Goodput Improvement for Multipath TCP by Congestion Window Adaptation in Multi-Radio Devices

Dizhi Zhou, Wei Song, *Member, IEEE*

Faculty of Computer Science
University of New Brunswick, Fredericton, Canada
Emails: dizhiz@acm.org, wsong@unb.ca

Minghui Shi

Communications Research Centre
3701 Carling Avenue, Ottawa, Canada
Email: minghui.shi@crc.gc.ca

***Abstract*—Multipath Transport Control Protocol (MPTCP) has been standardized by Internet Engineering Task Force (IETF) to support simultaneous delivery of transport control protocol (TCP) packets over multiple interfaces of multi-radio mobile devices. Although MPTCP provides an efficient solution to aggregate the available bandwidth of multiple paths, the goodput of MPTCP is usually far lower than the aggregate throughput due to out-of-order received packets. One key reason for the out-of-order issue is the large variation of end-to-end delay for multiple paths over wireless channels. In this paper, we propose a congestion window adaption algorithm for the MPTCP source (referred to as *CWA-MPTCP*), which dynamically adjusts the congestion window for each TCP subflow so as to mitigate the variation of end-to-end path delay. We consider typical multipath transmission scenarios over wireless links, as well as a cooperative multi-hop wireless network with multiple relays. For wired paths with stable end-to-end delay, we further develop a proactive scheduling algorithm to determine the packet sending sequence to each path. This algorithm effectively reduces the out-of-order packets by predicting the receiving sequence. Experiments are conducted to evaluate the goodput performance of the two enhancements to MPTCP. Significant performance gain is achieved in terms of goodput, while the receive buffer requirement is minimized.**

***Index Terms*—MPTCP, multipath transmission, goodput, congestion window, out-of-order problem.**

I. Introduction

In recent years, mainstream mobile devices in the market are equipped with multiple radio interfaces. Multi-radio mobile devices usually have at least one built-in wireless wide area network (WWAN) interface, such as high speed packet access (HSPA). Also, the multi-radio mobile devices often have one or more short-range wireless network interfaces, such as Wi-Fi and Bluetooth. This offers a good opportunity to explore several interfaces for multipath transmission, so as to aggregate the bandwidth among multiple wireless links and further improve the quality of service (QoS) for bandwidth-intensive applications, such as video streaming and video conference.

The transport control protocol (TCP) is the *de facto* standard for the transport layer. Nonetheless, there is a critical problem with TCP over multiple paths for multi-radio devices, i.e., a high level of out-of-order packets. In traditional TCP, such as TCP Reno and selective acknowledgment (SACK), the source node halves its congestion window once three duplicate acknowledgments (ACK) are received from the sink node. That is, three duplicate ACKs are viewed as an indicator of packet loss in transmission. In a multipath transmission scenario, because the round-trip time (RTT) of each path varies, there is a high probability that packets with lower sequence numbers sent over a slower path arrive at the sink later than packets with higher sequence numbers sent over a faster path. As a result, the sink node receives out-of-order packets and then returns duplicate ACKs, which is misinterpreted by the source as packet loss. Then, the source reduces its congestion window and enters fast retransmit and recovery stage. This behavior jeopardizes the efficiency of TCP transmission because the sending window can be mistakenly set to a small value [1].

Multipath transport control protocol (MPTCP) is standardized by Internet Engineering Task Force (IETF) in 2011 [2]. A typical scenario of MPTCP in wireless networks is shown in Fig. 1. MPTCP runs in multi-homed mobile devices to simultaneously deliver TCP packets over multiple paths and pool the available bandwidth together. Although MPTCP improves the available throughput for the upper layer, there is still another unresolved issue caused by out-of-order packets. Throughput indeed represents the overall receiving capacity of successful packet delivery over multiple paths. Nonetheless, it is goodput that reflects the real application-level throughput, which is the amount of useful data available to the receiver application per time unit. Specifically, in-order packets received at the transport layer can be forwarded to the application layer and counted for goodput. Some most recent work in 2012 tries to improve goodput for MPTCP, by using network coding [3] and packet retransmission over fast path [4]. However, these studies only show the average goodput improvement over a long term. In fact, stable goodput with minimal variation is preferable for QoS assurance to real-time applications.

In this paper, we conduct extensive experiments to examine the goodput performance of MPTCP. An interesting observation is that the MPTCP goodput is near optimal (approaching the upper bound of aggregate throughput) when the end-to-end delays of two transmission paths are very close. The goodput variation is not directly related to the absolute delay values of multiple paths. Based on such observations, we propose a congestion window adaption algorithm for MPTCP, referred to as *CWA-MPTCP*, in which the MPTCP source dynamically adjusts the congestion window of each TCP subflow so as to maintain similar end-to-end delays over

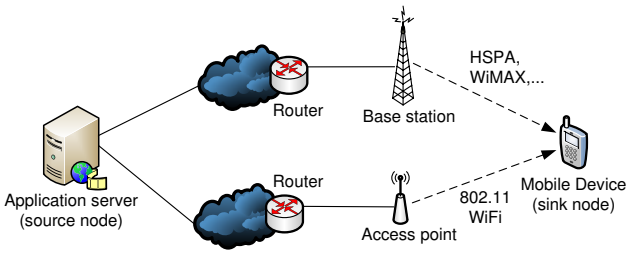


Fig. 1. Multipath scenario in wireless networks.

multiple paths. The CWA-MPTCP solution is evaluated for a typical wireless multipath scenario with multi-radio devices, as well as a wireless cooperative scenario with multi-hop relays. Moreover, in a wired environment, since the path delay is more stable in comparison with that of wireless links, the goodput performance of MPTCP can be further enhanced with a proactive packet scheduler. The simulation results demonstrate significant goodput improvement and greatly reduced receive buffer requirement for the sink node.

The rest of this paper is organized as follows. An overview of MPTCP is given in Section II. In Section III, we introduce the proposed extensions to MPTCP for goodput improvement. Experiment results are presented in Section IV. Related work is outlined in Section V, followed by conclusions in Section VI.

II. Overview of MPTCP

As shown in Fig. 2, MPTCP loosely splits the transport layer into two sublayers, namely, MPTCP and subflow TCP. Based on this architecture, MPTCP can be easily implemented within current network stack. As seen, subflow TCP runs on each path independently and reuses most functions of regular TCP. The main difference between subflow TCP and regular TCP is that congestion control on each path is delegated to MPTCP sublayer [5]. Although each subflow TCP maintains a congestion window at the source (sender), the congestion window is updated by a coupled congestion control algorithm which aims to balance the traffic load on each path and improve throughput without jeopardizing regular TCP users.

MPTCP sublayer is responsible for coordinating data packets on multiple paths, such as reordering packets received from each path at the sink, scheduling packets toward each path at the source, and balancing the congestion window of each subflow TCP. In addition to the aforementioned congestion control algorithm, another main function of MPTCP is packet reordering for multiple paths. As each TCP subflow maintains an independent sequence number space, the sink may receive two packets of the same sequence number. Further, packets received at the sink can be out-of-order because of mismatched round-trip time (RTT) of multiple paths. Therefore, the source needs to tell the sink how to reassemble the data forwarded to the application. MPTCP solves this problem by using two levels of sequence numbers. First, the sequence number for TCP subflow is referred to as *subflow sequence number* (SSN), which is similar to the one in regular TCP. The subflow sequence number independently works within each subflow

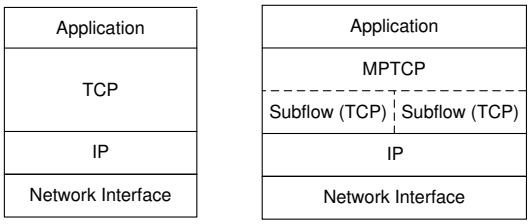


Fig. 2. Network protocol stack with MPTCP.

and ensures that data packets of each subflow are successfully transmitted to the sink in order. The sequence number at the MPTCP level is called *data sequence number* (DSN). Each packet received at the sink has a unique DSN no matter which path it is sent over. Hence, the sink can easily sequence and reassemble packets from different paths by DSN.

III. Goodput Improvement for MPTCP

A. Problem Analysis

As discussed Section II, MPTCP can provide aggregate bandwidth to the application by pooling network resources of multiple paths together. In this work, we focus on another important performance metric, i.e., *goodput*. In particular, we define the goodput of MPTCP as the data throughput of in-order packets forwarded by MPTCP to the application layer. Intuitively, we have

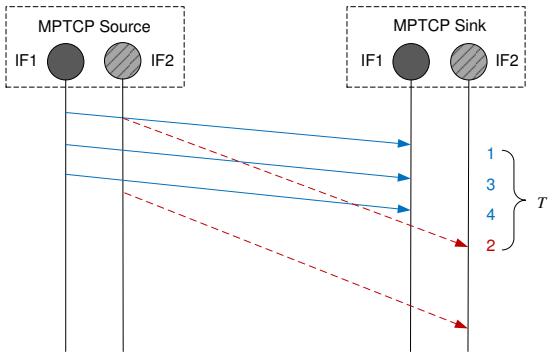
$$Goodput = \frac{\text{Size of } N \text{ in-order packets}}{\text{Total receiving time of } N \text{ packets}}. \quad (1)$$

Next, we examine two special scenarios of MPTCP so as to find out the primary factors affecting goodput performance. Suppose that there are two available paths. Let τ_i denote the packet sending interval at the MPTCP source for path i, $i = 1, 2$. Assume that the throughput of path 1 is greater than that of path 2. Denoting the end-to-end delay of path i by d_i, we have $d_1 < d_2$. Consider a block of N packets with continuous DSN numbers, among which $N-1$ packets are received on path 1 and only 1 packet is from path 2. Such a block of data packets is referred to as an *in-order unit*. Let S and T denote the total size in the unit of maximum segment size (MSS) and the total receiving time of an in-order unit, respectively. Then, we can evaluate the goodput by $G = S/T$.

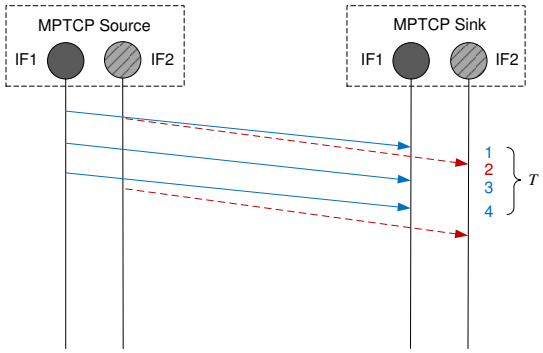
Consider two special cases illustrated in Fig. 3. The in-order unit comprises 4 packets of DSN numbers $1, 2, 3,$ and 4. Suppose that packet 1 and packet 2 are sent at the same time to path 1 and path 2, respectively. Fig. 3(a) shows the case with $\Delta D \triangleq |d_2 - d_1| > \tau_1$. We can easily obtain $T = \Delta D$ and the goodput, given by

$$G = \frac{S}{T} = \frac{\tau_2/\tau_1 + 1}{\triangle D}. \quad (2)$$

Fig. 3(b) shows another special scenario with $\Delta D \leq \tau_1$. In this case, the MPTCP sink needs less time to receive all packets within the in-order unit. Here, the total time to receive all N packets of the in-order unit is just the time for path 1 to receive



(a) General case with $\Delta D > \tau_1$



(b) Near optimal case with $\Delta D \leq \tau_1$

Fig. 3. Special cases with two transmission paths for goodput analysis.

all $N-1$ packets sent over it. Obviously,

$$G = \frac{S}{T} = \frac{\tau_2/\tau_1 + 1}{\tau_2}. \tag{3}$$

Actually, Eq. (3) is also the aggregate throughput (denoted by Υ) over two paths. That is,

$$\Upsilon = \frac{1}{\tau_1} + \frac{1}{\tau_2}. \tag{4}$$

This observation implies that goodput is inversely proportional to the end-to-end path delay difference ΔD. The larger the delay difference between two paths, the longer the time that MPTCP sink needs to receive all packets within the in-order unit, and the smaller the achievable goodput. To approach the upper bound of goodput, which is the aggregate throughput of all available paths, we need to minimize the end-to-end delay difference among transmission paths without jeopardizing the aggregate throughput.

B. Congestion Window Adaptation

In regular TCP, the TCP source node maintains a congestion window to control the maximum amount of packets to send at the current time. Once triple duplicate ACKs are received by the source, it is interpreted as an indicator of packet loss and the source node halves its congestion window to reduce the traffic load toward the transmission path. In MPTCP, each TCP subflow maintains its own congestion window and triggers a decrease of the congestion window by receiving duplicate ACKs. In contrast, the increase of the congestion windows of all subflows is controlled by a coupled algorithm at the MPTCP flow level. This congestion window control algorithm can aggregate the available bandwidth of each path and prevent a MPTCP source from taking up too much resource to guarantee TCP friendliness. In this congestion control algorithm, the only reason to decrease the congestion window is packet loss indicated by duplicate ACKs. Consequently, the congestion window of each path may greatly differ from each other and lead to a large path delay difference, which is detrimental to the goodput performance.

Based on the design principle in Section III-A, we propose the congestion window adaptation (CWA) given in Algorithm 1 to improve MPTCP goodput. In CWA-MPTCP, the end-to-end delays of multiple paths are monitored. Here, we define the *delay ratio* as the ratio of the maximum path delay over the minimum path delay. When a large delay ratio is detected, the source node proportionally decreases its congestion window although there is no packet loss indicated by duplicate ACKs. The main purpose is to minimize the path delay difference ΔD in order to increase the goodput. On the other hand, the increase of all subflow congestion windows follows the algorithm in original MPTCP.

Algorithm 1 Congestion Window Adaptation.

if $\theta_{\min} \leq \theta \leq \theta_{\max}$ **then** // High delay ratio detected
  // Select path i for $cwnd$ adaptation
  $i = \arg\max_p$(end-to-end delay of path p)
  // Adaptation counter does not exceed maximum limit
  if $count_i < m$ **then**
    // Decrease congestion window of path i
    $cwnd_i \leftarrow cwnd_i/\theta$
    if $ssthresh_i > cwnd_i$ **then**
      $ssthresh_i = cwnd_i$
    end if
    $count_i \leftarrow count_i + 1$
  else
    // Reset adaptation counter
    $count_i = 0$
  end if
end if

As seen in Algorithm 1, the congestion window adaptation is triggered when the delay ratio θ is within a certain range $[\theta_{\min}, \theta_{\max}]$. The congestion window (denoted by $cwnd_i$) of the path i with the maximum delay is decreased proportionally to the delay ratio θ. This is because a larger delay ratio indicates that the high-delay path is overloaded. Its congestion window needs to be decreased to relieve traffic and reduce path delay. Here, $\theta_{\max}$ is introduced to avoid over-blocking slow path and severely jeopardizing aggregate throughput. Meanwhile, the TCP slow start threshold ($ssthresh_i$) is updated with the new $cwnd_i$ if $ssthresh_i > cwnd_i$. Otherwise, $cwnd_i$ will be recovered quickly with the slow start procedure (i.e., $cwnd_i$ is linearly increased by 1 for each successful ACK received at the source). As a consequence, it would be hard

to guarantee that the congestion window of the slow path is decreased for sufficient time to reduce the end-to-end delay.

Although the $cwnd$ adaptation can reduce the end-to-end delay variation of multipath paths, it is not guaranteed that all paths maintain similar delays as the ideal case illustrated in Fig. 3(b). This is due to a variety of reasons in addition to the traffic load over the paths that affect the end-to-end delay, such as transmission, processing, and queueing delays at routers, base stations, and intermediate nodes between communication peers. The link-layer interference and retransmission over wireless channels also result in delay variation. The path delay variation can be reduced by decreasing the congestion window of the slow path and relieving its carried traffic load. Nonetheless, the transport-layer control itself cannot completely eliminate the path delay variation. Therefore, we introduce the parameter $count_i$ to restrict the number of continuous reductions of congestion window for a single path i by m, which is the maximum adaptation limit (e.g., $m = 3$ for the experiments in Section IV). As such, we can avoid severe throughput degradation on an individual path.

After the $cwnd$ of a high-delay path is reduced according to Algorithm 1, the corresponding TCP subflow is blocked from sending more packets, because of the gap between the original $cwnd$ and the adapted new one, i.e., $(cwnd_i - cwnd_i/\theta)$. The TCP subflow is blocked since the highest acknowledged DSN plus the adapted smaller $cwnd$ becomes less than the highest DSN of packets that are sent to the sink node. This subflow is then blocked for a period ΔT, given by

$$\Delta T = (cwnd_i - cwnd_i/\theta) \times \tau_i. \quad (5)$$

For instance, when $1 \leq \theta \leq 3$, $\tau_i = 5$ ms, and $cwnd_i = 100$ packets, ΔT ranges from 170 ms to 340 ms. During this short period, although one slow path is blocked and the overall throughput slightly decreases, more significant performance gain is achieved for goodput.

C. Proactive Scheduler for Wired Multipath Links

The above congestion window adaptation algorithm aims to minimize the end-to-end delay variation among multiple paths. Nonetheless, it is hard to guarantee close path delay due to various factors at different layers. The path delay of wired links is more stable than that of wireless links. Taking a closer examination on the scenario in Fig. 3(a), we can further complement CWA-MPTCP with a proactive scheduler (Pro) for wired multipath links, in case that the path delay difference is still relatively large even when congestion window adaptation is activated.

In Fig. 3(a), it is assumed that $\tau_2 = 3 \times \tau_1$ and $\Delta D = 3 \times \tau_1$. Suppose that the MPTCP source sends packets 1 and 2 simultaneously. Since $\Delta D/\tau_1 = 3$, the number of intermediate packets received between packets 1 and 2 is 2. That is, the MPTCP sink receives two more packets on path 1 before it receives packet 2 on path 2. This example demonstrates that, if the path delay difference ΔD is relatively stable, the source is able to predict the receiving packet order at the sink. Based on the above observation, we develop Algorithm 2 to schedule the sequence of packets sent to each path at the source. It requires relatively stable path delay to predict packet receiving order. Since the congestion window adaptation in Algorithm 1 can reduce path delay variation for goodput improvement, it is a complementary match to the proactive scheduler.

Algorithm 2 Proactive Packet Scheduler for Two Paths.

```
ACK for DSN packet received at MPTCP source
Path selection by MPTCP scheduler
if Slow path is selected then
    Δ_dsn = ΔD/τ_f    // τ_f is packet interval of fast path
else    // Fast path is selected
    Δ_dsn = 1
end if
// Initialize DSN of next packet with proper increment
S_dsn = F_dsn + Δ_dsn
// Check if the packet of DSN S_dsn has been sent
while S_dsn ∈ L_dsn do    // L_dsn is DSN list of sent packets
    S_dsn ← S_dsn + 1
end while
Insert S_dsn into L_dsn
if Fast path is selected then
    // Update the DSN of packets sent sequentially
    F_dsn = S_dsn
end if
```

To reduce out-of-order packets received at the sink, we can properly adjust the DSN increment value (denoted by Δ_{dsn}) for the next packet over the slow path and the fast path. Specifically, Δ_{dsn} for the slow path depends on the path delay difference ΔD and the packet sending interval of the fast path. Based on Δ_{dsn}, the DSN of the next packet sent over each path is determined according to Algorithm 2. Once the MPTCP source receives a data ACK from the sink, the MPTCP scheduler scans all paths and finds the first path having one packet free space in its sending window. If a fast path is chosen, the DSN of next packet to send is just $F_{dsn} + 1$, where F_{dsn} is the largest packet DSN sent over the fast path. For a slow path, the next packet to send takes an advanced DSN, i.e., $S_{dsn} = F_{dsn} + \Delta_{dsn}$, so as to better ensure in-order receiving sequence at the sink. In both cases, the source checks the DSN list of sent packets ($\mathcal{L}_{dsn}$) to avoid duplicate transmission.

IV. Experimental Results and Discussion

To evaluate the performance of the proposed solution in Section III, we extend the MPTCP model in NS-2 [6] with the congestion window adaption and proactive scheduler. The performance is evaluated in terms of goodput and receive buffer requirement for both wireless and wired scenarios. In the wireless case, the multi-radio sink node is equipped with two network interfaces and connected to a multi-channel base station over typical wireless links or cooperative multi-hop links through multiple relays. The detailed system parameters are given in Table I. The bandwidth for each path is an average and actually varying in the simulation. To evaluate

TABLE I

SYSTEM PARAMETERS FOR EXPERIMENTS.

Parameter	Sample value
Number of transmission paths	2
Radius of base station coverage	500 m
Average bandwidth on path 1	4.0 Mbit/s
Average bandwidth on path 2	2.0 Mbit/s
Application for testing	FTP
Minimum delay ratio $\theta_{\min}$	1.0
Maximum delay ratio $\theta_{\max}$	3.0
Receive buffer at sink	1.16 Mbit (100 MSS)
Receive buffer at relays	0.38 Mbit (33 MSS)

throughput and goodput performance over a long term, we simulate saturated data traffic by using an extremely large file. In the following experiments, the congestion window is adapted for every 0.5 seconds according to Algorithm 1. The goodput and receive buffer size are measured whenever the sink node receives 100 packets of continuous DSN. The receive buffer size is the required space to accommodate all incoming packets (including the out-of-order ones) until a block of 100 packets of continuous DSN are completely received.

In MPTCP, *receive window* specifies the maximum amount of data that can be received and buffered at the sink. That indicates the overall receiving capacity of all interfaces of the MPTCP sink. In this paper, we use another term *receive buffer* to represent the storage space required for the sink to accommodate K ($K = 100$ in the following experiments) packets of continuous DSN sequence numbers. Because packets arrived at the sink may be out of order due to multipath transmission, the actual space required can be much larger than the nominal size of K packets. Hence, the receive buffer implies the out-of-order extent of received packets through multiple paths.

A. Performance of CWA-MPTCP in Wireless Networks

Fig. 4 and Fig. 5 compare the goodput and receive buffer size when regular MPTCP and CWA-MPTCP are used. As seen in Fig. 4, the goodput of CWA-MPTCP almost approaches the aggregated throughput (6 Mbit/s), which is the upper bound of goodput achievable at the MPTCP sink. The goodput of original MPTCP is much lower and only close to the throughput of the slow path in some periods. This is because the large delay of the slow path introduces a large amount of out-of-order packets, which degrade the goodput substantially. In contrast, CWA-MPTCP can minimize the end-to-end delay difference between the two paths, so that the goodput is significantly improved and much more stable. Fig. 5 shows that the required receive buffer in CWA-MPTCP almost equals to the minimum receive buffer for 100 continuous packets, which is given by the black line. This observation indicates that CWA-MPTCP efficiently reduces the out-of-order packets received at the sink without jeopardizing the aggregate throughput.

Although MPTCP can aggregate the bandwidth over several wireless interfaces via multipath transmission, multiple

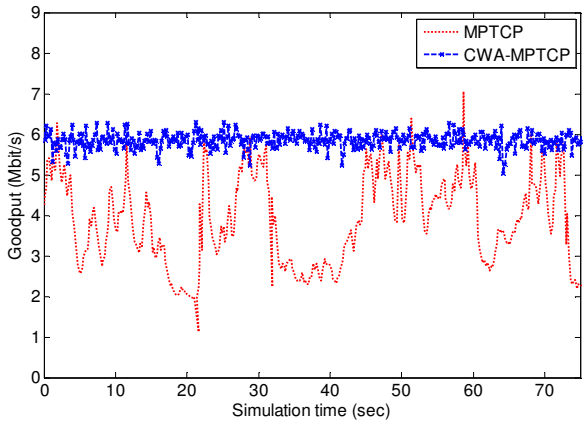


Fig. 4. Comparison of multipath goodput over wireless links.

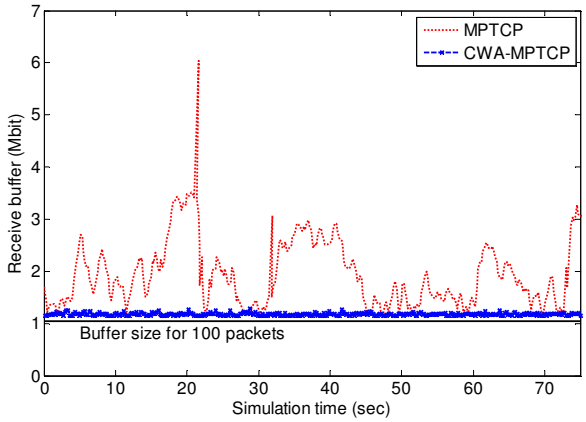


Fig. 5. Receive buffer size for multipath transmission over wireless links.

wireless accesses may not be always available. According to a recent survey in [7], 79% of smartphone users also own additional wireless devices such as laptops. Hence, multipath transmission can be enabled by exploiting multi-homed mobile devices in vicinity as relays as shown in Fig. 6. The relay nodes receive packets on behalf of the sink via their WWAN interfaces and then forward packets toward the sink via short-distance links (e.g., Wi-Fi or Bluetooth). Traditionally, multi-hop relay is supported at the link layer or the network layer. However, packet collisions may exist among multiple relays and the sink node with contention-based channel access. Such interference if not addressed properly limits the achievable performance of multipath transmission. We can extend the CWA-MPTCP solution by using relays at the transport layer [8]. Once an MPTCP connection is established, the sink node sends a DSN range, `(MIN_DSN, MAX_DSN)`, for the incoming data packets to all relays. This range gives the DSN of in-order packets that the sink expects to receive. If any packet within this DSN range is received by a relay, the relay accepts and forwards the packet to the sink. Otherwise, the relay node buffers the packet outside the range and only forwards the head packet once the buffer becomes full. As such, we can relax the buffer requirement for the sink node by sharing the receive buffer of relays.

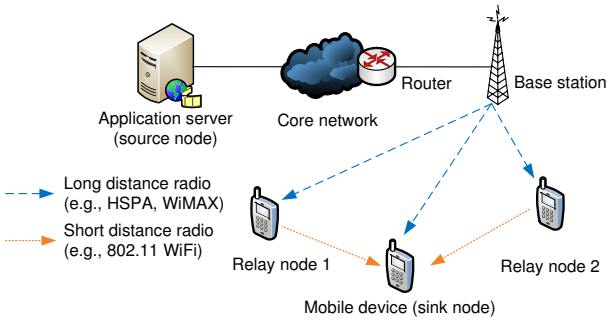


Fig. 6. Multipath with cooperative relays in a cooperative wireless network.

Fig. 7 and Fig. 8 present the simulation results of CWA-MPTCP in a cooperative multi-hop wireless scenario. Here, we compare the performance of regular MPTCP and CWA-MPTCP with relays implemented at the IP layer and TCP layer. As seen in Fig. 7, CWA-MPTCP greatly outperforms original MPTCP no matter whether IP relays or TCP relays are used. This observation demonstrates that the proposed congestion window adaptation algorithm works well in both a regular multipath scenario and a cooperative wireless scenario with multi-hop relays. This is because the adaptation algorithm effectively minimizes the path delay difference ΔD, which in turn significantly reduces the out-of-order packets to achieve a much higher goodput.

Additionally, relays implemented at the TCP layer further improves the goodput. With TCP relays, the TCP subflow connections end at the two relays rather than the two interfaces of the sink node. Hence, the source node is invisible to the additional delay caused by packet collisions among the relays and the sink node. As a result, a much smaller end-to-end delay is viewed at the source node for transport control. Thus, the goodput is benefited from the smaller path delay difference and the smaller absolute value of end-to-end delay. Similar performance gain is observed in Fig. 8 for the receive buffer requirement at the sink node.

B. Performance of MPTCP Extensions for Wired Links

Fig. 9 shows the goodput of original MPTCP and extended MPTCP with congestion window adaptation and proactive scheduler. The congestion window adaptation can also effectively mitigate the path delay difference ΔD. The delay variance of wired links is much smaller that that of wireless networks. This enables the implementation of the proactive scheduler to predict the receiving DSN sequence and rearrange the packet sending sequence to each path. The goodput of CWA-MPTCP with the proactive scheduler significantly improves the goodput and almost approaches the aggregate throughput (6 Mbit/s in this case), which is the upper bound of the achievable goodput.

In addition, a few spikes of dropped goodput are observed in Fig. 9. As discussed in Section III-B, when a large end-to-end delay ratio is detected between the fast path and the slow path, the *cwnd* of the slow path is decreased to block the corresponding TCP subflow, so as to minimize out-of-order packets received at the sink. As a consequence, the

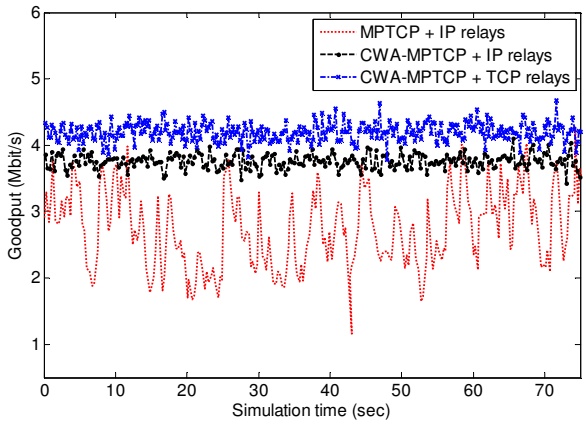


Fig. 7. Goodput with multi-hop relays in a cooperative wireless scenario.

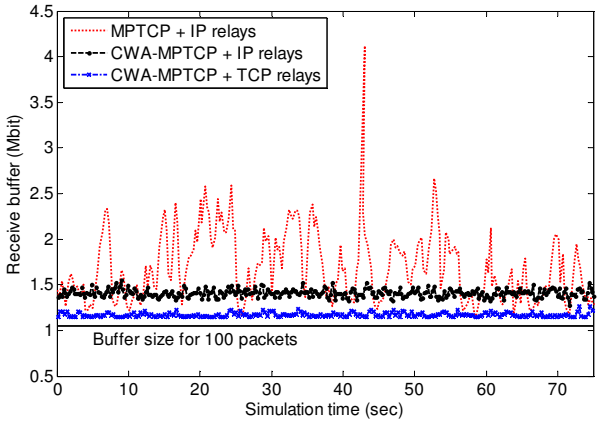


Fig. 8. Gain of receive buffer in a cooperative multipath scenario.

throughput of the slow path and the resulting goodput can be reduced for a very short period and recovered quickly. The property is acceptable since the overall goodput is significantly improved and much more stable than in regular MPTCP. The enhancement for receive buffer sharing is evidently observed Fig. 10. The sink node only needs a minimum receive buffer by effectively controlling out-of-order packets.

V. Related Work

Many multipath transmission architecture based on TCP have been proposed, such as pTCP [9], mTCP [10], and PRISM [11]. Such proposals modify the regular single-path TCP, aiming to aggregate bandwidth on multiple paths. Nonetheless, substantial changes to TCP stack are needed, which may limit wide deployment. There are also studies on enabling multiple TCP transmission for bandwidth-demanding applications (e.g., video streaming), so as to provide stable throughput throughout a session. In [12], TCP traffic is split at the IP layer and sent to multiple paths so as to enhance the performance of real-time streaming. Authors in [13] find the required aggregate TCP throughput (1.6 times the video bit rate) to support multipath live streaming. The approach in [14] adjusts the receiver window size to achieve the desired throughput using multiple TCP connections for multimedia

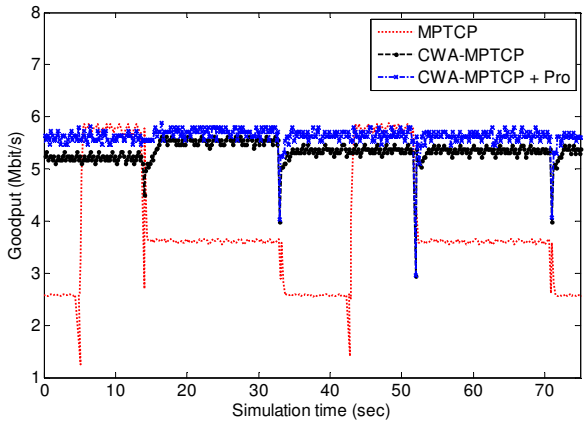


Fig. 9. Goodput performance over wired multipath links.

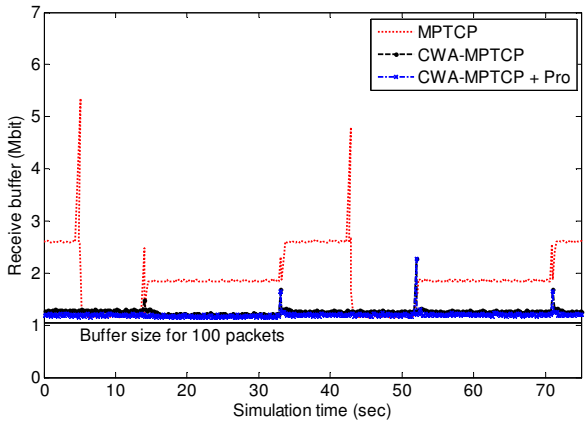


Fig. 10. Comparison of receive buffer size for wired multipath links.

streaming. Most of the previous solutions assume that the performance of video applications can be improved with a higher aggregate throughput. Although this is true in principle, out-of-order received packets may severely degrade the useful data throughput delivered by the transport layer to the application.

In 2012, there are some specific studies on the goodput performance of MPTCP. In [4], the goodput of MPTCP is enhanced by appropriately selecting the path for packet retransmission. When the slow path is blocked by a full receive buffer due to too many out-of-order packets, the source will retransmit packets toward the fast path. Also, the source simply halves the congestion window size of the slower path. As this scheme is only triggered when the receive buffer is full, it cannot handle goodput degradation in normal transmission stages. The schemes in [3] and [15] utilize network coding to recover packet loss at the sink and in turn increase the goodput. In such coding-based schemes, the source transmits the original data in one subflow and linear combinations of original data in the other subflow. Thus, the redundancy of network coding data is utilized to recover lost and delayed packets. However, these schemes require the support of network coding in both communication peers. The solutions in [3,4] mainly focus on the average goodput in a long term and neglect the goodput variation. Our approaches with congestion window adaptation and proactive scheduling significantly improves overall goodput to approach the upper bound of aggregate throughput. Meanwhile, the goodput variation is minimized to provide a stable goodput to the application.

VI. Conclusions

In this paper, we propose a congestion window adaptation algorithm (CWA-MPTCP) to enhance the goodput of MPTCP and decrease the receive buffer requirement for the sink node. By adapting the congestion window based on the end-to-end delay difference between paths, CWA-MPTCP effectively improves the goodput of multipath transmission over wireless and wired links. Moreover, CWA-MPTCP works well in cooperative wireless networks with multiple paths constructed via multi-hop relay nodes. Taking advantage of stable end-to-end delay of wired links, we further improve the goodput by using a proactive scheduler to arrange the packet sending sequence to each path. Simulation results demonstrate that our solutions achieve stable goodput with significant improvement and relax receive buffer requirement for the sink node.

Acknowledgement

This research was supported by Natural Sciences and Engineering Research Council (NSERC) of Canada.

References

[1] M. Zhang, B. Karp, S. Floyd, and L. Perterson, "RR-TCP: A reordering-robust TCP with DSACK," in *Proc. IEEE ICNP*, Nov. 2003.
[2] A. Ford, C. Raiciu, M. Handley, S. Barre, and J. Iyengar, "Architectural guidelines for multipath TCP development," IETF RFC 6182, Mar. 2011.
[3] M. Li, A. Lukyanenko, and Y. Cui, "Network coding based multipath TCP," in *Proc. IEEE INFOCOM Computer Communication Workshop*, Mar. 2012.
[4] C. Raiciu, C. Paasch, S. Barre, and A. Ford, "How hard can it be? Designing and implementing a deployable multipath TCP," in *Proc. USENIX NSDI*, Apr. 2012.
[5] C. Raiciu, M. Handley, and D. Wischik, "Coupled congestion control for multipath transport protocols," IETF RFC 6356, Oct. 2011.
[6] Google Code Project, "Multipath-TCP: Implement multipath TCP on NS-2," http://code.google.com/p/multipath-tcp/.
[7] K. Purcell, "E-reader ownership doubles in six months," Pew Research Center, 2011.
[8] D. Zhou, P. Ju, and W. Song, "Performance enhancement of multipath TCP with cooperative relays in a collaborative community," in *Proc. IEEE PIMRC*, Sep. 2012.
[9] H. Hsieh and R. Sivakumar, "pTCP: An end-to-end transport layer protocol for striped connections," in *Proc. IEEE ICNP*, Nov. 2002.
[10] M. Zhang, J. Lai, A. Krishnamurthy, L. Peterson, and R. Wang, "A transport layer approach for improving end-to-end performance and robustness using redundant paths," in *Proc. USENIX ATC*, Jun. 2004.
[11] K. Kim and K. Shin, "Improving TCP performance over wireless networks with collaborative multi-homed mobile hosts," in *Proc. ACM MOBISYS*, Jun. 2005.
[12] M. Tsai, N. Chilamkurti, J. Park, and C. Shieh, "Multi-path transmission control scheme combining bandwidth aggregation and packet scheduling for real-time streaming in multi-path environment," *IEEE IET Communications*, vol. 4, no. 8, pp. 937–945, May. 2009.
[13] B. Wang, W. Wei, and Z. Guo, "Multipath live streaming via TCP: Scheme, performance and benefits," *ACM Trans. Multi. Comput. Commun. and Appl.*, vol. 5, no. 3, Aug. 2009.
[14] S. Tullimas, T. Nguyen, and R. Edgecomb, "Multimedia streaming using multiple TCP connections," *ACM Trans. Multi. Comput. Commun. and Appl.*, vol. 4, no. 2, May 2008.
[15] Y. Cui, X. Wang, H. Wang, G. Pan, and Y. Wang, "FMTCP: A fountain code-based multipath transmission control protocol," in *Proc. IEEE ICDCS*, Jun. 2012.