=r_1} a_1 * 2^{(n-1)}$, thus

$$r_1 = \left\lceil \log_2\left(\frac{i_1}{a_1} + 1\right) \right\rceil \tag{3}$$

$$n_{max}(i_1) = a_1 * 2^{(r_1-1)} \tag{4}$$

2) Congestion avoidance phase: If $a_1 > ssthresh$, the i_1 packets are all transmitted in congestion avoidance phase and cwnd is increased linearly. With a_1 the cwnd of the first round, r_1 is obtained by solving $\sum_{n=1}^{n=r_1-1}(a_1 + n - 1) \leq i_1 \leq \sum_{n=1}^{n=r_1}(a_1 + n - 1)$, thus

$$r_1 = \left\lceil \frac{\sqrt{4a_1^2 - 4a_1 + 1 + 8i_1} + 1 - 2a_1}{2} \right\rceil \tag{5}$$

$$n_{max}(i_1) = \frac{(2a_1 + r_1 - 1) * r_1}{2} \tag{6}$$

3) Both phases: If $cwnd^0_{i_1} > ssthresh$ and $a_1 < ssthresh$, these i_1 packets are transmitted firstly in slow start phase and then congestion avoidance phase. The slow start phase ends when cwnd reaches $ssthresh$, the number of rounds,rss, in slow start phase can be obtained by solving $a_1 * 2^{r_{ss}-1} \leq ssthresh \leq a_1 * 2^{r_{ss}}$, thus

$$r_{ss} = \left\lceil \log_2 \frac{ssthresh}{a_1} \right\rceil. \tag{7}$$

The $(r_{ss}+1)$th round enters in congestion avoidance phase, the cwnd of this round is $a'_1 = ssthresh + 1$ and the left number of packets is $i'_1 = i_1 - a_1 * (2^{r_{ss}} - 1)$. Thus, the number of rounds to transmit the left packets in congestion avoidance phase, referring to Enq.(5), is given by

$$r_{ca} = \left\lceil \frac{\sqrt{4a'^2_1 - 4a'_1 + 1 + 8i'_1} + 1 - 2a'_1}{2} \right\rceil \tag{8}$$

Thus, r_1 and $n_{max}(i_1)$ in this case are

$$r_1 = r_{ss} + r_{ca} \tag{9}$$

$$n_{max}(i_1) = a_1 * 2^{(r_{ss}-1)} + \frac{[2(ssthresh+1) + r_{ca} - 1] * r_{ca}}{2} \tag{10}$$

C. Calculation of N_1

Since multiple losses are allowed in a round, we denote $nloss_1$ the number of losses in r_1th round, including the first loss. For this given $(i_1, nloss_1)$, TCP's behavior to recover these lost packets is certain no matter where the losses happen. Let t_1 the time instance when $nloss_1$ are all recovered and the sender starts to send a new round of data, we can then calculate the total amount of packets, N_1, sent in $[t, t_1]$ excluding retransmitted ones.

We denote $cwnd^0_{i_1}$ the cwnd of $r_1(i_1)$th round. The cwnd of the round following is $cwnd^1_{i_1}$, the new data transmitted in it besides the packets in $cwnd^0_{i_1}$ is $nrnd^1_{i_1}$. The number of successfully transmitted packets in $cwnd^0_{i_1}$ is $ndup^1_{i_1}$, it also represents the number of duplicated ACKs received by sender. Thus, we have

$$cwnd^1_{i_1} = cwnd^0_{i_1} \tag{11}$$

$$nrnd^1_{i_1} = cwnd^0_{i_1} - n_{max}(i_1) + i - 1 \tag{12}$$

$$ndup^1_{i_1} = cwnd^0_{i_1} - nloss_1 \tag{13}$$

Considering that losses can be recovered by fast retransmission or timeout retransmission, there are two cases as follows:

1) Fast retransmissions: If $ndup^1_{i_1} \geq 3$, the sender will get at least three duplicated ACKs for packet i_1 and enter fast retransmission and recovery. Otherwise, the sender resorts to a timeout. We denote t_1 the time instance when loss is recovered after fast retransmission and N_1 the total amount of packets excluding retransmitted ones during$[t, t_1]$, thus

$$t_1 = t + (r_1 + 1) * RTT_0 \tag{14}$$

$$N_1 = n_{max}(i_1) + nrnd^1_{i_1} \tag{15}$$

After fast retransmission, cwnd is reduced to $a_2 = max\{2, cwnd^1_{i_1}/2^{nloss_1}\}$ and the sender enters into congestion avoidance phase with $cwnd = a_2$.

2) Timout retransmission: If $ndup^1_{i_1} < 3$, the sender will not receive not enough duplicated ACKs and resort to a timeout to recover. With p_0 the packet loss rate and TO_0 the waiting time for a timeout on subflow0, the average duration of a timeout is $E(TO_0)$ according to Enq.(1).

$$t_1 = t + r_1 * RTT_0 + E(TO_0) \tag{16}$$

$$N_1 = n_{max}(i_1) + nrnd^1_{i_1} \tag{17}$$

where $nrnd^1_{i_1}$ is obtained from Enq.(12). After a timeout retransmission, cwnd is set to $a_2 = 1$ and sender enters into slow start phase with $cwnd = a_2$.

D. Expected N_{total}

During the period $[t_1, t']$, subflow0's initial cwnd is a_2, and we assume the first loss indication occurs at packet i_2. Hence, t_2, N_2 is obtained in the same way as t_1, N_1, and $t_3, N_3; ...t_l, N_l$ can also be acquired. Here, t_l must be satisfied

$$t_l \leq t', t_{l+1} > t' \tag{18}$$

For each $(i_k, nloss_k)$, it provides the place of the first loss and the number of all the losses in the same round. TCP's behavior to recover these lost packets is certain no matter where the following losses happen. It means N_k is the same if different packet loss events have the same i_k and $nloss_k$. The number of possible packet loss events with given $(i_k, nloss_k)$ is $C^{nloss_k}_{cwnd^1_{i_k}}$. The probability, denoted by P_k, of this given $(i_k, nloss_k)$ during$[t_{k-1}, t_k]$ is

$$P_k = C^{nloss_k}_{cwnd^1_{i_k}} * p_0^{nloss_k} * (1 - p_0)^{N_k - nloss_k} \tag{19}$$

Once pairs $(i_1, nloss_1), (i_2, nloss_2)...(i_l, nloss_l)$ are given, $N_1, N_2, ...N_l$ and $P_1, P_2...P_l$ can be acquired referring to Enq.(15), Enq.(17) and Enq.(19). Let N_{total} the total number of packets sent on subflow0 during$[t, t']$ and P the probability of these given pairs, we have

$$N_{total} = N_1(i_1) + ... + N_l(i_l) \tag{20}$$

$$P = \prod_{k=1}^{l} P_k \tag{21}$$

The expected data amount N_{total} transmitted on subflow0 is obtained by averaging Enq.(20) for the possible value of $nloss_1...nloss_l, i_1...i_l$. Thus,

$$E(N_{total}) = \sum_{\substack{nloss_1...nloss_l \\ i_1...i_l}} P * N_{total} \tag{22}$$

Finally, $E(N_{total})$ is the total number of packets estimated to be transmitted successfully during period $[t, t']$ on subflow0, subflow1 takes $(E(N_{total}) + 1)$th packet and the following ones from sending buffer to fill its congestion window.

F^{2}P-DPS provides an estimation model by averaging N_{total} acquired from all the possible packet loss events on subflow0. But the true value is the result of one trial, and our estimated value may be deviate from the true value. Even though, F^{2}P-DPS provide more precise estimation model because it considers the packet losses. Inevitably, F^{2}P-DPS brings higher complexity.

IV. Performance Evaluation

We evaluate our scheduling mechanism(F^{2}P-DPS) proposed in this paper on NS3 simulator. Also we implement another scheduling mechanism(FPS) for comparison. Both these intelligent scheduling mechanisms work to schedule packets dynamically over multiple TCP flows on different paths.

Compared with the simple round robin scheduler(RR), FPS enhances the chances of in-order delivery through multiple paths and improves the global throughput. The simulation results can certify that F^{2}P-DPS is more suitable in lossy networks, improving global throughput and reducing the number of out-of-order packets.

In the simulation, the sender establishes a connection containing two TCP subflows with the receiver, each subflow is through a different path. These two different paths have significantly different path characteristics. It is allowed that more subflows join the same connection, then the data will be distributed over all the subflows to aggregate the network resources. But in our simulation scenario, we only establish two subflows through two different paths.

TABLE I
Configuration of subflow0 and subflow1

Configurations	subflow0	subflow1
Delay(ms)	50	250
Bandwidth(Mbps)	5	2
Loss Rate(%)	0.1~ 5	0.1
MSS(bytes)	1400	1400

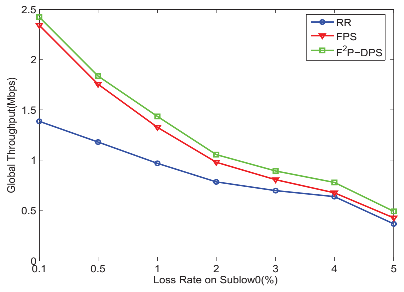

Fig. 5. Global throughput

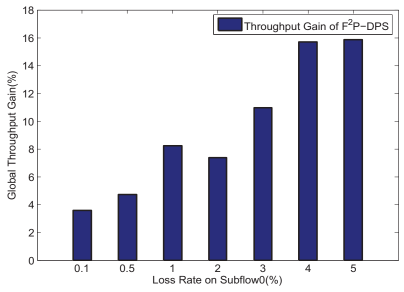

Fig. 6. Global throughput gain of F^{2}P-DPS based on FPS

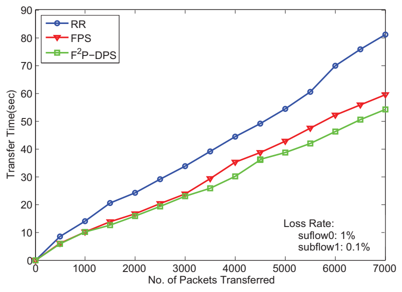

Fig. 7. Transfer time of the packets

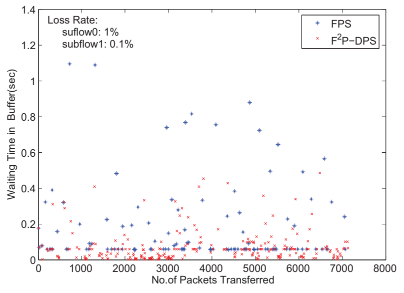

Fig. 8. Waiting time in the receive buffer of the out-of-order packets

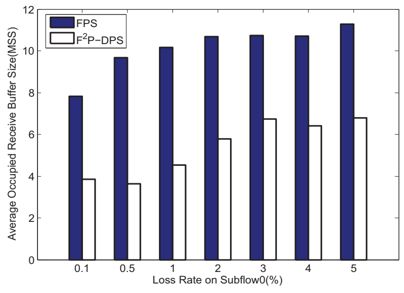

Fig. 9. Average of occupied receive buffer size

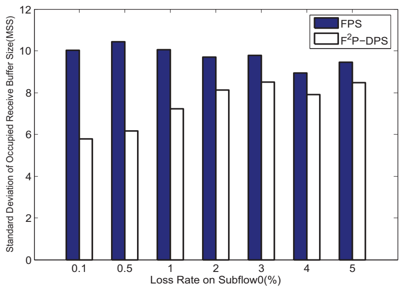

Fig. 10. Standard deviation(SD) of occupied receive buffer size

The configuration of these two subflows on different paths is presented in Table.I. The bandwidth of subflow0 and subflow1 are 5Mbps and 2Mbps respectively. The delay on subflow0 is 50ms, while the delay on subflow1 is 250ms. The random loss rates on subflow0 vary from 0.1% to 5%, which represent the lossy network on different levels. However, the loss rate on subflow1 is fixed to 0.1%. Maximal Segment Size(MSS) in our simulation is set to 1400 bytes, which is also the packet size at TCP layer. At endpoints, the sender generates 10MB data to sent out, and the receive buffer size is set to 64KB, shared by both TCP subflows.

The transfer time of 10MB data from sender to receiver in both mechanisms are recorded in the simulation. With loss rate on TCP subflow0 increasing, the transfer time increases accordingly and the throughput, calculated by $10MB * 8/transfertime$, degrades. Fig. 5 illustrates the global throughput of RR, FPS and F^{2}P-DPS, where RR is the scheduling mechanism without any intelligence. We note that both FPS and F^{2}P-DPS outperform RR, that is to say, the throughput can be improved if any intelligent scheduling algorithm is used. Also, the throughput of FPS and F^{2}P-DPS decreases with loss rate on subflow0 increasing. It is because the precision of the estimation model in both mechanisms degrades when packet loss events occur frequently. Anyhow, F^{2}P-DPS outperforms FPS at any loss rate, improving the global throughput in lossy networks.

To see the throughput enhancement more clearly, we define throughput gain of F^{2}P-DPS based on FPS to measure it, which is given by

$$ThoughputGain = \frac{Th_{F^2P-DPS} - Th_{FPS}}{Th_{FPS}} * 100\% \quad (23)$$

where Th_{F^2P-DPS},Th_{FPS} are the global throughput of F^{2}P-DPS and FPS respectively. The global throughput gain is presented in Fig. 6, from which we note that the global throughput gain of F^{2}P-DPS can reach as much as 15%. Fig. 7 illustrates the transfer times of packets, which shows the relationship between the sequence number of each packet and the time when the sender receives cumulative ACK for the packet. The result indicates that F^{2}P-DPS has the highest throughput all the time. In this experiment, the packet loss rates on subflow0 and subflow1 are fixed to 1% and 0.1%.

Further, we evaluate F^{2}P-DPS in another perspective: out-of-order packets. To the best of our knowledge, out-of-order packets at receiver will have to wait in the buffer until the packets are correctly reordered at connection level, and then these packets will be submitted to the upper layer. We plot the waiting time of out-of-order packets at receive buffer in Fig. 8. F^{2}P-DPS can provide more precise estimation model than FPS and reduce the re-ordering time for the packets as well. In addition, occupied receive buffer size plays an important role of measuring the number of reordering packets. Large occupied receive buffer size indicates large number of reordering packets and leads to the degrading of MPTCP's performance. In the simulation, we read the size of occupied receive buffer and record it for every 100ms. Fig. 9 and Fig. 10 show the average and standard deviation(SD) of all the records, the buffer size is counted by MSS(1400 bytes).

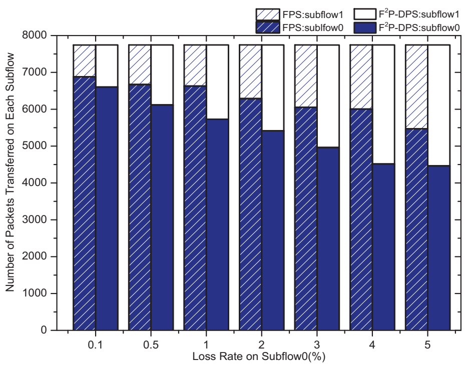

Fig. 11. Number of packets transferred on each subflow

The occupied buffer size of both mechanisms increases with loss rate on subflow0 increasing as shown in Fig. 9, but FPS always occupies more receive buffer size. The SD curves in Fig. 10 also illustrate that F^2P-DPS has more robustness than FPS in lossy networks.

Fig. 11 illustrates the number of packets taken over by each subflow. The result shows that F^2P-DPS can more efficiently utilize subflow1 by scheduling more data to it, making data load more balance between two subflows. Even though subflow1 is less congested compared with subflow0, its large delay leads to low throughput. F^2P-DPS tries hard to make sending window of each subflow slide more quickly and improves the throughput on slower path.

We can conclude that the estimation model of F^2P-DPS provides more precision than FPS in lossy networks. It improves the global throughput acquired from all the paths and reduces the number of reordering packets at receiver.

V. Conclusions

Multipath TCP(MPTCP) can exploit heterogeneous paths by implementing an intelligent packet scheduling mechanism at sender. The data in a connection should be carefully scheduled to multiple TCP subflows on different paths with minimal occurrence of reordering at receiver. However, some previous scheduling mechanisms, such as FPS, provide a coarse estimation model ignoring any losses. We present Fine-grained Forward Prediction based Dynamic Packet Scheduling(F^2P-DPS) in this paper for packet-loss environment. Our mechanism estimates the data amount to be transmitted on all the other paths during one successful delivery time on the under-scheduling path by modeling TCP's behavior. The modeling takes cwnd, RTT, and packet loss rate into account and averages all the possible packet loss events that may happen. Our model adopts TCP Reno and is based on the assumption of independent random losses. F^2P-DPS is validated to outperform FPS from two important perspectives, global throughput and the number of out-of-order packets in lossy networks.

F^2P-DPS can be applied to not only MPTCP, but also any multipath transmission based on TCP. It provides a more accurate estimation model compared with any other scheduling mechanisms, which just ignore packet losses. Since the behaviors of some other TCP versions are more complicated, F^2P-DPS based on these TCP versions remains to be solved. Furthermore, F^2P-DPS with the correlated or bursty losses should also be considered in the future.

Acknowledgment

This work is supported by the National Natural Science Foundation of China under Grant No. 61379129, No. 61170231, DSLAB Open Foundation and Intel ICRI MNC.

References

[1] Karim Habak, Khaled A Harras, and Moustafa Youssef. Bandwidth aggregation techniques in heterogeneous multi-homed devices: A survey. *arXiv preprint arXiv:1309.0542*, 2013.

[2] Jong-Ok Kim, Tetsuro Ueda, and Sadao Obana. Mac-level measurement based traffic distribution over ieee 802.11 multi-radio networks. *Consumer Electronics, IEEE Transactions on*, 54(3):1185–1191, 2008.

[3] Kameswari Chebrolu and Ramesh Rao. Communication using multiple wireless interfaces. In *Proceedings of IEEE WCNC*, volume 1, pages 327–331. Citeseer, 2002.

[4] Dominik Kaspar, Kristian Evensen, Paal Engelstad, and Audun Fosselie Hansen. Using http pipelining to improve progressive download over multiple heterogeneous interfaces. In *Communications (ICC), 2010 IEEE International Conference on*, pages 1–5. IEEE, 2010.

[5] Nader Mohamed, Jameela Al-Jaroodi, Hong Jiang, and David R Swanson. A user-level socket layer over multiple physical network interfaces. In *IASTED PDCS*, pages 804–810, 2002.

[6] Y Hasegawa, I Yamaguchi, Takayuki Hama, H Shimonishi, and T Murase. Improved data distribution for multipath tcp communication. In *Global Telecommunications Conference, 2005. GLOBECOM'05. IEEE*, volume 1, pages 5–pp. IEEE.

[7] Luiz Magalhaes and Robin Kravets. Mmtp: multimedia multiplexing transport protocol. *ACM SIGCOMM Computer Communication Review*, 31(2 supplement):220–243, 2001.

[8] Chung-Ming Huang, Ming-Sian Lin, and Lik-Hou Chang. The design of mobile concurrent multipath transfer in multihomed wireless mobile networks. *The Computer Journal*, 53(10):1704–1718, 2010.

[9] Samar Shailendra, R Bhattacharjee, and Sanjay K Bose. Mpsctp: a simple and efficient multipath algorithm for sctp. *Communications Letters, IEEE*, 15(10):1139–1141, 2011.

[10] P Amer. Load sharing for the stream control transmission protocol (sctp). *draft-tuexen-tsvwg-sctp-multipath-07*, October 2013.

[11] R Stewart and Q Xie. Stream control transmission protocol. *RFC 2960*, October 2000.

[12] A Ford. Tcp extensions for multipath operation with multiple addresses. *RFC 6824*, January 2013.

[13] Janardhan R Iyengar, Paul D Amer, and Randall Stewart. Concurrent multipath transfer using sctp multihoming over independent end-to-end paths. *Networking, IEEE/ACM Transactions on*, 14(5):951–964, 2006.

[14] C Casetti and W Gaiotto. Westwood sctp: load balancing over multipaths using bandwidth-aware source scheduling. In *Vehicular Technology Conference, 2004. VTC2004-Fall. 2004 IEEE 60th*, volume 4, pages 3025–3029. IEEE, 2004.

[15] Farhan H Mirani, Nadia Boukhatem, and Minh Anh Tran. A data-scheduling mechanism for multi-homed mobile terminals with disparate link latencies. In *Vehicular Technology Conference Fall (VTC 2010-Fall), 2010 IEEE 72nd*, pages 1–5. IEEE, 2010.

[16] Sébastien Barré. *Implementation and assessment of modern host-based multipath solutions*. PhD thesis, Citeseer, 2011.

[17] http://www.multipath-tcp.org/.

[18] Biplab Sikdar, Shivkumar Kalyanaraman, and KS Vastola. Analytic models and comparative study of the latency and steady-state throughput of tcp tahoe, reno and sack. In *Global Telecommunications Conference, 2001. GLOBECOM'01. IEEE*, volume 3, pages 1781–1787. IEEE, 2001.

[19] Jitendra Padhye, Victor Firoiu, Donald F Towsley, and James F Kurose. Modeling tcp reno performance: a simple model and its empirical validation. *IEEE/ACM Transactions on Networking (ToN)*, 8(2):133–145, 2000.