Improving TCP Performance with Periodic Disconnections over Wireless Links

Wan G. Zeng, Meihua Zhan, Zhiwen Lin, and Ljiljana Trajkovic
Simon Fraser University
Burnaby, B.C. V5A1S6
Email: {wgzeng, mjzhan, zlin, ljilja}@cs.sfu.ca

Abstract

Since 1988, when the congestion control functions were first introduced, the performance of TCP has been greatly improved. However, TCP still suffers large performance degradation over wireless links due to characteristics specific to wireless environment that affect the behavior of TCP's congestion control and avoidance mechanisms. These mechanisms were designed and optimized for traditional wireline networks where packet losses are predominantly due to network congestion. In wireless networks, packet losses are mainly caused by the high bit error rate (BER) and hand-offs. Since TCP cannot differentiate packet losses caused by congestion from losses introduced by the wireless links, its performance degrades. M-TCP is a solution proposed to address the problems of TCP over wireless links with periodic disconnections.

In this paper, we first give a brief introduction to M-TCP protocol. We then describe the OPNET implementation of M-TCP and the simulation scenarios in a mixed wireline/wireless environment. We give performance comparisons between M-TCP and TCP with and without presence of frequent disconnections in wireless links. Simulation results indicate that in the presence of frequent disconnections M-TCP outperforms TCP in terms of maintaining congestion window size, goodput, and sender size retransmission timer.

Keywords: Wireless networks, mobile cellular networks, TCP/IP, congestion control, quality of service.

1. Introduction

TCP's congestion control mechanisms [1] were designed and optimized for traditional wireline networks where packet loss is predominantly due to network congestion. These mechanisms greatly improved TCP performance and made TCP the most popular Internet transport protocol. However, when wireless networks are merged with wireline networks, TCP performance largely degrades. In wireless networks, packet losses are mainly caused by the high BER and hand-offs. Since TCP cannot differentiate packet losses caused by congestion from losses introduced by the wireless links, it is unable to handle these losses separately. This is the main cause for TCP's performance degradation in a mixed wireless/wireline environment. Numerous solutions have been proposed [2] - [4] to prevent TCP from performing unnecessary congestion control when there is no congestion in the network. M-TCP [4] is one of the solutions proposed to address the problems of TCP over wireless links with periodic disconnections, as in the case of frequent hand-offs.

We have implemented M-TCP in OPNET and have done performance comparisons between M-TCP and TCP. Simulation results indicate that in the presence of disconnections M-TCP has better performance in terms of congestion window size, goodput, and sender side retransmission timer. Congestion window size of the M-TCP sender is very close to the window size in an ideal network without disconnections. There is also a visible difference between M-TCP and TCP goodputs. The M-TCP sender side retransmission timer is, again, close to its value in an ideal network.

This paper is organized as follows. Section 2 provides TCP overview, describes the properties of regular TCP, and introduces M-TCP protocol. Section 3 describes OPNET implementation of M-TCP in a Wireless LAN (WLAN) network. Section 4 shows the performance comparisons of a network employing M-TCP protocol with the same network using regular TCP protocol. We conclude with Section 5.

2. Transmission Control Protocol (TCP)

2.1 TCP Overview

In this section, we discuss TCP's timer and window management mechanisms.

To perform congestion control and avoidance, TCP maintains two windows: the receiver's advertised window (*rwnd*) and the congestion window (*cwnd*). They define the maximum number of bytes the receiver may receive and the sender may send, respectively. The number of bytes that may be sent into the network is the minimum of the two. Therefore, when *rwnd* is large enough, the larger the *cwnd*, the more data TCP can send into the network. TCP is known to be acknowledgement (ACK) paced. With every byte that TCP sends, the sender will wait for the ACK from the receiver to send the amount of data defined by *cwnd*.

TCP detects packet losses in the network by either duplicate ACKs or retransmission timeouts [1]. Therefore, TCP keeps a timer called retransmission timer for its timer based retransmission mechanism. The timer is used to define how long TCP sender should wait before retransmitting an unacknowledged packet. Retransmission timeout (RTO) is calculated based on the estimation of the packet round-trip time (RTT) in the network. When a packet loss is detected, TCP considers it as the sign of congestion in the network and shrinks its *cwnd* accordingly to limit the amount of data to be sent into the network. In addition, if the retransmission timer detects the packet loss, TCP also exponentially increases its RTO. Repeated losses will force the sender's *cwnd* to remain small and the RTO to remain large, which results in smaller amount of data being sent into network and longer intervals between each attempt to probe network connectivity.

TCP moves into *persist* state when there is data to be sent and the other end of the connection sends a windows update equal to zero [1]. In the persist state, TCP freezes all of its states, including *cwnd* and retransmission timer, waiting to resume transmission at any time. The persist timer starts, and when it expires the sender sends a one-byte packet to probe the other end of the connection to check if the window is open. A response to the probe with window size greater than zero will trigger the sender to leave the *persist* state and be ready to send new data.

2.2 M-TCP Protocol

M-TCP is a modified version of TCP, designed to specifically suit wireless connections. It is designed to work with *cellular* network infrastructure [5], a typical type of wireless network architecture. In such wireless networks, congestion control algorithms introduce difficulties for TCP. A *cellular* network infrastructure is typically used to connect mobile users to the Internet. A geographical region, such as a city or a campus, is divided into cells. Each cell contains one or more base stations that provide connection end-points for mobile hosts.

It is likely that in order to provide high-bandwidth wireless connections, cell sizes will have to be kept small. Small cell size, unfortunately, results in small cell latency that, in turn, causes frequent disconnections as a user roams among cells [5]. This causes serial timeouts at the TCP sender. Because the sender doubles the retransmission timer with each unsuccessful retransmission, even when the mobile is reconnected it is feasible that in most TCP implementations no data could be transmitted for as long as one min.

Figure 1 shows the network architecture suitable for M-TCP deployment. It has a three level hierarchy. At the lowest level, each cell contains mobile hosts (MH) communicating with their mobile support stations (MSS). A Supervisor Host (SH) controls several MSSs. The SHs are connected to the wireline network, and they handle most routing and other protocol details for the mobile users. They also maintain connections for mobile users, handle flow-control, and are responsible for maintaining the negotiated quality of service. These SHs, thus, perform the function of gateways [5]. The fixed hosts (FH) reside in high-speed wireline networks.

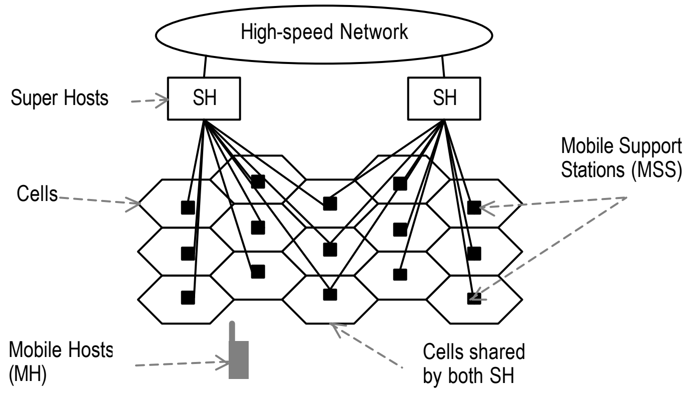


Figure 1: Proposed architecture [6].

M-TCP takes the split connection approach and divides the connection into wireline and wireless domains. In the case of disconnections, the sender is forced into *persist* state by receiving persist packets from M-TCP. While in persist state, the sender will not suffer retransmit timeout, it will not exponentially back off its retransmission timer, and it will preserve the size of its congestion window [1]. Hence, when the connection recovers upon receiving a notification from M-TCP, the sender will be able to transmit at full speed.

A simplified high-level model of the network is shown in Figure 2. TCP connections are split into two at the SH. The TCP client at the SH, called SH-TCP, receives data packets transmitted by the sender (FH) in the wireline network. It passes these packets to the M-TCP client for delivery to the MH. ACKs received by M-TCP at the SH are forwarded to SH-TCP for delivery to the TCP sender. When SH-TCP receives a packet from a TCP sender, it passes the packet to the M-TCP client, and acknowledges packet when its ACK is received from the MH. However, a protocol needs to ensure that the sender does not trigger congestion control algorithms when MH is temporarily disconnected or a packet is lost between SH and MH.

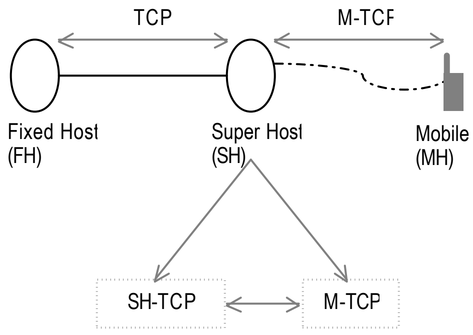


Figure 2: Setting up a TCP connection [4].

Because only a new ACK packet with *cwnd* equals to zero can force the sender into persist state according to the standard TCP protocol [1], the last ACK number (the largest acknowledged sequence number from MH) has to be kept in order to be able to create such a new ACK packet. When retransmission timeout occurs, SH-TCP creates and sends this new ACK packet to FH. This packet will also contain a TCP *rwnd* equal to zero. Upon receiving this ACK packet, FH is forced into the *persist* state. While in this state, FH will not suffer from retransmission timeouts, it will not exponentially back off its retransmission timer, nor will close its congestion window. The state of the sender does not depend on the length of the disconnection period.

SH checks for the ACK packets to determine if MH is reconnected. When the MH is reconnected, it will send a reconnection ACK to SH. Upon receiving reconnection ACK, SH will retransmit all the packets that have not been acknowledged by MH. The new ACK packets will be forwarded to SH-TCP and FH so that they can resume their transmission and exit the persist state. Since the sender never times out, it never performs TCP congestion control. Thus, the sender can resume transmission at the rate before the packet loss occurred

M-TCP protocol does not take into consideration wireless high BER because, in most mobile environments, a good wireless link layer protocol will ensure a small BER seen at the TCP layer. However, with M-TCP, even in mobile environment with high BER, the sender can still leave *persist* state quickly and resume regular data transmissions. When in *persist* state, sender constantly sends persist packets to the receiver who will then be forced to respond.

3. OPNET Implementation

We have implemented M-TCP in OPNET. OPNET provides sets of standard models that we used for the model development. By carefully choosing OPNET models from the Model Library, with only minor modifications we completed the implementation and performance evaluation of M-TCP.

3.1 OPNET Network Model

We used the network model configuration shown in Figure 3. Three node models are selected from two standard lists of OPNET models: Wireless LAN (WLAN) and Ethernet. We use standard WLAN Ethernet Router model (*wlan_ethernet_router*) to model the SH, WLAN Workstation (*wlan_wkstn_adv*) for MH, and Ethernet Server (*ethernet_server_adv*) for FH. Ethernet Server is a model with server applications running over TCP/IP and UDP/IP. It has an interface to one Ethernet connection at 10 Mbps, 100 Mbps, or 1,000 Mbps. These meet our requirements to have a TCP connection to SH via Ethernet. WLAN Workstation is chosen for the mobile host for similar reasons. It represents a workstation with applications running over TCP/IP and UDP/IP. It supports one underlying WLAN connection at 1 Mbps, 5.5 Mbps, or 11 Mbps. Lastly, WLAN Ethernet Router was used to establish connection between FH and MH because it has the capability of connecting to both Ethernet (one Ethernet port available) and wireless LAN (one wireless LAN port available).

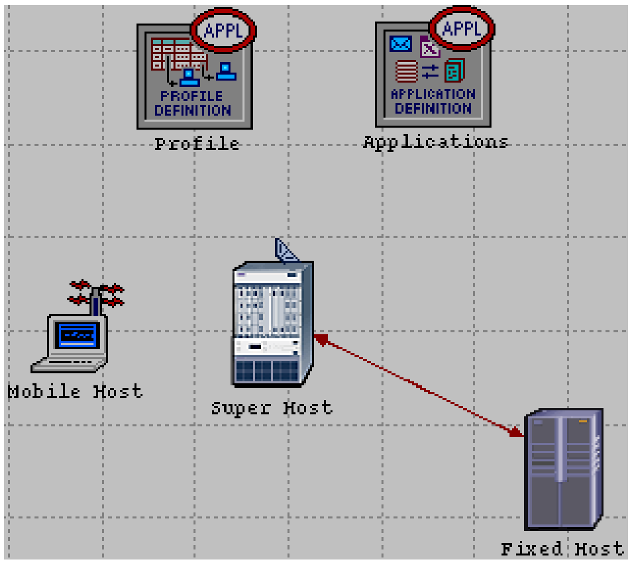


Figure 3: Network model.

3.2 OPNET Node Models

We made modifications (described in Section 2.2) only to MH and SH. There was no need for modifications in the wireline network domain (including FH). This is one of the most important features of M-TCP. Most existing mobile TCP protocols are designed not to require modifications of existing wireline network components, because majority of the network components belong to wireline networks. The TCP within MH is modified into an M-TCP client. Most modifications are implemented in SH, where the functionality of the original TCP connection is split into two layers: SH-TCP and M-TCP. The SH-TCP layer is used for communications between the SH and the FH on wireline links. The M-TCP layer is used to handle communications between the SH and the MH in the wireless links. We now describe in details the modifications to the MH and SH.

3.3 Modifications to the Mobile Host

Figure 4 shows the MH node model. We modified the *ip_encap* process. The modified process, named *m_ip_encap,* simulates MH's temporary disconnection behavior. Packets handled by this process are dropped periodically. This models the disconnections between SH and MH. In theory, any processes along the data path between MH and SH can break the packet stream. We choose to modify *ip_encap* for the ease of implementation. This process is relatively simple. It is responsible for de-capsulation of IP packets sent to the higher layer, and en-capsulation of packets into IP packets sent to the lower layer.

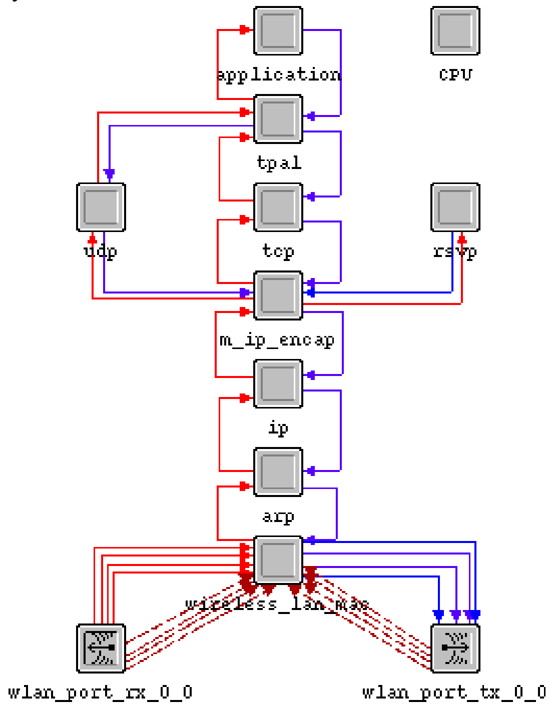


Figure 4: Node model for Mobile Host.

In our implementation, we intercept the network traffic in both *DECAP* and *ENCAP* states of the *ip_encap* state diagram shown in Figure 5.

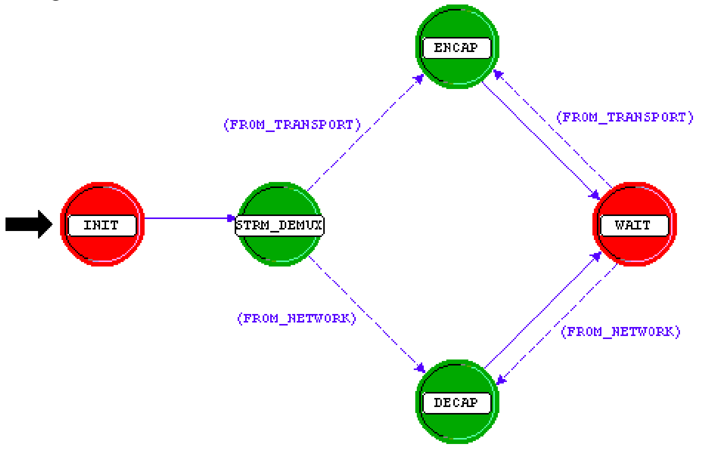
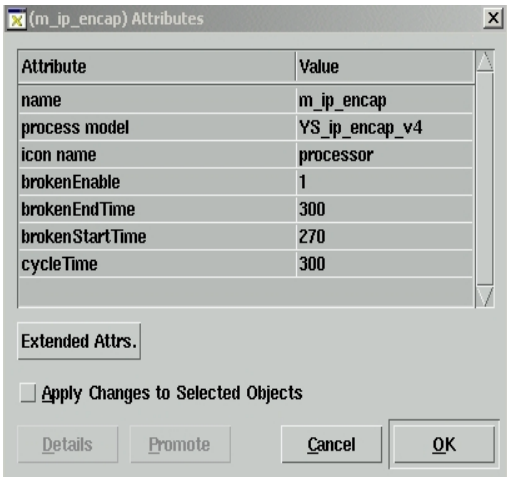


Figure 5: *ip_encap* state diagram.

Four configurable attributes are implemented to configure the *m_ip_encap* process module:

BrokenEnable:	If set to 1, periodical disconnection is enabled.
CycleTime:	Indicates period (in seconds) between two disconnections.
BrokenStartTime:	The start time of disconnection in each disconnection cycle.
BrokenEndTime:	The end time of disconnection in each disconnection cycle.

Figure 6 shows a typical simulation configuration, with *CycleTime* = 300 sec, *BrokenStartTime* = 270 sec, and *BrokenEndTime* = 300 sec. This generates a 30 sec disconnection for every 5 min.


(m_ip_encap) Attributes

Attribute	Value
name	m_ip_encap
process model	YS_ip_encap_v4
icon name	processor
brokenEnable	1
brokenEndTime	300
brokenStartTime	270
cycleTime	300

Figure 6: Typical simulation configuration.

The C code for a cyclical disconnection in *DECAP* state is (Similar code is also implemented in *ENCAP* state.):

```
/*If it is time to break the connection, then
* intercept the packets and destroy them.
* Otherwise forward it to next layer.*/
op_pk_nfd_access(pkptr, "fields", &tcp_fd_ptr);
if(!(tcp_fd_ptr->syn || tcp_fd_ptr->fin)
   && isBroken() )
{
   ip_encap_pk_destroy(pkptr);
}
else
{
   op_pk_send (pkptr, output_strm);
}
```

The *m_ip_encap* state also allows "recovery packet" to reach the FH when the connection is restored. When the MH recovers from a disconnection, it will send an ACK packet as a "recovery packet" to FH to restore the connection. In our implementation, this ACK packet contains the last ACK number with *rwnd* (usually greater than zero).

3.4 Modifications to Super Host

Since we use a WLAN Ethernet router as an SH to directly connect FH and MH, there is no data stream passed to layers higher than IP. We modified the *ip* and *ip_encap* processes so that packets will always be directed to higher layers such as M-TCP. Figure 7 shows the node model of the modified SH, which has the same structure as the OPNET standard model *wlan_ethernet_router*. Modifications are made in processes *ip* and *ip_encap*. We also created a new process model *m_tcp*.

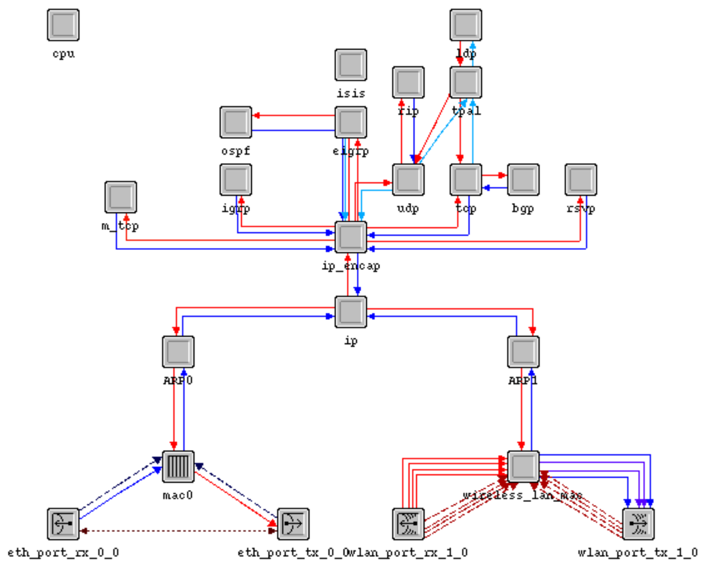


Figure 7: Modified Node Model of Super Host.

3.4.1 IP Routing Process

Routing is performed in the *ip* process. The structure of a standard routing module is shown in Figure 8. Routing task is performed in *ip_dispatch* and *ip_rte_central_cpu.* If the packet has reached its destination node, the routing part will forward it to the higher layer such as TCP and UDP. In a router, however, the packets are routed directly through the *ip_rte_central_cpu* module. We modified the *ip_rte_central_cpu* and let all packets pass through *ip_encap* to reach M-TCP process.

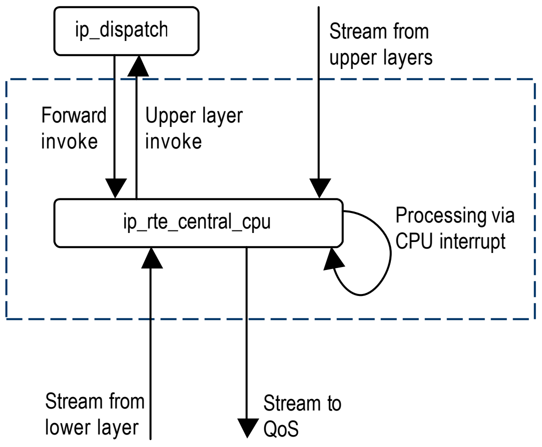


Figure 8: Centralized IP Routing Module [7].

The core code to re-route packets is:

```
static void ip_rte_central_cpu_packet_arrival()
{
    /*Packet arriving from a stream*/
  instrm = op_intrpt_strm ();
  pkptr = op_pk_get (instrm);
  if (pkptr == OPC_NIL)
    ip_rte_cpu_error ("Unable to get packet from
      the input stream.");
  if(((instrm == 1 ) // From lan arp0
    || (instrm == 2)) // From wireless arp1
    && (op_id_self() == 723)) // It is a router
  {
    printTrace("Directly sending packet to
      ip-encap");
    //send to ip_encap stream
    op_pk_send( pkptr, 0 );
    FOUT;
  }
  }
```

3.4.2. IP Encap Process

The original *ip_encap* process is able to decapsulate IP packets and to encapsulate packets from higher layers. Since our M-TCP is only a TCP-like layer above *ip_encap*, we bypass most functions in *ip_encap* and let the packets pass through unchanged. The process model is similar to the model shown in Figure 5, except that we modified *ip_encap* for the different purpose from the modifications made in *ip_encap* of MH described in Section 3.3. The additional code in the *ENCAP* state is:

```
// Obtain the packet arriving from a higher
// protocol layer.
input_strm = op_intrpt_strm();
pkptr = op_pk_get (input_strm);
if(pkptr == OPC_NIL)
   ip_encap_error ("Unable to get packet from
      transport layer.");
if(input_strm == INSTRM_FROM_MTCP)
   printTrace("ENCAP sends packet to ip layer");
// Send the packet to IP layer
op_pk_send_forced(pkptr, outstrm_to_network );
goto en1;
}
```

3.4.3 M-TCP Process

We implemented the M-TCP process model and inserted it into the transport layer of the SH. Figure 9 shows the M-TCP state diagram. After initialization (*INIT* state), the M-TCP process remains in *WAIT* state. When a packet arrives, a stream interrupt is issued, and M-TCP proceeds to *HANDLE* state where the packet's source address is checked. The FH and the MH packets are handled separately. *HANDLE* state for FH and MH packets are described in Figures 10 and 11, respectively. If retransmission timer expires, such as in the case of MH disconnections, *TIME_OUT* state is triggered. The details of *TIME_OUT* state is shown in Figure 12.

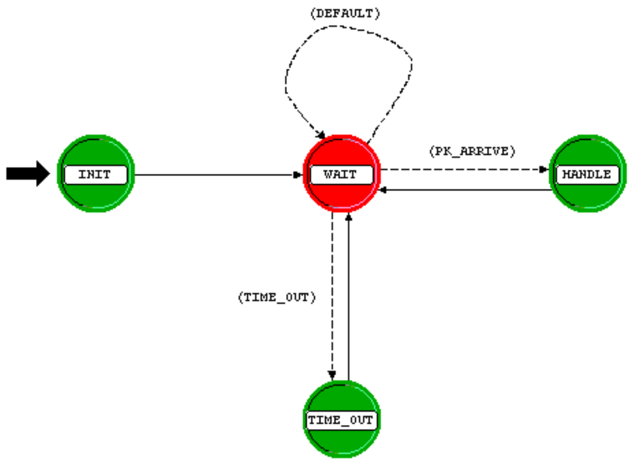


Figure 9: M-TCP state diagram.

The top-level logic in *HANDLE* state is:

```
analyse_incoming_pkt( ip_pkptr );
if(isFromFixHost())
   handle_Fix_Host_Pkt ( ip_pkptr );
if(isFromMobileHost())
   handle_Mobile_Host_Pkt( ip_pkptr );
```

If packet arrives from an FH, it is handled as shown in Figure 10. M-TCP maintains a packet queue in order to retransmit packets from SH, as opposed to retransmit from FH when packets are lost between SH and MH. A packet is removed when its ACK is received from MH. The queue structure that stores unacknowledged packets is:

```
typedef struct
{
  Packet *    pkt;
  Unsigned    seq_num;
  Unsigned    data_len;
  Double      arr_time;
  Double      time_out;
} QueueItem;
```

For the newly arrived packet, M-TCP first checks if it is a new packet. A new packed is queued in the received packet queue.

In Figure 10, state SHRINK is defined as the state during MH disconnection. Otherwise, M-TCP stays in the NOSHRINK state. In the NOSHRINK state, the timer should be set to the value of the RTO between SH and MH. Calculation of the accurate RTO in real time may be difficult. For simplicity, we have not implemented the RTO calculation and have used a constant RTO value of 0.4 sec. (The average value from our simulations was approximately 0.5 sec.) Finally, after the packet is queued, it is forwarded to MH.

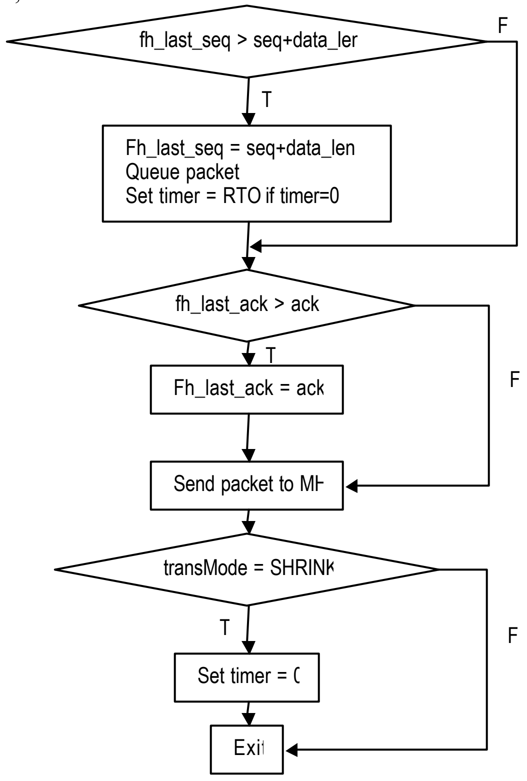


Figure 10: Handling packets from Fixed Host.

If a packet arrives from MH, it is handled as shown in Figure 11. If the packet contains a duplicate ACK number, M-TCP checks the packet queue. If the queue is empty, M-TCP forwards the packet to FH. This case rarely happens because it implies that a packet is lost in the wireline network. If this is a new ACK packet, M-TCP changes from SHRINK to NOSHRINK state because the new ACK number indicates that MH could have reconnected and replied to the packet sent by the SH. If the packet contains a new ACK number, the newly acknowledged packet in the queue is purged, and the last ACK number reduced by one is sent to FH.

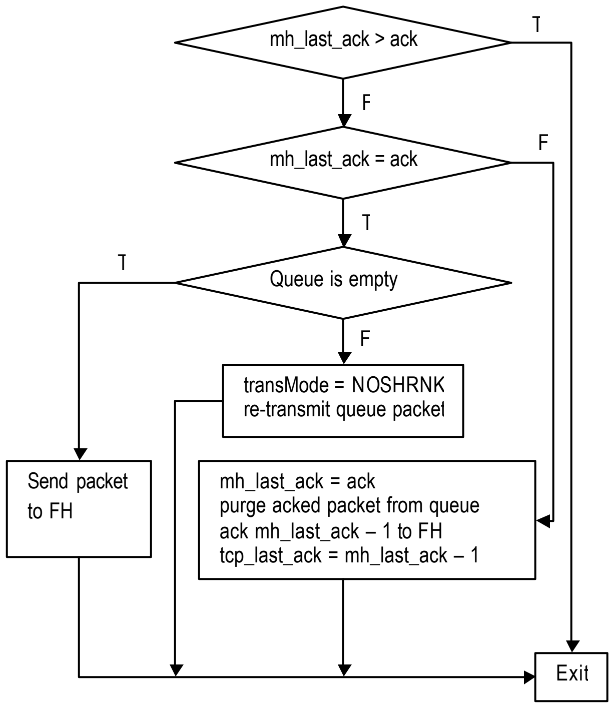


Figure 11: Handling packets from Mobile Host.

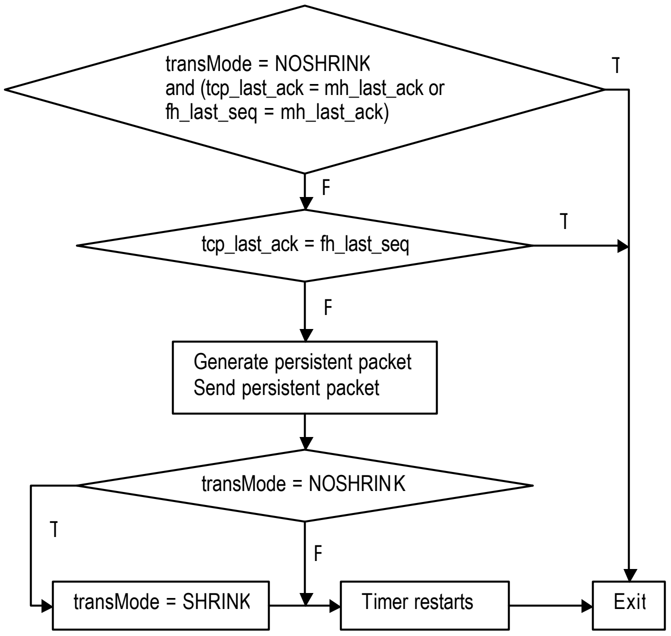

transMode = NOSHRINK
and (tcp_last_ack = mh_last_ack or
fh_last_seq = mh_last_ack)
T
F
tcp_last_ack = fh_last_seq
T
F
Generate persistent packet
Send persistent packet
transMode = NOSHRINK
T
F
transMode = SHRINK
Timer restarts
Exit

Figure 12: Handling Time-Out.

The timeouts are handled as shown in Figure 12. In order to handle timeouts, the last ACK number should always be kept. When there are ACK packets received from MH, M-TCP acknowledges to FH only the last ACK number reduced by one. In case of retransmission timeouts at SH, M-TCP generates and sends a persist packet to FH using this last kept ACK number, along with *rwnd* set to zero. This will trigger FH into *persist* state. Finally, before M-TCP exits *TIME_OUT* state, the SH state is set to SHRINK because the timeout suggests that MH is disconnected. The timer starts again. This is unnecessary if there is a mechanism in MH to re-establish the connection and to generate recovery packet. However, the OPNET WLAN Workstation model does not have such a mechanism. Instead, un-acknowledged packets are retransmitted to MH periodically. MH is forced to respond to these packets if it is connected. Hence, when MH is re-connected, a reply from MH will force SH to exit persist state.

4. Performance Comparisons

4.1 Simulation Scenario

We used three simulation scenarios to evaluate the performance of M-TCP. The objective of our simulations is to compare the size of the per-connection congestion window. Hence, our simulation scenarios consist of a single connection between MH and FH. The application used in this connection is FTP. FTP performance is observed in the FTP server (residing in FH), under three simulation scenarios.

Scenario 1. Regular TCP without disconnections: This is the ideal network situation without any simulated disconnections between MH and FH. It serves as the baseline for comparisons with the other two simulation scenarios.

Scenario 2. Regular TCP with disconnections: Regular TCP protocol is used in both hosts and the router. Disconnections are scheduled every 5 min. Each disconnection time is 30 sec at the end of each 5-min disconnection period. 30 sec is chosen because if the disconnection time is longer than 30 sec in Scenario 1, the connection will be closed by the FH. In order to provide comparable result with Scenario 1, we choose 30 sec as the duration of each disconnection.

Scenario 3. M-TCP with disconnections: We use similar setup as in Scenario 2, except that M-TCP is deployed in SH and MH. Disconnections are scheduled every 5 min. Each disconnection time is scheduled for 30 sec at the end of each 5-min disconnecting period.

In all three simulation scenarios, FH sends to the network the same FTP data stream: a large file is transferred from FH to MH. The FTP file is large so that FH will always have data to send.

4.2 Simulation Results

4.2.1 Congestion Window Size

Figure 13 shows the comparison of the congestion window size for FH. The M-TCP performance is very close to the regular TCP without disconnections. A few horizontal segments in the M-TCP graph indicate the periods when FH is in *persist* state (congestion window is frozen). In contrast, the congestion window in case of regular TCP with disconnections decreases drastically during the disconnection period and is unable to reach the value in M-TCP. This suggests that M-TCP helps prevent congestion window from decreasing during disconnections.

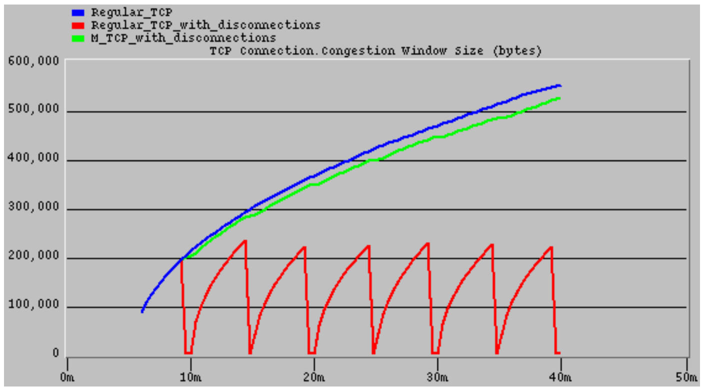


Figure 13: Comparison of Congestion Window Sizes: M-TCP has a larger congestion window than regular TCP with disconnections.

4.2.2 Goodput

Figure 14 illustrates the goodput for three simulation scenarios. The top line is the regular TCP in an ideal network. The middle line is M-TCP with simulated disconnections and the bottom line is regular TCP with simulated disconnections. It illustrates that in a network with disconnections, M-TCP has a higher goodput than regular TCP. This is expected because M-TCP tends to have larger congestion window size that enables more data to be transmitted. Moreover, even though each disconnection period lasts only 30 sec, the performance difference is still visible. In deployed mobile network environment, the hand-off time may range from 10 sec to a few minutes [4]. If we have taken into account the effects of retransmission timer, the difference in goodput would have been much greater. The longer the disconnection is, the larger number of packets will be lost and the longer the retransmission timeout will last. Furthermore, the longer the retransmission timeout, the longer FH waits to try to probe the connectivity of the network, and the longer it takes for FH to re-establish the transmission each time network recovers.

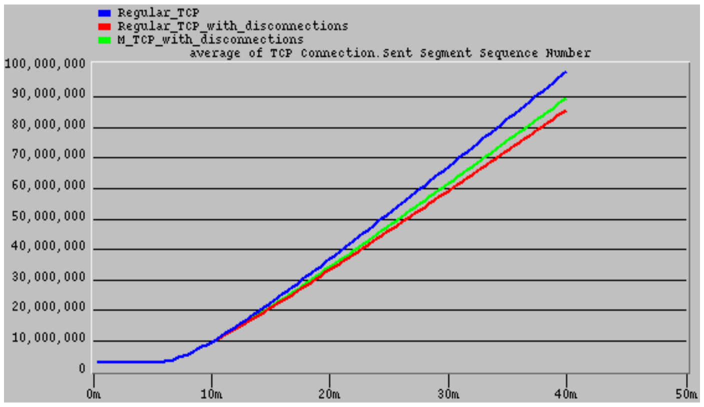


Figure 14: Comparison of goodput: M-TCP has higher goodput than regular TCP with disconnections.

4.2. 3 Retransmission Timeout (RTO)

Figure 15 indicates the differences in RTOs for three simulation scenarios. M-TCP has similar RTO as the regular TCP without disconnections, while the regular TCP with disconnections has a much higher RTO. This is due to the fact that MH is disconnected from FH during regular TCP disconnections, while the FH is shielded from seeing disconnections in case of M-TCP.

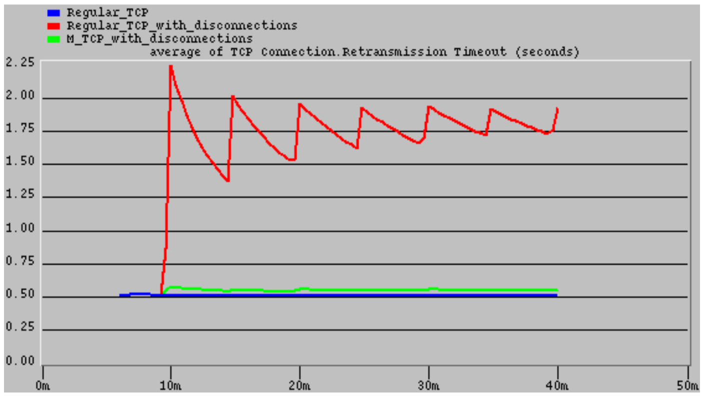


Figure 15: Comparison of RTOs: M-TCP achieves a much smaller RTO than regular TCP with disconnections.

5. Conclusions

In this paper, we described OPNET implementation of M-TCP protocol for use in mobile networks. M-TCP is designed to handle frequent disconnections, caused by signal fading and handoffs. We described simulation scenarios employed to evaluate the M-TCP performance. The simulation results show that in case of disconnections, M-TCP outperforms regular TCP in terms of congestion window size, goodput, and retransmission timeout. M-TCP's performance illustrates its ability to reduce the effect of the periodical network disconnections in wireless networks.

5 References

[1] W. Stevens, *TCP Illustrated*, Volume 1. Reading, MA: Addison-Wesley, Professional Computing Series, 1984.

[2] A. Bakre and B. R. Badrinath, "I-TCP: indirect TCP for mobile hosts," in *Proceeding of 15th International Conference on Distributed Computing Systems (ICDCS)*, Vancouver, Canada, May 1995, pp. 136-143.

[3] J. Border, M. Kojo, J. Griner, G. Montenegro, and Z. Shelby "Performance enhancing proxies," Internet Draft: http://community.roxen.com/developers/idocs/drafts/draft-ietf-pilc-pep-04.html (accessed June 2003).

[4] K. Brown and S. Singh, "M-TCP: TCP for mobile cellular networks," *ACM SIGCOMM Computer Communication Review*, vol. 27, no. 5, pp. 19-42, October 1997.

[5] K. Brown and S. Singh, "A network architecture for mobile computing," in *Proceeding of IEEE INFOCOMM*, San Francisco, CA, March 1996, pp. 1388-1396.

[6] S. Singh, "Quality of service guarantees in mobile computing," *Journal of Computer Communications*, vol. 19, no. 4, pp. 359-371, April 1996.

[7] OPNET documentation V.8.0.B, OPNET Technologies Inc., Washington DC.