M-TCP$^+$: Using Disconnection Feedback to Improve Performance of TCP in Wired/Wireless Networks

Modupe Omueti and Ljiljana Trajković
Simon Fraser University
Vancouver, BC, Canada
{momueti, ljilja}@cs.sfu.ca

Keywords: Heterogeneous wired/wireless networks, transmission control protocol (TCP), congestion control, disconnections, delayed acknowledgement.

Abstract
In this paper, we propose the M-TCP$^+$ algorithm for heterogeneous wired/wireless networks. The algorithm is a modification of M-TCP that was proposed for deployment in mobile cellular networks. It is recommended that Internet hosts enable the delayed acknowledgement (delayed ACK) option to maximize network bandwidth by reducing the number of ACKs sent to a TCP sender by a TCP receiver. The M-TCP$^+$ algorithm performs best when the TCP delayed ACK option is enabled. The algorithm relies on feedback sent from a wireless host in anticipation of disconnections. We compare the performance of the M-TCP$^+$ algorithm with the performance of M-TCP, TCP NewReno, and TCP SACK in both the absence and the presence of disconnections for a file transfer protocol (download) application. We also simulate network scenarios with traffic congestion. The M-TCP$^+$ algorithm performance is evaluated in terms of file download response time, goodput, and retransmission ratio, with and without the delayed ACK option. In scenarios without disconnections, the M-TCP$^+$ algorithm does not introduce significant processing delay. Furthermore, in scenarios with disconnections, the M-TCP$^+$ algorithm shows 2%–15% performance improvement.

1. INTRODUCTION

The Transmission Control Protocol (TCP) provides byte-stream delivery service for applications such as remote login, telnet, hyper-text transfer protocol (HTTP), and file transfer protocol (FTP) [1]. TCP carries over 90% of Internet traffic [2]. Heterogeneous wired/wireless networks are designed to support numerous wireless applications and a large number of users. They incorporate wireless infrastructure that provides connection to the Internet for wireless and mobile hosts [3].

An underlying assumption in TCP is that the network is wired and, hence, packet losses indicate congestion [1], [4]. TCP congestion avoidance and control algorithms [4], [5] were developed to reduce packet losses due to congestion, enabling TCP to perform well in wired networks. However, TCP performance degrades in wireless and heterogeneous wired/wireless networks because packet losses occur more frequently due to network characteristics [6] rather than traffic congestion. If packet losses are not caused by congestion, TCP throughput decreases because of invoked congestion avoidance and control algorithms.

TCP performance degrades in wireless networks even without client mobility [7]. Disconnections occur due to temporary link outages, signal path blockages, and interference. If disconnections occur during a large file transfer, the TCP retransmission time-out (RTO) mechanism employs the exponential backoff algorithm where retransmission interval is doubled for each retransmission. The initial RTO value is set to 2 s while the maximum RTO value cannot exceed 64 s. Before a connection closes, the accumulated time-out interval may vary from 120 s to 540 s depending on the TCP implementation, permitting between 6 and 13 retransmissions. The connection closes when the maximum number of retransmissions is reached. This is a common scenario when wireless clients download large files from servers connected to a wired backbone. If the disconnection occurs near completion of the file transfer, the entire transfer needs to be re-initiated.

The proposed M-TCP$^+$ algorithm is a modification of M-TCP [8]. The M-TCP$^+$ algorithm improves TCP performance by ensuring self-resumption of a file transfer in the presence of frequent and lengthy disconnections, with and without the delayed acknowledgement (delayed ACK) option enabled. The delayed ACK option [9] allows the receiver to send an ACK for every second consecutive full-size packet received from the sender. (A full-size packet is equivalent to the sender maximum segment size (SMSS) packet.) The M-TCP$^+$ algorithm incorporates the delayed ACK option recommended for Internet hosts [9] and enabled in many current TCP implementations [10]. In wireless local area networks (WLANs), the delayed ACK option reduces the number of ACKs sent and improves TCP performance by reducing the number of collisions of data

* This research was supported by the NSERC grant 216844–03 and Canada Foundation for Innovation.

packets with ACKs [11]. In cases with disconnections, the proposed M-TCP$^+$ algorithm shows improved performance compared to M-TCP [8], TCP NewReno [12], and TCP SACK [13].

This paper is organized as follows. An overview of TCP congestion control algorithms is given in Section 2. In Section 3, we describe the M-TCP$^+$ algorithm. Simulation scenarios and performance comparison of the M-TCP$^+$ algorithm, M-TCP, TCP NewReno, and TCP SACK are presented in Section 4. We conclude with Section 5.

2. TRANSMISSION CONTROL PROTOCOL

2.1 Congestion Control Algorithms

TCP employs four congestion control algorithms [4], [5]: slow start, congestion avoidance, fast retransmit, and fast recovery, as shown in Fig. 1. Two TCP state variables, the congestion window size *cwnd* and the receiver's advertised window *rwnd*, are used by the congestion control algorithms to control the amount of data transmitted through the network. The minimum of the two determines the amount of data sent into the network.

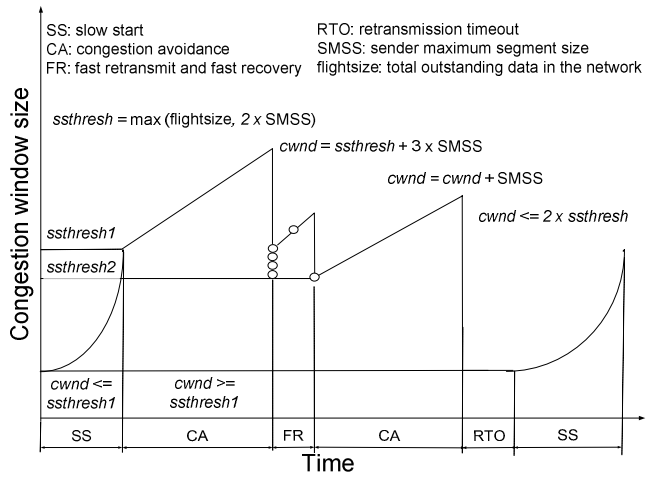

Fig. 1. TCP congestion control algorithms. The congestion window size depends on the congestion control algorithm and the mechanism used to indicate congestion.

The slow start threshold *ssthresh* determines when to use the slow start or congestion avoidance algorithm. The *ssthresh* may have an arbitrarily high initial value and is adjusted in response to congestion. At the start of transmission after the three-way handshake is completed, the *cwnd* value is equal to the initial window (IW) set to:

$$IW = \min (4 \times SMSS, \max (2 \times SMSS, 4380 \text{ bytes})). \quad (1)$$

The sender increments the *cwnd* exponentially during slow start and linearly during congestion avoidance phase, as shown in Algorithm 1.

```
if (cwnd < ssthresh)
    /* slow start phase */
    /* increment cwnd exponentially */
    cwnd += 1;
else
    /* change to congestion avoidance */
    /* increment cwnd by linearly */
    cwnd += 1/cwnd;
```

Algorithm 1. The pseudocode for incrementing the *cwnd* during slow start and congestion avoidance phases.

Congestion is detected by three duplicate ACKs or the RTO value. A TCP sender activates fast retransmit and fast recovery algorithms when three duplicate ACKs indicate congestion. In fast retransmit, a TCP sender retransmits data without waiting for the RTO timer to expire and sets the *ssthresh* value to half the current *cwnd*. In fast recovery, a TCP sender adjusts the *cwnd* for all segments buffered by a TCP receiver:

$$cwnd = ssthresh + 3 \times SMSS. \quad (2)$$

If the RTO mechanism is used, the *ssthresh* is set to:

$$ssthresh = \max (flightsize / 2, 2 \times SMSS), \quad (3)$$

where *flightsize* is the size of outstanding data in the network. A TCP sender then reduces *cwnd* to the SMSS and reverts to slow start algorithm. Hence, the RTO is more restrictive than the three duplicate ACK mechanism.

2.2 Estimation of RTT and RTO

A TCP sender maintains two variables used to compute the RTO value: smoothed RTT (*srtt*) (the moving average of RTT) and RTT variation (*rttvar*). The value of RTT is estimated using the Karn's algorithm [14] from RTT samples (*sampleRTT*) of data segments that are not retransmitted. The values of *srtt* and *rttvar* are computed as:

$$rttvar = (1 - \beta) \times rttvar + \beta \times | sampleRTT - srtt | \quad (4)$$

$$srtt = (1 - \alpha) \times srtt + \alpha \times sampleRTT. \quad (5)$$

Recommended parameter values are $\alpha = 0.125$ and $\beta = 0.25$ [9]. The value of *srtt* (4) is its value before the update (5). Hence, these values are calculated in the given order (4) and (5) [15]. RTO is then computed as:

$$RTO = srtt + \max (G, 4 \times rttvar), \quad (6)$$

where G is the clock granularity in seconds [14].

2.3 Delayed Acknowledgement

Many Internet TCP receivers implement the delayed ACK option [10]. A TCP receiver may increase efficiency by not sending ACKs for every data segment received [9]. By enabling the delayed ACK option, a TCP receiver increases network efficiency and maximizes bandwidth by acknowledging multiple segments and window updates with a single ACK. This option also reduces protocol processing overhead by reducing the number of ACKs per segment received [16]. Although ACKs are used by TCP to ensure window flow control and reliability, generating more ACKs than necessary is not a desirable characteristic in

heterogeneous wired/wireless networks. Hence, a TCP receiver may enable the delayed ACK option to generate the optimal number of ACKs required for reliable delivery of data segments and improved TCP performance [17]. It has been recommended that Internet hosts "should" implement delayed ACKs. The default interval period before sending an ACK is 200 ms [9]. However, a TCP receiver may wait up to 500 ms within the arrival of the last unacknowledged segment before the delayed ACK timer expires.

2.4 TCP Persist State

The TCP "persist" is a state during which a TCP sender maintains an open connection by sending window probes to the receiver while refraining from transmitting data segments. A TCP sender enters the persist state when it receives an ACK with the value *rwnd* reduced to zero and there are no outstanding unacknowledged bytes in flight [1]. (The TCP sender checks the absence of outstanding bytes by comparing the acknowledged byte numbers (ACK numbers) with the sent segment sequence number.) In the persist state, a TCP sender stops sending data segments until it receives an ACK with a nonzero value of *rwnd* [1]. However, it maintains the current *cwnd* and does not deflate the *cwnd* or close the TCP connection. A TCP sender uses the persist timer to maintain exchange of window size information while the value of the *rwnd* is zero. The persist timer computes its expiration period similar to the RTO timer. When the persist timer expires, a TCP sender sends a window probe segment to query a TCP receiver and determine if it can resume sending data segments. The window probe segment contains one byte of data. When an ACK with a nonzero value of *rwnd* is received in response to a probe segment, a TCP sender exits the persist state and resumes data transmission with the *cwnd* value prior to entering the persist state. The persist timer is reset once a TCP sender exits the persist state.

2.5 Related Work

Various solutions have been proposed to alleviate the effect of disconnections on TCP performance in wireless and heterogeneous networks. The M-TCP algorithm [8] and its OPNET implementation [18] were proposed to improve TCP performance in the presence of disconnections caused by handoff in mobile cellular networks. The M-TCP architecture [8] has a three-level hierarchy. The mobile hosts (MHs) at the lowest level communicate with base stations (BSs), which are connected to supervisor hosts (SHs) at the highest level. A single SH controls several BSs. The SHs are connected to a high-speed wired network and handle routing and connection management of MHs through the BSs. TCP connections between MHs and the high-speed wired network are split in two: one from the mobile to the SH and the other from the SH to the wired network. The SH receives data packets from a fixed host (FH) in the wired network and forwards it to the MH through the BS. When the SH receives an ACK from the MH, it forwards to the FH the ACK with the ACK number reduced by one. By reducing the ACK number, the SH is able to force the TCP sender into the persist state with the ACK number acknowledging the last (unacknowledged) byte when disconnections are detected.

V-TCP [19] is a TCP enhancement technique that mitigates the adverse effect of host mobility on TCP performance, with the TCP sender being either a FH or a MH. The Freeze-TCP algorithm [20] was proposed to improve performance of TCP in the presence of frequent disconnections and reconnections due to handoff or temporary blockage of radio signals by obstacles. When an impending disconnection is detected, the MH sends a zero window advertisement that forces the sender (FH) into the persist state. The described solutions assume that a TCP receiver sends an ACK for each data segment received and for data segments in flight and, hence, the last byte acknowledges the last data segment sent by a TCP sender. When a disconnection occurs, a TCP sender may not receive an ACK for the last unacknowledged data segment sent and, hence, it cannot enter the persist state even if the ACK previously received has the *rwnd* reduced to zero. A TCP sender then only refrains from sending additional data segments. However, it deflates the *cwnd* when the RTO timer expires.

3. M-TCP$^+$ ALGORITHM

The M-TCP$^+$ algorithm is proposed for heterogeneous wired/wireless networks. The main network elements are shown in Fig. 2. The algorithm considers cases with and without the delayed ACK option enabled. Without disconnections, the M-TCP$^+$ algorithm maintains the end-to-end TCP semantics by acknowledging data that the wireless host (WH) has already acknowledged. When the WH senses an impending disconnection, it sends a disconnection feedback signal to the intermediate host (IH). This signal is based on the received signal level measured at the WH [21]. When the IH receives this signal, it acknowledges all bytes sent by the FH including data in flight yet to be acknowledged by the WH. The IH then sets the *rwnd* of the last ACK sent to the TCP sender to zero. After all bytes have been acknowledged, the FH enters the persist state and the RTO timer does not expire. Hence, the *cwnd* is not reduced to the SMSS. Once the WH is reconnected, the FH resumes sending data with the value of *cwnd* prior to the disconnection. If the IH does not receive a disconnection feedback signal from the WH, the M-TCP$^+$ algorithm at the IH would not be initiated.

3.1 Network Model

The network model that we consider is a closed-loop TCP system. The FH sends data and uses the ACKs received as a feedback from the network to increase or decrease its *cwnd*. The TCP congestion control and flow control mechanisms rely on ACKs received from the network. Hence, if a feedback signal could be sent to indicate an

impending disconnection, TCP could react appropriately by preventing unnecessary expiration of the RTO timer and deflation of the *cwnd*. Thus, the available network bandwidth is preserved for other TCP connections.

We employ a simple client-server model suitable for evaluating performance of the M-TCP$^+$ algorithm, as shown in Fig. 2. Simulation scenarios capture WH downloads of a lengthy file from a FH in a wired network. The WH connects to the IH through a WLAN link, often used as a last-hop link [22]. The IH is connected to the wired high-speed Ethernet infrastructure network. The IH may be a WLAN router

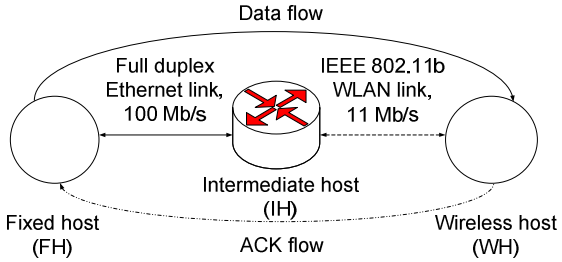

Fig. 2. Heterogeneous network with the FH (server) connected to the IH (router) through a 100 Mb/s full duplex Ethernet link. The WH (client) is connected to the IH (router) through an IEEE 802.11b WLAN link.

3.2 M-TCP$^+$ Modification

Implementation of M-TCP$^+$ requires modifications only at the IH. The IH employs TCP port numbers to differentiate between TCP connections. When the IH receives a TCP segment from the FH, it checks the TCP segment's sequence number (*seq_no*) to determine if it is a new segment. If the segment is new, the IH stores the value of its *seq_no*, keeps a copy of the segment in a first-in-first-out (FIFO) finite queue, and then forwards it to the WH. When the IH receives an ACK from the WH, it checks the ACK number (*ack_no*) of the segment, compares it to the value of the stored *seq_no*, and purges any segments in the FIFO queue preceding the received segment. The *ack_no* indicates the *seq_no* of the next segment expected by the WH. The IH then forwards the ACK to the FH.

However, unlike the SH of the M-TCP, the IH forwards all ACKs as received from the WH and does not retain the last byte of the ACK. In the proposed M-TCP$^+$ implementation, the IH forwards all ACKs as received because the sender increases the *cwnd* based on the number of acknowledged bytes rather than the number of ACKs. With the delayed ACK option enabled, there may be data segments in flight to be acknowledged to force the TCP sender into the persist state. Hence, the IH does not need to retain a byte to force the TCP sender into the persist state. The unacknowledged data or ACKs segments in flight may be located either in the wired or wireless section of the connection, as shown in Fig. 3.

The IH may not receive the ACKs in flight when disconnections occur. When the IH receives a disconnection feedback signal from the WH, the IH employs the procedure shown in Algorithm 2. The IH sends the last received ACK from the WH with *rwnd* set to zero and, thus, the FH refrains from sending additional data segments. By sending an ACK segment with *ack_no* set to the FH maximum sent sequence number (*snd_max*), the IH acknowledges all segments up to the *snd_max*. Hence, all outstanding unacknowledged FH segments shown in Figs. 3(a), (b), and (c) are acknowledged. The IH also stores all segments unacknowledged by the WH in the FIFO queue to be forwarded to the WH after reconnection. When the TCP sender receives the ACK for the maximum unacknowledged byte, it enters the persist state. While in the persist state, the FH sends TCP window probes of one byte after each expiration of the persist timer. As long as the WH remains disconnected, the IH responds to these probes with the last ACK sent and *rwnd* set to zero.

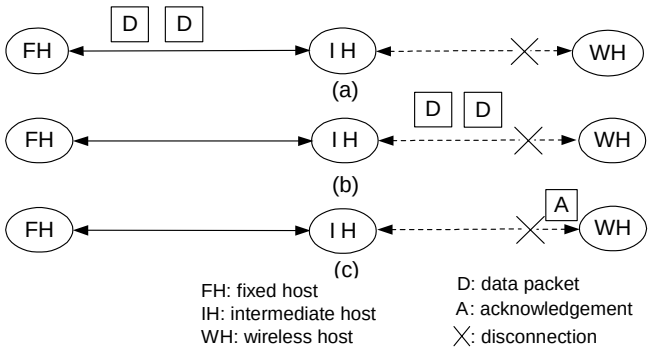

Fig. 3. Scenarios when WH is disconnected from the network: (a) unacknowledged data packets in flight in the wired link, (b) unacknowledged data packets in flight in the wireless link, and (c) ACK packet in flight in the wireless link. Solid and dashed lines represent wired and wireless links, respectively.

w = *ack_no* of the most recent ACK segment received from the WH
snd_max = *seq_no* of most recent unacknowledged segment received from FH
if (disconnection signal received)
{
/* set rwnd value in most recent ACK */
/* segment received from the WH to zero */
1) set *rwnd* = 0
2) forward the ACK to sender
if (*w* < *snd_max*)
{
/* set the ack_no of most recent ACK segment */
/* received from the WH to snd_max */
1) set *w* = *snd_max*
2) set *rwnd* = 0
3) forward the ACK to sender
}
}

Algorithm 2. The pseudocode of the M-TCP$^+$ algorithm at the IH in case of disconnections.

4. SIMULATION SCENARIOS AND RESULTS

4.1 Simulation Scenarios

We simulated a heterogeneous network shown in Fig. 2 using the OPNET simulation tool [23]. The FH is an FTP server connected to a router represented by the IH. We consider disconnections when the delayed ACK option is enabled. We assume that forward error correction (FEC) is used at the MAC layer. Hence, we assume the negligible bit error rate (BER) in the IEEE 802.11b WLAN link. A distance less than 300 m is maintained between the fixed WH and the router, as required by the IEEE 802.11 WLAN standard, to ensure that all packets reach their destinations except during simulated disconnection periods.

The receiver's advertised window *rwnd* was set to 8,760 bytes, thus allowing up to six data segments to be sent when the WH reconnects to the network. A larger *rwnd* value may allow additional data segments to be transmitted through the network. However, this may cause frequent bursty data transmissions. The Karn's algorithm was enabled to calculate the RTT. Selected TCP simulation parameters when the delayed ACK option was enabled are shown in Table 1. The identical set of parameters was used when the delayed ACK option was disabled, with the exception of the maximum ACK delay and maximum ACK segment that were set to 0.0 s and 1, respectively.

Table 1. Selected TCP simulation parameters.

TCP Parameters	Value
Initial RTO	3.0 s
Minimum RTO	1.0 s
Maximum RTO	64.0 s
Timer granularity	0.5 s
Persist time-out	1.0 s
Maximum ACK delay	0.2 s
Maximum ACK segment	2
Duplicate ACK threshold	3
Sender maximum segment size (SMSS)	1,460 bytes
Slow start initial count	1 SMSS
Receiver's advertised window	8,760 bytes
Retransmission threshold	6
RTT gain	0.125
RTT deviation coefficient	4
Deviation gain	0.25

In the case of frequent disconnections, we chose three disconnection periods (5 s, 10 s, and 30 s) within a one-minute cycle. A 180 s disconnection period within a five-minute cycle was chosen for lengthy disconnections. We simulated 1,000 s of an FTP application file download. The file size was 100 MB. Scenarios without disconnections were simulated to verify the M-TCP$^+$ implementation and detect any significant delay due to additional processing introduced by the M-TCP$^+$ algorithm. We simulated each scenario with and without delayed ACKs. The effect of the M-TCP$^+$ algorithm in the presence of congestion was evaluated by simulating a scenario with congestion with the 10 s disconnection period. Identical random number seeds (equal to 128) were used in all simulation scenarios because of the negligible BER.

4.2 Simulation Results

TCP NewReno and TCP SACK are common Internet TCP implementations [21]. We compared the performance of the M-TCP$^+$ algorithm with the original M-TCP, TCP NewReno, and TCP SACK in all simulation scenarios.

We evaluated the performance in terms of file download response time, goodput, and retransmission ratio. Download response time is the time elapsed between the instant the wireless host (WH) sends the FTP application request and receives the complete file from the fixed host (FH). The download response time is considered to be zero if the file download is terminated before completion. Goodput is calculated as the difference between the number of segments received by the WH and the number of retransmitted segments during the duration of the connection. We used the received segment sequence number at the WH as an indicator of goodput. For FH segments, retransmission ratio percentage is calculated as:

$$retransmission\ ratio = \frac{\#\,retransmitted\ segments}{total\ number\ of\ segments - retransmitted\ segments} \times 100\,. \quad (7)$$

1) Absence of disconnections and congestion: We verified the implementation of the proposed algorithm by considering the simple scenario without disconnections and congestion. The download response time with and without delayed ACKs is shown in Table 2. The four TCP variants exhibit similar performance in all cases. Hence, the M-TCP$^+$ algorithm does not introduce significant processing delay to the file download. The download response times for the case with delayed ACKs (enabled) are lower than for the case with delayed ACKs (disabled). In the absence of disconnections, the case with delayed ACKs (enabled) shows higher goodput than the case without delayed ACKs (disabled), as shown in Fig. 4. All four TCP variants reach steady-state beyond 300 s of simulation time. No segment losses occurred in the scenarios without disconnection for both cases with and without the delayed ACK enabled and, hence, there are no retransmissions.

Table 2. Download response time (s) for scenarios without disconnections.

Delayed ACK	TCP Variant			
	M-TCP$^+$	M-TCP	NewReno	SACK
Enabled	261.0	261.1	261.0	261.0
Disabled	287.4	287.8	287.4	287.4

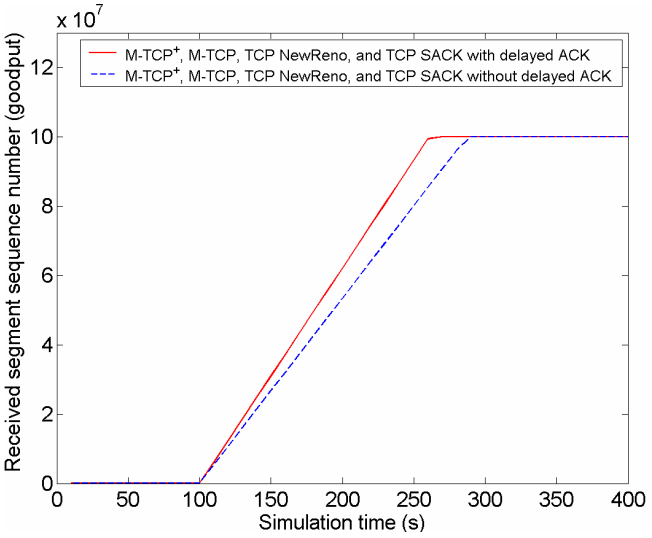

Fig. 4. Scenarios without disconnections. The received segment sequence number at the TCP receiver is used as an indicator of goodput. The four TCP variants have similar goodput performance.

2) Presence of disconnections only: The download response time with delayed ACKs (option enabled) and without it (option disabled) is shown in Table 3. The M-TCP$^+$ algorithm exhibits the lowest download response time in both cases for all disconnection periods. In the case of a lengthy (180 s) disconnection with delayed ACKs, the M-TCP$^+$ algorithm is the only variant that completes the file download. The M-TCP algorithm, TCP NewReno, and TCP SACK exceed the maximum number of retransmissions. Hence, their connections close and the file downloads are not completed. The M-TCP$^+$ algorithm reduces download response time by 2%–15%. The download response time with delayed ACKs remains lower than without delayed ACKs. The M-TCP$^+$ algorithm ensures the fastest file download because it eliminates returning to the slow start phase after each RTO timer expiration due to disconnections.

Table 3. Download response time (s) for scenarios with disconnections.

Delayed ACK	TCP variant	Disconnection period			
		5 s	10 s	30 s	180 s
Enabled	M-TCP$^+$	277.9	302.4	435.6	622.4
	M-TCP	278.7	316.2	484.5	0.0
	NewReno	283.0	322.5	439.4	0.0
	SACK	283.0	322.5	439.4	0.0
Disabled	M-TCP$^+$	311.4	327.9	489.3	647.8
	M-TCP	311.9	328.8	489.3	649.4
	NewReno	316.8	384.4	497.8	0.0
	SACK	316.8	384.4	497.8	0.0

The goodput of the scenarios with 10 s disconnection for cases with and without delayed ACK option enabled is shown in Figs. 5 and 6, respectively. The four TCP variants are in steady-state beyond 350 s of the simulation time. With delayed ACK option enabled, the M-TCP$^+$ algorithm outperforms the other three TCP variants and improves goodput by ~12%. When the delayed ACK option is disabled, the M-TCP$^+$ and M-TCP algorithms exhibit similar goodput performance and improve goodput by ~10%. TCP NewReno and TCP SACK have similar goodput in cases with and without delayed ACK option enabled. Other frequent disconnection periods that we considered show a similar pattern. In the case of a lengthy (180 s) disconnection, TCP NewReno and TCP SACK did not complete the file transfer for either case with or without the delayed ACK option enabled.

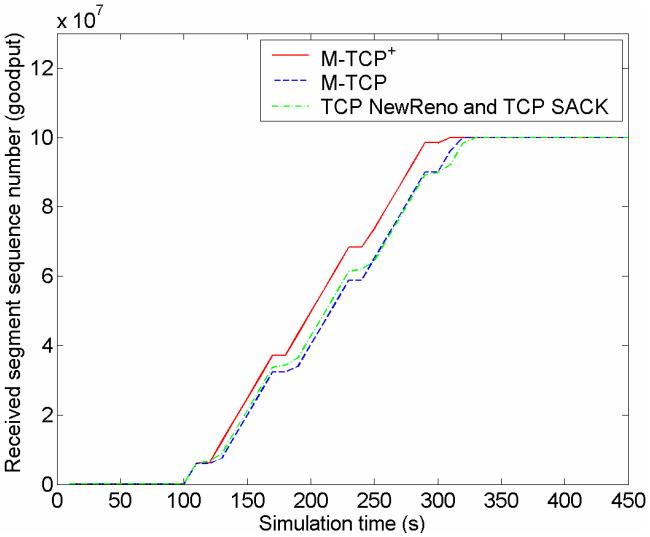

Fig. 5. Disconnection scenarios with the delayed ACK option disabled. The disconnection period is 10 s. The M-TCP$^+$ and M-TCP algorithms exhibit higher goodput than TCP NewReno and TCP SACK.

The retransmission ratio in scenarios with disconnections for cases with and without delayed ACK option enabled is shown in Table 4. The M-TCP$^+$ algorithm shows the lowest retransmission ratio in all cases because it prevents unnecessary retransmission of segments when the RTO timer expires during disconnections. In the case of a lengthy (180 s) disconnection, TCP NewReno and TCP SACK have higher retransmission ratios even though fewer segments were transmitted by the FH. When the delayed ACK option is disabled, the M-TCP$^+$ and M-TCP algorithms show similar retransmission ratios, which are 2%–8% lower than TCP NewReno and TCP SACK. The M-TCP$^+$ algorithm prevents unnecessary retransmissions when congestion is absent and when segment losses are only due to disconnections.

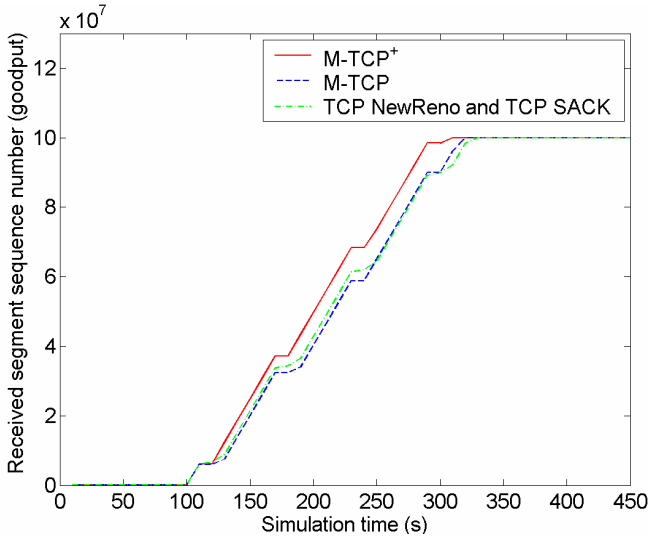

Fig. 6. Disconnection scenario with the delayed ACK option enabled. The disconnection period is 10 s. The M-TCP$^+$ algorithm exhibits the best goodput.

Table 4. Retransmission ratio percent for scenarios with disconnection only.

Delayed ACK	TCP variant	Disconnection period			
		5 s	10 s	30 s	180 s
Enabled	M-TCP$^+$	0.4	0.5	1.2	0.3
	M-TCP	2.6	5.0	9.2	8.4
	NewReno	3.5	5.3	7.7	6.8
	SACK	3.5	5.3	7.7	6.8
Disabled	M-TCP$^+$	0.6	0.6	1.3	0.3
	M-TCP	0.6	0.6	1.3	0.3
	NewReno	4.7	5.3	9.2	7.8
	SACK	4.7	5.3	9.2	7.8

3) Presence of disconnections and congestion: The download response time with and without delayed ACKs is shown in Table 5. In the case with delayed ACKs, the M-TCP$^+$ algorithm outperforms the M-TCP algorithm, TCP NewReno, and TCP SACK. As expected, the download response times in both cases, with and without delayed ACKs, are higher for the same disconnection period (10 s) without congestion.

The goodput of the congestion scenarios with and without the delayed ACK option enabled is shown in Figs. 7 and 8, respectively. The TCP variants reach steady-state beyond 500 s of the simulation time. The M-TCP$^+$ algorithm shows higher goodput (with or without the delayed ACK option enabled) than the other three TCP variants. Hence, the M-TCP$^+$ algorithm does not degrade goodput performance during congestion. It improves goodput by ~8% and ~12% with and without delayed ACKs, respectively. For the four TCP variants, goodput is lower in the scenarios with both disconnections and congestion than in the scenarios with disconnections only. This indicates, as expected, the impact of congestion.

Table 5. Download response time (s) for congestion scenarios with 10 s disconnection period.

Delayed ACK	TCP Variant			
	M-TCP$^+$	M-TCP	NewReno	SACK
Enabled	421.0	445.4	440.7	440.7
Disabled	445.5	447.4	464.3	464.3

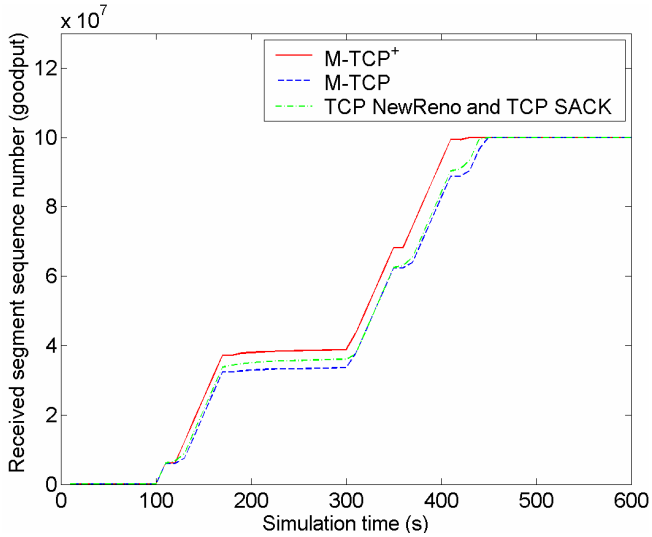

Fig. 7. Congestion scenario for the 10 s disconnection period with the delayed ACK option enabled. The M-TCP$^+$ algorithm exhibits the highest goodput.

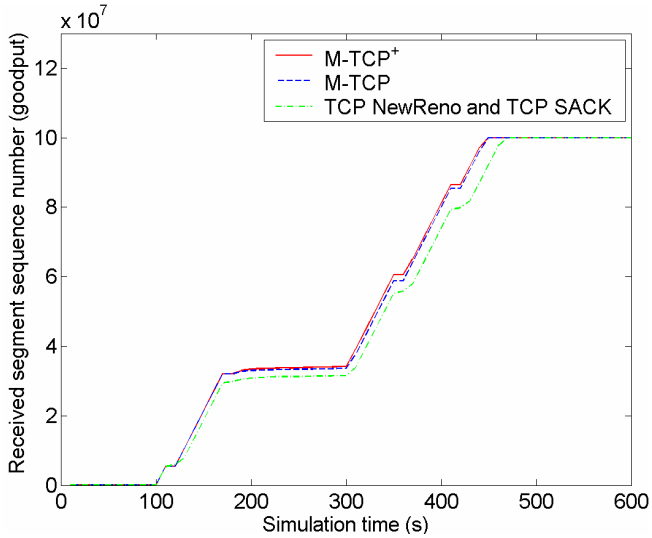

Fig. 8. Congestion scenario with disconnection period of 10 s and the delayed ACK option disabled.

The retransmission ratios for congestion scenarios with and without the delayed ACK option enabled are shown in Table 6. The M-TCP$^+$ algorithm shows lower retransmission ratio than the other three TCP variants in the case with the delayed ACK option enabled. The M-TCP$^+$ and M-TCP algorithms have similar reduction of ~7% in terms of retransmission ratios and both outperform TCP NewReno and TCP SACK when the delayed ACK option is disabled.

In the presence of congestion, there are additional segment losses. Hence, as expected the retransmission ratio is higher for the four TCP variants in the congestion scenarios than in scenarios without congestion.

Table 6. Retransmission ratio (%) for congestion scenarios with 10 s disconnection period.

Delayed ACK	TCP Variant			
	M-TCP$^+$	M-TCP	NewReno	SACK
Enabled	0.9	7.0	7.6	7.6
Disabled	0.7	1.6	7.6	7.6

5. CONCLUSIONS

We described the M-TCP$^+$ algorithm for improving TCP performance in the presence of disconnections during lengthy file transfers. The M-TCP$^+$ algorithm was implemented in the OPNET network simulator as an extension to TCP NewReno. Simulation results illustrate that the M-TCP$^+$ algorithm does not increase processing delay of the file download response time in the scenarios without disconnections. The M-TCP$^+$ algorithm reduces the file download response time, increases the goodput, and reduces the retransmission ratio compared to TCP NewReno and TCP SACK. When the delayed ACK option is enabled, the performance of the original M-TCP algorithm degrades because of unacknowledged data segments in flight during disconnections. However, when the delayed ACK option is disabled, the M-TCP$^+$ and M-TCP algorithms had similar performance. In the case of congestion, simulation results also indicated that the M-TCP$^+$ algorithm outperformed M-TCP, TCP NewReno, and TCP SACK. In all cases, the delayed ACK option enabled outperforms the cases with the delayed ACK option disabled.

The deployment of the M-TCP$^+$ algorithm in wired/wireless networks requires additional buffers at an IH to store packets that will be retransmitted to a WH when it reconnect to the network after disconnections. However, implementation of the M-TCP$^+$ algorithm only requires modifications at intermediate nodes such as routers. No modifications are necessary at FH (TCP senders) or WH (TCP receivers) nodes.

ACKNOWLEDGEMENT

The authors thank W. G. Zeng, R. Narayanan, S. Lau, and B. Vujičić from the Communication Networks Laboratory at Simon Fraser University for constructive suggestions and comments.

REFERENCES

[1] W. Stevens, *TCP Illustrated*, vol. 1. Reading, MA: Addison-Wesley, 1994.

[2] C. Williamson, "Internet traffic measurement," *IEEE Internet Computing,* vol. 5, no. 6, pp. 70–74, Nov./Dec. 2001.

[3] K. Pahlavan and P. Krishnamurthy, *Principles of Wireless Networks: A Unified Approach*, Upper Saddle River, NJ: Prentice Hall, 2001.

[4] V. Jacobson, "Congestion avoidance and control," in *Proc. ACM SIGCOMM Symp. on Commun. Archit. and Protocols*, Stanford, CA, USA, Aug. 1988, pp. 314–329.

[5] M. Allman, V. Paxson, and W. Stevens, "TCP congestion control," *IETF RFC 2581*, Apr. 1999.

[6] K. Ratnam and I. Matta, "WTCP: an efficient transmission control protocol for networks with wireless links," in *Proc. IEEE Int. Symp. on Comput. and Commun.*, Athens, Greece, July 1998, pp. 74–78.

[7] Z. Fu et al., "The impact of multiphop wireless channel on TCP throughput and loss," in *Proc IEEE INFOCOM*, San Francisco, CA, USA, Apr. 2003, pp. 1744–1753.

[8] K. Brown and S. Singh, "M-TCP: TCP for mobile cellular networks," *ACM Computer Commun. Review*, vol. 27, no. 5, pp. 19–43, Oct. 1997.

[9] R. Braden, "Requirements for Internet hosts-communication layers," *RFC 1122*, Oct. 1989.

[10] V. Paxson, "Automated packet trace analysis of TCP implementations," in *Proc. ACM SIGCOMM Conf. on Applications, Technol., Archit., and Protocols for Comput. Commun.*, Cannes, France, Sept. 1997, pp. 167–179.

[11] C. Joo and S. Bahk, "Increasing TCP capacity in wireless multihop networks," in *Web and Commun. Technol. and Internet-Related Social Issues–HSI*, *Lecture Notes in Comput. Science*. Springer, Berlin: vol. 3597, pp. 37–44, 2005.

[12] S. Floyd, T. Henderson, and A. Gurtov, "The NewReno modification to TCP's fast recovery algorithm," *IETF RFC 3782*, Apr. 2004.

[13] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanov, "TCP selective acknowledgement options," *RFC 2018*, Oct. 1996.

[14] P. Karn and C. Patridge, "Improving round-trip time estimates in reliable transport protocols," *ACM Trans. Comput. Syst.*, vol. 9, no. 4, pp. 364–373, Nov. 1991.

[15] V. Paxson and M. Allman, "Computing TCP's retransmission timer," *IETF RFC 2988*, Nov. 2000.

[16] D. D. Clark, "Window and acknowledgement strategy in TCP," *RFC 813*, July 1982.

[17] J. Chen, Y. Z. Lee, M. Gerla, and M. Y. Sandidi, "TCP with delayed ack for wireless networks," in *Proc. IEEE/CreateNet BROADNETS 2006*, San Jose, CA, USA, Oct. 2006.

[18] W. G. Zeng, M. Zhan, Z. Lin, and Lj. Trajković, "Performance evaluation of M-TCP over wireless links with periodic disconnections," *OPNETWORK 2003*, Washington DC, USA, Aug. 2003.

[19] D. Nagamalai, B. C. Dhinnakaran, B.-S. Choi, and J.-K. Lee, "V-TCP: a novel TCP enhancement technique," in *Networking-ICN, Lecture Notes in Comput. Science.* Springer, Berlin: vol. 3421, pp. 125–132, Mar. 2005.

[20] T. Goff, J. Moronski, and V. Gupta, "Freeze-TCP: a true end-to-end TCP enhancement mechanism for mobile environments," in *Proc. IEEE INFOCOM*, Tel-Aviv, Israel, Mar. 2000, pp. 1537–1545.

[21] Z. Cheng and W. B. Heinzelman, "Exploring long lifetime routing (LLR) in ad hoc networks," in *Proc. ACM MSWiM*, Venice, Italy, Oct. 2004, pp. 203–210.

[22] A. Gurtov and S. Floyd, "Modeling wireless links for transport protocols," *ACM Comput. Commun. Review,* vol. 34, no. 2, pp. 85–96, Apr. 2004.

[23] OPNET Modeler software [Online]. Available: http://www.opnet.com/products/modeler/home.html.