On Mitigating Packet Reordering in FiWi Networks

Shiliang Li[1,3], Jianping Wang[1], Chunming Qiao[2], and Bei Hua[3]

[1] Department of Computer Science, City University of Hong Kong, Hong Kong
{lshiliang2,jianwang}@cityu.edu.hk
[2] Department of Computer Science and Engineering, SUNY at Buffalo, NY
qiao@computer.org
[3] Department of Computer Science, University of Science and Technology of China
bhua@ustc.edu.cn

Abstract. In an integrated fiber and wireless (FiWi) access network, multi-path routing may be applied in the wireless subnetwork to improve throughput. Due to different delays along multiple paths, packets may arrive out of order, which may cause TCP performance degradation. Although the effect of packet reordering due to multi-path routing has been well studied, remedy solutions are either to schedule packets at the source node to proactively reduce the chance of packet reordering, or to modify TCP protocol. Resequencing packets arrived out-of-order has only been considered at the end systems which can cause long delay as packets must be buffered until there is no sequence gap. As all traffic in a FiWi network is sent to the Optical Line Terminal (OLT), the OLT serves as a convergence node which naturally makes it possible to resequence packets at the OLT before they are sent to the Internet. However, the challenge is that OLT must re-sequence packets effectively with a very small delay to avoid a performance hit. In this paper, we propose a scheduling algorithm at the OLT to resequence packets while providing fairness. Simulation results validate that our packet scheduling algorithm is effective in improving the performance of TCP flows. Since resequencing is conducted in the access network which has a much fewer number of flows compared with those at routers, our proposed work provides a scalable solution to mitigate the side-effect of packet reordering caused by multi-path routing.

Keywords: FiWi, PON, WMN, Packet Reordering, Multi-path Routing, Resequence.

1 Introduction

Recently, the hybrid fiber-wireless (FiWi) [1] access network integrating the passive optical networks (PONs) and wireless mesh networks (WMNs) has been proposed to provide cost efficient, high bandwidth and ubiquitous last mile Internet access. A FiWi network consists of a PON subnetwork and a wireless subnetwork as shown in Fig. 1. In the PON subnetwork of a FiWi network, Optical Line Terminal (OLT) resides in a Central Office (CO) and feeds multiple

X. Jun Hei and L. Cheung (Eds.): AccessNets 2009, LNICST 37, pp. 89–102, 2010.

Optical Network Units (ONU). In the wireless subnetwork of a FiWi network, WMN is deployed for ubiquitous communications at users' premises. Typically, the WMN consists of multiple gateways for the Internet access where one or more gateways can be connected to an ONU through wired line, a group of wireless mesh routers that provide multi-hop wireless communications and a group of wireless mesh clients.

Multi-path routing has been widely considered in the wireless/wired networks as an approach to achieve load balancing, fault tolerance, and a higher network throughput [2–5]. In a FiWi network, packets of a flow can be sent through multiple paths in the wireless subnetwork to the OLT so that network congestion is alleviated and throughput can be improved. These packets, however, may be reordered when they arrive at the OLT due to delay variance along different paths. As a result, the increase of throughput by exploiting multi-path routing may be affected by packet reordering [6, 7].

In the literature, many efforts have been made to mitigate the effects of packet reordering caused by multi-path routing. The work can be classified into three main categories: (1) to determine which path each packet should be sent to so that packet reordering can be proactively avoided. FLARE is introduced as a traffic splitting algorithm in [10] where it is shown that it is possible to systematically slice a TCP flow across multiple paths without causing packet reordering. The work in [8] studied how to route packets efficiently at the sender side. Two traffic congestion control techniques, namely, flow assignment and packet scheduling, have been investigated in [8]. (2) to modify the TCP protocol to improve TCP performance. The work in this category needs to be implemented at TCP clients to generate congestion responses when packet reordering occurs, and/or at participating routers to report packet dropping information to TCP clients. Most reordering-tolerant approaches are sender-side solutions which increase the threshold of fast retransmission. RR-TCP [11] uses the false fast retransmission avoidance ratio (FA ratio) to adjust *dupthresh*. The work in [12] provides receiver-side solutions which delay ACKs for out-of-order segments and immediately sends ACKs for retransmitted segments. A more comprehensive survey of reordering-tolerant algorithms can be found in [14]. (3) to resequence packets at the end systems. Resequencing packets to deliver the arrived packets to the application in sequence has been well studied in the literature [15, 16] where resequencing is conducted at the end application. In such work, packets stay in the resequencing buffer until there is no sequencing gap in the accepted packets.

All above mentioned works resolve the effects of packet reordering at either the source node or the destination node. One question arose is whether some work can be done at the intermediate "nodes" to resequence out-of-order packets. Resequencing is seldom considered at the routers since a router may forward packets for many flows. Obviously, if an "intermediate" node which can resequence packets is at the access network where there are relatively fewer number of flows than routers, resequencing can be considered as one of the effective approaches to avoid packet reordering at the core network. In a FiWi network,

the OLT serves as the gateway to the core network. Since there is only a single path from the OLT to the destination in the core network for each connection, it can be expected that packets from each flow will arrive at their destination in almost the same order as they are sent from the OLT. In other words, if the OLT can resequence the packets, it can mitigate the effects of packet reordering caused by multi-path routing in the wireless subnetwork. Thus, throughput can be improved. Compared with resequencing at the end system, resequencing at the OLT requires negligible resequencing delay as packets may experience unpredictable delay at the core network and we can not afford long resequencing delay at the access network. The tight resequencing delay implies that 100% in-order resequencing is impossible to achieve when resequencing is conducted at the intermediate node. Thus, a fast resequencing algorithm which can reduce the out-of-order probability of packets departing from the OLT is desired.

In this paper, given the out-of-order (OOD) packet arrivals at the OLT from different flows, we propose a packet scheduling algorithm at the OLT which aims to resequence the packets of each flow to assure possible in-order arrivals at the destinations. The work proposed in this paper requires that the OLT is capable of maintaining some information for each flow (as to be introduced shortly, minimum amount of information for each flow is maintained at the OLT to provide scalability). We would like to note that some scheduling algorithms at the E-PON OLT [17] has been proposed to provide per-flow and per-class forwarding discipline to satisfy various types of QoS constraints for downstream traffic.

The rest of the paper is organized as follows. In Section 2, we give an example to demonstrate how resequencing may be able to mitigate the effect of packet reordering. The problem description is given in Section 3. Section 4 presents the scheduling algorithm. Section 5 evaluates the proposed algorithm through simulations, and Section 6 concludes the paper.

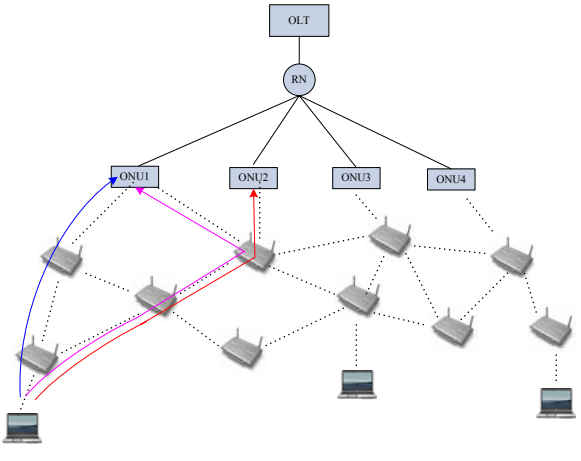

Fig. 1. A conceptual architecture of FiWi Networks

2 Motivation Example

As mentioned earlier, the OLT serves as a *convergence* node in the FiWi network. For the upstream traffic, the OLT needs to send packets from different flows to their destinations where the sending sequence at the OLT will be the last chance in a FiWi network to ensure in-order packet delivery in the core network. In this section, we give one motivation example to demonstrate how packet scheduling algorithm can resequence packets and mitigate the effect of packet reordering.

Suppose that there are two flows sending packets through the OLT to different destinations as shown in Fig. 2 (a). Seven packets are now in the OLT's buffer, and two packets will arrive at the end of time slots 4 and 5 respectively. We denote $P_{i,j}$ as the j-th packet of flow i, and $A_{i,j}$ as the acknowledgment of $P_{i,j-1}$. The service rate of the OLT's outgoing link is one packet per time slot.

Suppose that Round Robin (RR) scheduling mechanism is applied at the OLT. Then these packets will arrive at destinations in the order of $\{P_{1,1}, P_{1,3}, P_{1,4}, P_{1,5}, P_{1,2}\}$ and $\{P_{2,1}, P_{2,2}, P_{2,4}, P_{2,3}\}$, respectively (Fig. 2 (b)). Suppose that packets from a flow will arrive at their destination following the same order as they depart from the OLT, then we will get the following ACK sequences: $\{A_{1,2}, A_{1,2}, A_{1,2}, A_{1,2}, A_{1,6}\}$ and $\{A_{2,2}, A_{2,3}, A_{2,3}, A_{2,5}\}$ from the perspective of the OLT. Thus, it is very likely that the TCP sender of flow 1 will receive three duplicate ACKs ($A_{1,2}$), and then will perform fast retransmission and fast recovery. This will cause multiplicative decrease (and additive increase) in TCP's congestion window size. However, this is a spurious segment retransmission and keeping congestion window small is unnecessary. In fact, the network has not been in congestion condition yet. Therefore, RR scheduling at the OLT can not utilize bandwidth efficiently and may reduce throughput significantly.

We observe that if the OLT can estimate the number of duplicate ACKs (denote by $dack'$) which may be caused by scheduling the head-of-line (HOL) packet of each flow and schedule the HOL packet of the flow with the lowest $dack'$, it increases the chance for the OLT to resequence packets for flows with higher $dack'$. Using such an observation, for the example given in Fig. 2, the OLT will postpone the departure of packet $P_{1,3}$ and sends $P_{2,2}$ instead since flow 2's $dack'$ (0) is lower than flow 1's $dack'(1)$, as shown in Fig. 2 (c). After sending 4 packets, packet $P_{1,2}$ has already arrived at the OLT's buffer (Fig. 2 (d)). Thus, we transmit $P_{1,2}$ immediately since flow 1's $dack'$ (0) is lower than flow 2's $dack'$ (1)(Fig. 2 (e)). After sending all packets according to the order given in Fig. 2 (f)), we expect to receive the following ACK sequences $\{A_{1,2}, A_{1,2}, A_{1,4}, A_{1,5}, A_{1,6}\}$ for flow 1 and $\{A_{2,2}, A_{2,3}, A_{2,4}, A_{2,5}\}$ for flow 2. All packets are sent out within 9 time slots with only one duplicate ACK ($A_{1,2}$). It is inadequate to trigger fast retransmission and fast recovery. This motivation example shows that a good packet scheduling algorithm at the OLT can resequence packets to reduce the effect of packet reordering caused by the multi-path routing in wireless subnetwork and assure possible in-order arrivals at destinations. This paper focuses on designing such a packet scheduling algorithm at the OLT.

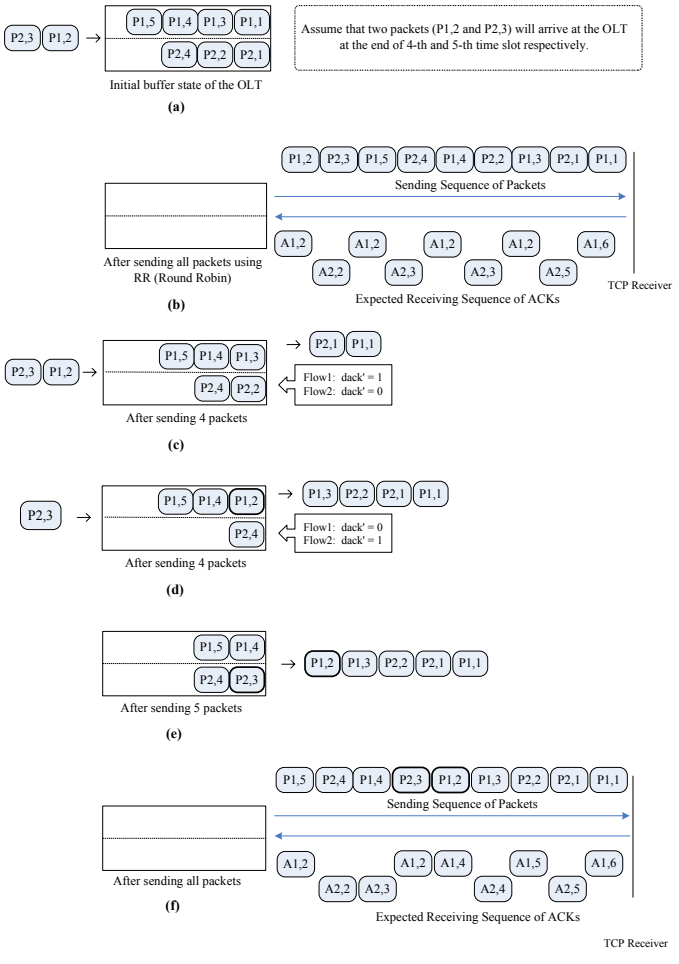

Fig. 2. An illustration of packet scheduling at the OLT and it's impact on TCP performance

3 Problem Description

Suppose that a pool of packets arrive dynamically from F different flows with packet reordering at the OLT where the OLT maintains a queue for each flow. Suppose that the time is partitioned into equal time slots where in each time slot at most one packet can be sent out from the OLT to the Internet. At each time slot, the OLT needs to determine which flow (queue)'s packet should be sent and which packet from that selected flow (queue) should be sent. As duplicate ACKs (three dupacks) may trigger the fast retransmission and fast recovery, which will cause multiplicative decrease (and additive increase) in TCP's congestion

window size (*cwnd*), it is important to avoid triggering three dupacks when we schedule packets. To achieve such a goal, at each time slot, a flow should be selected for transmission if it will most unlikely reduce the sender's *cwnd* and when a flow is selected for transmission, the packet with the smallest sequence number in the queue should be scheduled. In other words, we can implement a min-heap queue for each flow and assure that the HOL packet of each flow has the smallest sequence number. On the other hand, fairness among flows shall also been considered when we schedule packets.

In this section, we first analyze the potential change of the sender's *cwnd* if a flow's HOL packet is scheduled for transmission. Suppose that $P_{i,j}$ is the head packet of queue i at time t. Let $dack'_i(t)$ denote the total number of dupacks which may be caused by sending $P_{i,j}$. The impact of $dack'_i(t)$ on the change of the sender's *cwnd* can be summarized as follows:

- In-order delivery,
  - case 1: $dack'_i(t) = 0$ where $P_{i,j}$ is the expected packet. To transmit $P_{i,j}$ will increase the sender's *cwnd* and allow the receiver to generate cumulative ACKs.
- Out-of-order delivery,
  - case 2: $dack'_i(t) = 1$. To transmit $P_{i,j}$ will cause one dupack to be sent to the sender, which, however, will not cause any change on the sender's *cwnd*,
  - case 3: $dack'_i(t) = 2$, Same to case 2.
  - case 4: $dack'_i(t) = 3$. To transmit $P_{i,j}$ will cause one dupack to be sent to the sender, which will consequently trigger the fast retransmission and cause the reduction on the sender's *cwnd*;
  - case 5: $dack'_i(t) > 3$, TCP sender is now at the stage of fast recovery. To send $P_{i,j}$ will cause one dupack to be sent to the sender and increase the sender's *cwnd*.

As the sender's *cwnd* in both case 1 and case 5 will be increased, such a flow i should have the highest priority to be scheduled for transmission at time slot t. Case 4 will cause the reduction on the sender's *cwnd*, thus, such a flow should be scheduled later with the hope that the expected packet will arrive at the queue. Case 2 and case 3 will not cause the immediate change of the sender's *cwnd* and can be assigned with the priority between the highest priority and the lowest priority.

Let $p_i(t)$ be the priority of sending $P_{i,j}$, which is defined as follows:

$$p_i(t) = \begin{cases} 2 & \text{if } dack'_i(t) = 0 \text{ or } dack'_i(t) > 3 \\ 1 & \text{if } dack'_i(t) = 1 \text{ or } dack'_i(t) = 2 \\ 0 & \text{if } dack'_i(t) = 3 \end{cases} \tag{1}$$

In order to mitigate the effect of packet reordering, the HOL packet of queue i^* with the maximum $p_{i^*}(t)$ will be scheduled, which enhances the chance of other queues with lower priority value to be resequenced.

Apart from dupack, we also need to consider fairness among flows. Let $swait_i(t)$ be the time elapsed since last time when queue i is scheduled for transmission. If

queue i not backlogged, we set $swait_i(t) = 0$. Thus, for the sake of fairness, the queue with highest $swait_i(t)$ should be scheduled.

Let $f_i(t)$ be the scheduling weight of flow i at time slot t. Considering both priority from the perspective of potential change on the sender's *cwnd* and fairness, we define the following total order among flows: $f_i(t) \succ f_j(t)$ iff $p_i(t) > p_j(t)$ or $p_i(t) = p_j(t)$ and $swait_i(t) > swait_j(t)$.

At the beginning of scheduling, $p_i(0) = 2$ and $swait_i(0) = 0$ for $i \in F$ (F is the set of active flows at the OLT), indicating that no dupack is produced and the OLT doesn't have any packet waiting for transmission initially. Every time when a packet is to be transmitted, the scheduler schedules the head packet of the backlogged flow i^* with the maximum lexicographical order of $f_{i^*}(t)$, i.e.,

$$i^* = argmax_{i \in F} f_i(t) \tag{2}$$

Suppose that flow i^* is scheduled for tranmission at time slot t, if $p_{i^*}(t) = 0$, to send the HOL packet of flow i^* may trigger three dupacks, thus the sender's *cwnd* will be reduced. In such a case, we may prefer to delay the transmission at current time slot and wait for the expected packet. Note that in the next time slot, with the new arrival packets to each flow, a new flow may be selected for transmission or the current flow will be selected for transmission again. On the other hand, if the expected packet is lost, a TCP timeout will eventually be triggered. In such a case, it is more desirable that we send the packet immediately to trigger fast retransmission. To resolve such a dilemma, the OLT needs to have an estimation on whether the expected packet of flow i^* is lost or not. If enough time (more than a predefined threshold) has elapsed since flow i^* is scheduled for transmission last time, the OLT can regard the expected packet as lost packet and send the current HOL packet immediately. Such a threshold is denoted as max_hold_i for each flow i where max_hold_i can be determined by the delay difference of multiple paths in the wireless subnetwork.

4 Data Structure and Packet Scheduling Algorithm

The most challenging part of making the scheduling decision is to maintain the number of dupacks for each queue. In this section, we first use some examples to illustrate what information must be maintained for each queue in order to derive the number of dupacks. We then use a Finite State Machine (FSM) to formally describe the state change at each queue. We finally present our scheduling algorithm.

Assume that queue i has sent out $P_{i,1}$ and $dack_i(t) = 0$. The OLT is now expecting $P_{i,2}$. When packet $P_{i,3}$ becomes the HOL packet, suppose queue i is scheduled for transmission and packet $P_{i,3}$ is sent out, $dack_i(t)$ becomes 1. In the next time slot, suppose that packet $P_{i,5}$ becomes the head of queue i. In this case, the current packet's *seqno* is even higher than the highest *seqno* sent so far for this queue (3 in this case). Suppose that this packet is sent out immediately, $dack_i(t)$ becomes 2. In the next time slot, if the expected packet $P_{i,2}$ arrives at the OLT and is sent out immediately. The expected *seqno* will be 4 as packet

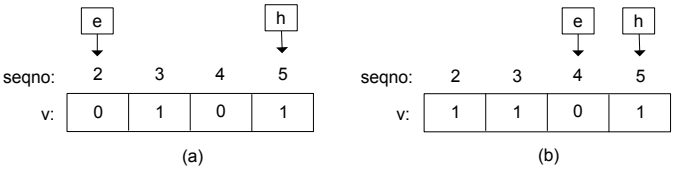

Fig. 3. A bit_vector for recording the OOD packet delivering

$P_{i,3}$ has been sent out from the OLT. This situation indicates that in order to update the expected *seqno*, we have to record which packets have been sent among the packets with *seqno* between current expected *seqno* and the highest *seqno*. We use a bit_vector v to record which packets between the expected one and the highest *seqno* have been sent out (Fig 3) where $v[i] = 0$ indicates that the corresponding packet has not been sent out from the OLT, and vice versa. As shown in (Fig 3 (a)), the expected packet's *seqno* is 2, the highest packet's *seqno* sent so far is 5, packet 3 has been sent out. Thus, when the expected packet $P_{i,2}$ arrives, we can immediately obtain that the next expected *seqno* is 4 (Fig 3 (b)).

This shows that, in order to maintain the number of dupacks, we need the information of the expected *seqno*, the highest *seqno* sent so far, and a vector for recording information about out-of-order packet delivering.

The above example only shows how to update the number of dupacks after the HOL packet of a queue is committed for transmission. However, in order to decide which queue's HOL packet should be transmitted at a time slot, we need to compute the number of dupacks *if* the HOL packet of a queue is scheduled for transmission. In other words, besides maintaining the number of *committed* dupacks from the perspective of sent packets, we also need to estimate the consequence if the current HOL packet is scheduled for transmission, which is the number of *potential* dupacks. For instance, as shown in Fig. 2 (c), after $P_{1,1}$ and $P_{2,1}$ were sent out, the number of *committed* dupacks for both queues is 0. Now, $P_{1,3}$ and $P_{2,2}$ become the head packet of each queue. If $P_{1,3}$ or $P_{2,2}$ is scheduled for transmission, the number of *potential* dupacks for each queue is 1 and 0, respectively.

We now formally present how to maintain *the expected seqno*, *the highest seqno*, *the number of committed dupacks*, and *the number of potential dupacks* for each queue. The OLT maintains a quadruplet $\{e, dack, dack', h\}$ and a variable $(swait)$ for each flow i, where e denotes the expected *seqno*, h denotes the highest *seqno* of packet which has been sent, *dack* denotes the number of committed dupacks, $dack'$ denotes the number of potential dupacks, and *swait* denotes the time elapsed since flow i is scheduled for transmission last time. Each queue is maintained as a min-heap queue where the HOL packet has the minimum *seqno*.

Initially, $\{e = 0, dack = 0, dack' = 0, h = -1\}$ and $swait = 0$ for each queue, which indicates that no packet has been sent for this flow and the packet with $seqno = 0$ is expected to be sent next. Whenever a packet enters a queue, min-heap insertion operation will be conducted at the queue. If the HOL packet of

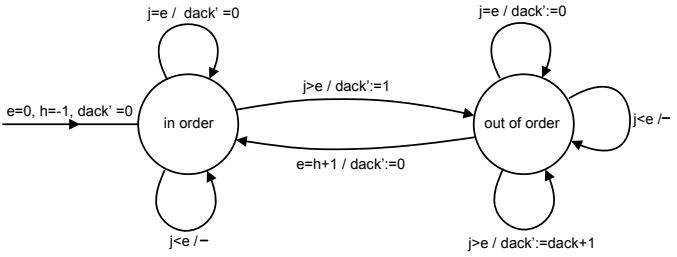

Fig. 4. State change of $dack'$ at each queue

the queue remains to be the same, no update is necessary. Suppose that the current packet becomes the HOL packet and its *seqno* is j. We need to see how $dack'$ will be changed if the HOL packet of this queue is scheduled next. We have the following cases:

- If there is in-order transmission which means $e = h + 1$,
  - if $j = e$ (or equivalently $j = h + 1$), this is still an in-order packet transmission, set $dack' = 0$;
  - if $j > e$, set $dack' = 1$, and bit_vector v is initialized for recording the sending information of OOD packets;
  - if $j < e$, it is a retransmitted packet and no update is necessary.
- If there is out-of-order transmission which means $e \leq h$,
  - if $j = e$, this is an in-order packet transmission, set $dack' = 0$;
  - if $j > e$, we need to update $dack'$ by $dack' = dack + 1$;
  - if $j < e$, it is a retransmitted packet and no update is necessary.

Such state change can be described using a Finite State Machine (FSM) as shown in Fig. 4. When a new packet enters a queue and becomes the HOL packet, state maintenance process will be triggered. When a packet from a queue with maximum $f_i(t)$ is scheduled for departure, we first make necessary update on e, v, $dack$ and h, then trigger the state maintenance for this queue since a new packet will become the head of the queue. The algorithm schedules packets so that the precedence constraint among packets are followed as much as possible. Such an algorithm is referred to as soft precedence constraint scheduling (*SPCS*) algorithm.

5 Performance Evaluation

In this section, we present our simulation results and compare our algorithm with other scheduling algorithms. Section 5.1 discusses the experimental setup for our simulation study. Simulation results are presented in Section 5.2.

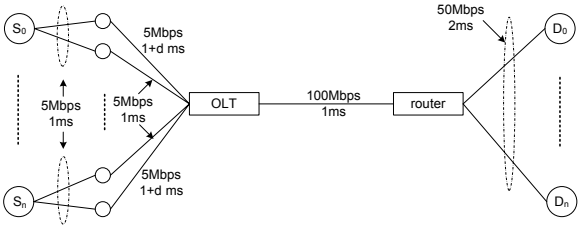

Fig. 5. Simulation Network Topology

5.1 Experimental Setup

We evaluate the performance of our algorithm using ns-2 (version 2.34) [18] running under ubuntu linux 9.04. Figure 5 shows the simulation topology. There are 10 wireless clients, each sending packets to the OLT through 2 paths in a round-robin fashion. Each path from the source to the OLT can provide 5 Mbps link capacity. The OLT is connected to the Internet through an Ethernet. As mentioned early, packets of a flow may arrive at the OLT through different paths. Due to different paths' delay, OOD packet arrivals will be produced. In order to simulate the delay difference between paths, for the two paths from each source to the OLT, one's delay is 1ms and the other is $1 + d$ ms where d varies from 0 to 100 ms. A larger d will introduce more variation in the path delay, thus increasing the degree of packet reordering.

We use TCP/Reno as the agent for TCP connections. TCP/Reno is chosen for its implementation of fast retransmission and fast recovery. The TCP window is set to be 50 segments and packet size is fixed to be 1000 bytes. A FTP application is started at $t = 0.05i$ s and is stopped at $t = 10 + 0.05i$ s at each wireless client s_i. In the simulation, max_hold is set to d.

5.2 Simulation Results

In this section, we compare the performance of our proposed *SPCS* algorithm with classical scheduling algorithms such as FIFO and DRR, and HPCS [19].

We use *goodput*, which is defined as the number of packets successfully received and acknowledged by the receiver, excluding retransmissions, as a performance metric to compare our proposed packet scheduling algorithm with other packet scheduling algorithms.

Firstly, the buffer size at the OLT is set to be large enough to ensure that no packet drops at the OLT. Fig. 6 shows the average goodput of all flows when d varies between 0 ms and 100 ms at intervals of 10 ms. From Fig. 6, we can see that a larger d does lead to lower goodput. We can also see that both our SPCS and HPCS outperform the other two classical scheduling algorithms.

Though the goodput metric of HPCS is similar to SPCS, Figure 7 shows that packets experience much longer queuing delay at the OLT in HPCS than other schedulers, which is not practical to be used as a resequencing algorithm at the

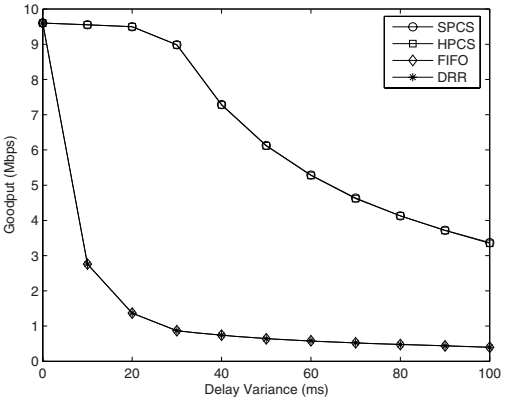

Fig. 6. Goodput vs. delay variance

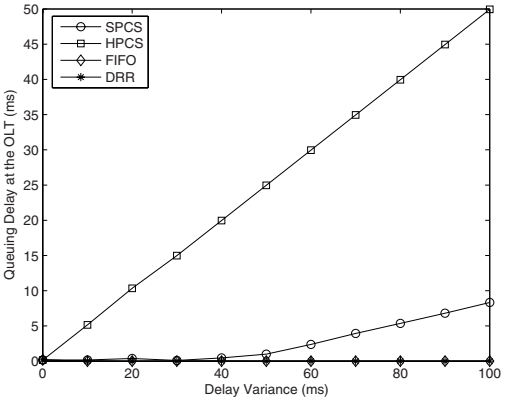

Fig. 7. Queueing delay at the OLT vs. delay variance

intermediate nodes. Our proposed SPCS scheduling algorithm will experience much less queuing delay at the OLT when the delay difference among multiple paths are moderate.

To verify fair bandwidth sharing, we use *min-max* ratio, $r_{min-max}$, as our fairness index, which is defined as follows: given a set of goodput $(x_1, x_2, ..., x_n)$, the following function assigns a fairness index to the set:

$$r_{min-max} = \frac{min_j\{x_i\}}{max_j\{x_i\}} = min_{i,j}\{\frac{x_i}{x_j}\} \tag{3}$$

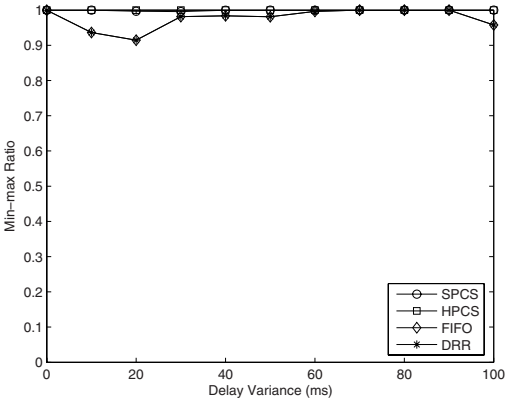

Fig. 8. Min-max ratio vs. delay variance

Fig. 8 shows the achieved min-max ratio of those scheduling algorithms when d changes from 0 ms to 100 ms. The results show that all compared algorithms can achieve the goal of providing fair bandwidth sharing. This validate the effectiveness of our fairness design criterion.

At last, we show the required buffer size at the OLT for each scheduler. We set d to 50 ms and simulate the goodput when the buffer size varies from 12.5 Kbytes to 250 Kbytes. Here, we adopt the shared buffer scheme, i.e., all flows share a common buffer pool in the OLT. When overflow happens, we use a pointer to drop the tail of each queue periodically. With the increase in buffer size, fewer packets are dropped. Hence, the goodput for upstream TCP traffic also increases. From Fig. 9, we can see that the goodputs of both SPCS and HPCS indeed increase with the buffer size while there is no change in FIFO and

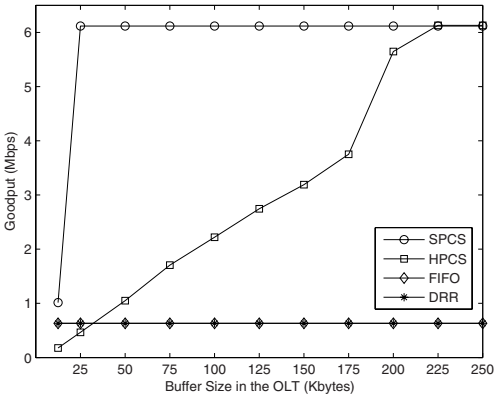

Fig. 9. Goodput vs. buffer size in the OLT

DRR. Fig. 9 also shows that SPCS only needs 25 Kbytes to behave well while HPCS requires as much as 225 Kbytes.

6 Conclusion

The integration of a PON and a WMN makes the OLT as a convergence node, which naturally makes it possible to resequence packets at the OLT before they are sent to the Internet. In this paper, we propose a scheduling algorithm at the OLT to resequence packets while providing fairness. Simulation results show that the proposed packet scheduling algorithm is efficient in reducing the effect of packet reordering, assuring fairness among different flows and reducing the required buffer size in the OLT. Compared with other reordering-tolerant algorithms which need to modify TCP protocol at clients, our proposed work provides a scalable solution since resequencing is conducted in the access network at the sender's side. The proposed resequence scheduling can be used not only at the OLT, but also at edge routers where per-flow packet scheduling is possible.

References

1. Sarkar, S., Dixit, S., Mukherjee, B.: Hybrid Wireless-Optical Broadband-Access Network (WOBAN): A Review of Relevant Challenges. Journal of Lightwave Technology 25(11), 3329–3340 (2007)
2. Mueller, S., Tsang, R., Ghosal, D.: Multipath Routing in Mobile Ad Hoc Networks: Issues and Challenges. In: Calzarossa, M.C.i., Gelenbe, E. (eds.). LNCS. Springer, Heidelberg (2004)
3. Liu, J.: On a Self-Organizing Multipath Routing Protocol in Mobile Wireless Networks. Journal of Network and Systems Management 14(1), 103–126 (2006)
4. Maxemchuk, N.F.: Dispersity routing in high-speed networks. Computer Networks and ISDN Systems 25(6), 645–661 (1993)
5. Pham, P.P., Perreau, S.: Performance analysis of reactive shortest path and multipath routing mechanism with load balance. In: IEEE INFOCOM 2003, vol. 1, pp. 251–259 (2003)
6. Lee, Y., Park, I., Choi, Y.: Improving TCP Performance in Multipath Packet Forwarding Networks. Journal of Communications and Networks 4(2) (2002)
7. Floyd, S.: A report on recent developments in TCP congestion control. IEEE Communications Magazine 39(4), 84–90 (2001)
8. Leung, K.-C., Li, V.O.K.: Flow Assignment and Packet Scheduling for Multipath Routing. Journal of Communications and Networks 5(3) (2003)
9. Rao, N.S.V., Batsell, S.G.: QoS routing via multiple paths using bandwidth reservation. In: IEEE INFOCOM 1998, pp. 11–18 (1998)
10. Kandula, S., Katabi, D., Sinha, S., Berger, A.: Dynamic Load Balancing Without Packet Reordering. ACM SIGCOMM Computer Comm. Rev. 37(2) (2007)
11. Zhang, M., Karp, B., Floyd, S., Peterson, L.: RR-TCP: A Reordering-Robust TCP with DSACK. In: Proc. IEEE ICNP 2003 (2003)
12. Lee, Y., Park, I., Choi, Y.: Improving TCP performance in multipath packet forwarding networks. J. Comm. and Networks 4(2), 148–157 (2002)

13. Sathiaseelan, A., Radzik, T.: Improving the Performance of TCP in the Case of Packet Reordering. In: Mammeri, Z., Lorenz, P. (eds.) HSNMC 2004. LNCS, vol. 3079, pp. 63–73. Springer, Heidelberg (2004)
14. Leung, K.-C., Li, V.O.K., Yang, D.: An Overview of Packet Reordering in Transmission Control Protocol (TCP): Problems, Solutions, and Challenges. IEEE Transactions on Parallel and Distributed Systems 18(4) (2007)
15. Xia, Y., Tse, D.: Analysis on packet resequencing for reliable network protocols. In: Proceedings of INFOCOM (2003)
16. Baccelli, F., Gelenbe, E., Plateau, B.: An end-to-end approach to the resequencing problem. Journal of ACM 31(3), 474–485 (1984)
17. Kim, M., Park, H.: Superposed Multiple QoS Guarantee in Optical Access Networks. In: Chung, C.-W., Kim, C.-k., Kim, W., Ling, T.-W., Song, K.-H. (eds.) HSI 2003. LNCS, vol. 2713, pp. 619–625. Springer, Heidelberg (2003)
18. `http://www.isi.edu/nsnam/ns`
19. Lane, J.R., Nakao, A.: Best-Effort Network Layer Packet Reordering in Support of Multipath Overlay Packet Dispersion. In: IEEE GLOBECOM 2008 (2008)