An Adaptive TCP Protocol for Lossy Mobile Environment

Choong Seon Hong[1], YingXia Niu[1], and Jae-Jo Lee[2]

[1] School of Electronics and Information, Kyung Hee University
Korea 449-701
{cshong, niuyx}@khu.ac.kr
[2] Korea Electrotechnology Research Institute
Korea 437-808
jjlee@keri.re.kr

Abstract. TCP has been designed and tuned as a reliable transfer protocol for wired links. However, it incurs end-to-end performance degradation in wireless environments where packet loss is very high. TCP HACK (Header Checksum Option) is a novel mechanism proposed to improve original TCP in lossy links. It presents an extension to TCP that enables TCP to distinguish packet corruption from congestion in lossy environments. TCP HACK performs well when the sender receives the special ACKs correctly, but if many ACKs are also lost, the efficient of TCP HACK will not be prominent. In this paper we present an extension to TCP HACK, which can perform well even if the ACKs are severely corrupted. We use OPNET to simulate our proposal. The results have shown that our proposal performs substantially better than TCP HACK when corruptions occur on both data transmission path and acknowledgement path.

1 Introduction

Recent years, supporting Internet service over wireless network is a hot issue that has attracted many researchers to develop enhancements. Many applications are built on top of TCP, and will continue to be in the foreseeable future. So the performance of TCP in wireless environments has received much attention in recent years. The transmission control protocol (TCP) has been designed, improved and tuned to work efficiently on wired networks where the packet loss is very small. Whenever a packet is lost, it is reasonable to assume that congestion has occurred on the connection path. Hence, TCP triggers congestion recovery algorithms when packet loss is detected. These algorithms work reasonably well as the assumption on packet losses remains valid in most situations. However, in the wireless Internet environment, the bit error rate is much higher. As a result, the assumption that packet loss is (mainly) due to congestion is no longer valid. And the original TCP cannot work well in a heterogeneous network with both wired and wireless links.

Many protocols have been proposed to improve the performance of TCP over wireless links, but most of them need to use an intermediary node (such as base station) to modify TCP. So they cannot maintain an end-to-end TCP. TCP HACK (Header Checksum Option) [12] is a true end-to-end protocol proposed to improve the performance of TCP over lossy links. TCP HACK can work well when there are bursty errors on the data transmission path but no corruption on the acknowledgement path.

In this paper, we propose an extended TCP HACK that can work well even when there are much corruptions on the acknowledgement path.

The rest of this paper is organized as follows: Section 2 gives the problems of original TCP over lossy links; In section 3, we introduce the existing proposed solutions, indicating their strengths and weaknesses; our extended TCP HACK is proposed in section 4; simulation comparisons and drawbacks of our protocol are also presented. Concluding remarks are given in section 5.

2 Problems of TCP over Lossy Links

The original TCP detects missing ACKs via three duplicate acknowledgements or time-outs. When high bit error rates occur, even if a single packet loss will be considered that congestions have happened, it just could not distinguish between the congestion and packet loss. Then the TCP sender will trigger slow start mechanism: it drops the congestion window down to 1, and first grows it by a factor of 2 each time an ACK is received, until it reaches half of the threshold of the congestion window. Fig.1-1 shows the process of slow-start mechanism after a packet loss is detected, Fig.1-2 shows if we can distinguish the packet loss from congestion and immediately retransmit the lost packet, the slow-start will be avoided. The comparison between these two figures indicates that the wrong assumption drastically decreases the performance of TCP in the cases where bit error rates are high.

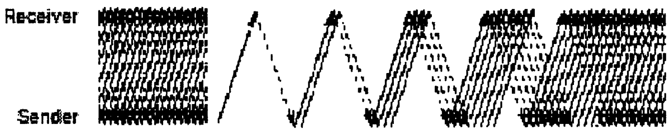

Fig. 1.1 Slow-start algorithm is triggered after packet loss in detected

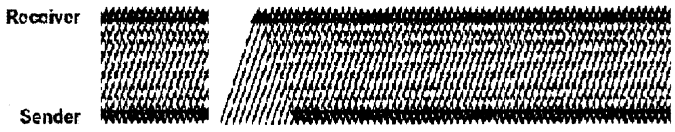

Fig. 1.2 Avoiding to trigger slow-start when detected packet loss

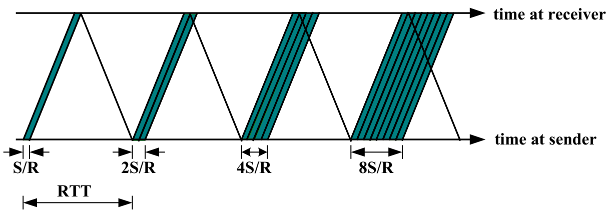

Fig. 2. TCP timing during slow-start

Fig.2. shows the TCP timing during slow start. S means the MSS (maximum size segment) is S bits; RTT is the round-trip time (including the transmission time of the packet) and R denotes the transmission rate of the link from the sender to the receiver. So the idle time (approximate) of the sender in slow start stage is given by:

Idle time=P[RTT]-(2^P-1)S/R

In which P=min {Q, K-1}, K is the number of windows that cover the object and Q is the number of times the sender would stall. For details, please read [15].

If the slow-start scheme is not triggered, then the number of extra packets that can be transferred is approximate given by [10]:

Extra Segments=W^2/8+WlgW-5W/4+1;

W is the unACKed packets can be sent in a congestion window.

It should be noted that the upper two expressions are under the assumptions that no corruptions in the traffic and ignoring all protocol header overheads.

3 Strengths and Drawbacks of Existing Solutions

Now the existing TCP implementation normally conforms to one of the four protocols: TCP Tahoe; TCP Reno; TCP NewReno and TCP SACK.

TCP Tahoe [3] is the original protocol that has Slow-Start, Congestion Avoidance, and Fast Retransmit algorithms.

TCP Reno [3] introduces a fast recovery algorithm to TCP Tahoe. After the fast retransmission algorithm sends what appears to be the missing segment, the fast recovery algorithm sets the congestion window to a half of its current window, and invokes congestion avoidance from a halved congestion window, not like TCP Tahoe that sets the congestion window to the smallest value. The reason that doesn't trigger the slow-start algorithm is because the receipt of the duplicate ACKs not only indicates that a segment has been lost, but also that the receiver can still receive the segments.

TCP NewReno [20] improves Reno with a Partial Acknowledgment algorithm. When TCP enters the Fast Recovery, it records the highest sequence number. If a new acknowledgment arrives during the Fast Recovery but does not cover the highest sequence number, TCP evaluates it as a Partial Acknowledgment and anticipates that more packets are lost.

TCP Tahoe, Reno, and NewReno will experience poor performance when multiple packets are lost from one window of data. With the limited information, A TCP sender can only learn about a single lost packet per round trip time.

TCP SACK [11], a Selective Acknowledgment mechanism, combined with a selective repeat retransmission policy, can help to overcome this limitation. The receiving TCP sends back SACK packets to the sender informing the sender of data that has been received. The sender can then retransmit only the missing data segments. But TCP SACK is still constrained by the tiny congestion window.

Recently, many protocols have been proposed to alleviate the poor end-to-end TCP performance in the heterogeneous network environments. These mechanisms can be mainly divided into two classes: one class needs to use Performance Enhancing Proxies (PEPs) [18]; the other class is end-to-end mechanisms that do not require TCP-level awareness by intermediate nodes. The first class includes Indirect TCP (I-TCP), Snoop TCP, New Snoop TCP, Mobile TCP (M-TCP), and Explicit bad-state notification (EBSN) etc; the second class includes TCP HACK etc.

In the following, we will describe the mechanisms of these proposals and see the strengths and drawbacks of each solution.

I-TCP [5] suggests that any interaction from a mobile host (MH) to a host on the fixed network (FH) should be split into two separate interactions - one between the MH and the Base Station (BS) over the wireless medium and the other between the BS and the FH over the fixed network. The data sent to MH is received, buffered and ACKed by BS. It is then the responsibility of BS to deliver the data to MH. The BS communicates with the MH on a separate connection using a variation of TCP that is tuned for wireless links and is also aware of mobility. By using I-TCP it can achieve better throughputs than standard TCP. But it does not preserve end-to-end semantics of TCP because the BS partitions the TCP connection.

The Snoop protocol [6] introduces a module, called snoop agent, at the base station. The agent monitors every packet that passes through the TCP connection in both directions and maintains a cache of TCP packets sent across the link that has not yet been acknowledged by the receiver. The snoop agent retransmits the lost packet if it has cached and suppresses the duplicate acknowledgments (ACKs). The Snoop module performs extremely well in high BER environments and maintains end-to-end TCP semantics. But it needs the intermediary (the Base Station) to do TCP modifications.

The New Snoop protocol [7] was proposed to overcome the shortcomings of Snoop protocol. It uses a two-layer hierarchical cache scheme. The main idea is to cache the unacknowledged packets at both Mobile Switch Center (MSC) and Base Station (BS), thus forming a two-layer cache hierarchy. If a packet is lost due to transmission errors in wireless link, the BS takes the responsibility to recover the loss. If the loss/interruption is due to a handoff, the MSC performs the necessary recovery. With this proposed hierarchical cache architecture, New Snoop protocol can effectively handle the packet losses caused by both handoffs and link impairments. But both Snoop protocol and New Snoop protocol need the intermediary (Such as a Base Station) to do TCP modifications and New Snoop even needs the MSC's participation.

The M-TCP [8] has the same goals as I-TCP and snoop TCP: to prevent the sender window from shrinking if bit errors or disconnection but not congestion cause current problems. M-TCP wants to improve overall throughput, to lower the delay, to maintain end-to-end semantics of TCP, and to provide a more efficient handover. It splits the TCP connection into two parts as I-TCP does. An unmodified TCP is used on the standard FH-BS connection, while an optimized TCP is used on the BS-MH connection. The BS monitors all packets sent to the MH and ACKs returned from the MH. And it retains the last ACK. If the BS does not receive an ACK for some time, it assumes that the MH is disconnected. It then shut down the TCP sender's window by sending the last ACK with a window set to zero. Thus, the TCP sender will go into persist mode. The M-TCP approach does not perform caching/retransmission of data via the BS. If a packet is lost on the wireless link, it has to be retransmitted by the original sender. This maintains the TCP end-to-end semantics. But it still requires a substantial base station involvement.

An explicit bad-state notification (EBSN) scheme [9] does not split the connection in two connections, it uses two types of acknowledgments: one is a partial acknowledgment informing the sender that the packet had been received by the base station and the other is a complete acknowledgment which has the same semantics as the normal TCP acknowledgment, i.e. the receiver (MH) received the packet. So it can distinguish the losses on the wired portion from the losses on the wireless link. Now the base station is responsible for retransmissions on the wireless link, while it delays timeout at the sender by sending a partial acknowledgement. This idea is that these explicit notifications prevent the sender from dropping congestion window. It also requires an intermediate node to modify TCP.

TCP HACK (Header Checksum Option) [12] is a solution based on the premise that when packet corruption occurs, it is more likely that the packet corruption occurs in the data and not the header portion of the packet. This is because the data portion of a packet is usually much larger than the header portion for many applications over typical MTUs. It introduced two TCP options: the first option is for data packets and contains the 1's-complement 16-bit checksum of the TCP header (and pseudo-IP header) while the second is for ACKs and contains the sequence number of the TCP segment that was corrupted. These "special" ACKs do not indicate congestion in the network. Hence, the TCP sender does not halve its congestion window if it receives multiple "special" ACKs with the same value in the ACK field. With this scheme, TCP is able to recover these uncorrupted headers and thus determine that packet corruption and not congestion has taken place in the network. TCP HACK performs substantially better than both TCP SACK and NewReno in cases where burst corruptions are frequent.

4 Our Proposed Solution: Extended TCP HACK

4.1 Requirements for Enhancing the Performance of TCP

Our goal in developing a new TCP protocol is to provide a general solution to the problem of improving TCP's efficiency for lossy links. Specifically, we want to design a protocol that has the following characteristics:

- It should preserve end-to-end TCP semantics. I-TCP and EBSN are not end-to-end-semantics. So, if a sender receives an acknowledgement, it assumes that the receiver got the packet. Receiving an acknowledgement now only means (for the mobile host and a correspondent host) that the foreign agent received the packet. The correspondent node does not know anything about the partitioning. Thus a crashing access node may also crash applications running on the correspondent node assuming reliable end-to-end delivery.
- It should not require the intermediate node to do TCP modifications. I-TCP, Snoop, New Snoop, M-TCP and EBSN all need the intermediate node to do TCP modifications. So the intermediary will become the bottleneck and add the third point of failure besides the endpoints themselves.
- It can handle encrypted traffic. As network security is taken more and more seriously, encryption is likely to be adopted very widely. Finally, all efforts for snooping and buffering data in the intermediate nodes may be useless if certain encryption schemes are applied end-to-end between the correspondent host and mobile host. Using IP encapsulation security payload the TCP protocol header will be encrypted, so that the intermediate nodes may not even know that the traffic being carried in the payload is TCP. Furthermore, retransmitting data from the foreign agent may not work any longer because many security schemes prevent replay attacks and retransmitting data from the foreign agent may be misinterpreted as replay. Encrypting end-to-end is the way many applications go. Therefore, it is not clear how these schemes (I-TCP, Snoop, New Snoop, M-TCP and EBSN) could be used in the future.
- It doesn't need a symmetric routing. The protocols that need to modify TCP in an intermediate node usually require that traffic to and from the end mobile host is routed through the same intermediate node. But in some networks, data and ACKs can take different paths, so these schemes based on intermediary involvement cannot be accomplished and may result in non-optimal routing.
- It can handle high BER. I-TCP, M-TCP, Snoop, New Snoop, and TCP HACK all can handle high BER.

From the analysis above we can get a conclusion that the existing mechanisms that need the PEPs have many drawbacks. The adoption of these protocols in the future should be considered. So we pay attention to the TCP HACK protocol. We think it is a novel mechanism to improve the TCP over lossy links; we do some modification to TCP HACK to make it more efficient. The modified TCP HACK is called Extended TCP HACK.

4.2 Our Proposal: Extended TCP HACK

In TCP HACK, when the receiver receives a corrupted packet whose header is not corrupted, it recovers the packet sequence number from the header and sends a special ACK packet including that sequence. If the return path is lossless, the TCP sender can get the information in time. But since the return path carrying ACKs and special ACKs is not lossless, the special ACK conveying the information that the packet was corrupted might be lost, now what the sender could do is waiting for the timeout.

The idea of our extended TCP HACK comes from the structure of TCP SACK. In SACK, the SACK option is defined to include more than one SACK block in a single packet. The redundant blocks in the SACK option packet increase the robustness of SACK delivery in the presence of lost ACKs. So we try to send more than one sequence number in one special ACK packet. We add a buffer in the TCP receiver (here we call it s_buffer). Then we save all the recovered sequence numbers into the s_buffer. These sequence numbers have been recovered from those packets whose data are corrupted but the sequence numbers in headers can be recovered (these packets have been transmitted in the same window). Then in the HACK special ACK option, we acknowledge all these sequence numbers saved in the s_buffer. So if the last special ACK is lost, the sender can get the sequence number from the next special ACK and retransmit the corrupted packet. It should be mentioned that sequence numbers in the s_buffer would be cleared if the corresponding retransmitted packets have been received correctly or the timers expire.

In TCP HACK, it introduced two options: one is Header Checksum option; the other is the Header Checksum ACK option. In our proposal, we don't make any change in the first option (see Fig.3.), but the second option has been extended (see Fig.4.). The packet length of the special ACK is variable and the sequence number in the option can be more than one.

Kind=14	Length=4	1's complement checksum of TCP header and pseudo-IP header

Fig. 3. TCP Header Checksum option

Kind=16	Length: Variable
1st 32-bit sequence number of corrupted segment to resend	
…	
The nth 32-bit sequence number of corrupted segment to resend	

Fig. 4. Extended TCP header Checksum ACK option

The nth 32-bit sequence number means that now there are n sequence numbers in the s_buffer. When the receiver recovers first sequence number, it writes it into the s_buffer and sends a special ACK contains the 1st sequence number; when the receiver recovers the second sequence number, it writes it into the s_buffer and sends another special ACK contains the 1st and the 2nd sequence numbers, and so on.

Like the TCP HACK, with our proposed extended TCP HACK, we should also do modifications to the TCP sender, receiver and the ACK processing algorithms, but there are some differences from the TCP HACK. In the following, we will explain in details.

4.2.1 Modification to TCP Sender

When a segment is sent, the TCP sender first checks if there is header checksum option enabled. If there is not, it will continue as normal; otherwise, it calculates the header checksum of that segment and places it into the header checksum option and then continues as normal TCP. This modification is same with TCP HACK.

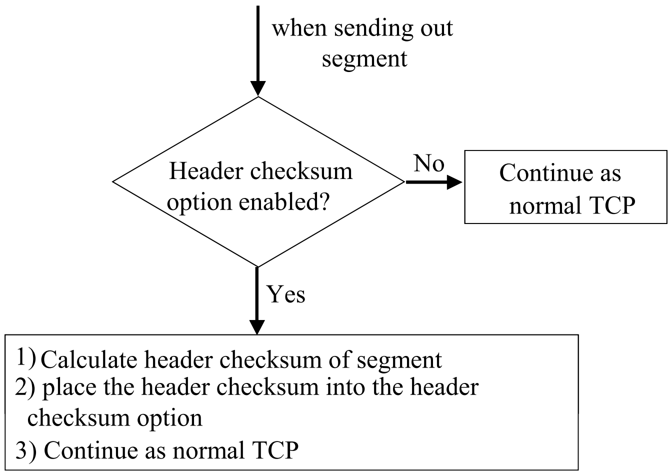
when sending out segment
Header checksum option enabled?
No
Continue as normal TCP
Yes
1) Calculate header checksum of segment
2) place the header checksum into the header checksum option
3) Continue as normal TCP

Fig. 5. Modification to TCP sender when sending a packet

4.2.2 Modification to TCP Receiver

When the TCP receiver receives a packet, first it checks the integrity of the packet using the standard checksum, if there is no error, then it works as normal TCP. If the packet is corrupted, then it will use the header checksum to check if the header portion has been corrupted. If the header portion is also corrupted, then the packet will be discarded, otherwise, the modified extended TCP does following:

1) Recover the sequence number form the packet header;
2) Save the sequence number into the s_buffer
3) Send a "special" ACK (option 16) to the sender. Its option contains all the sequence numbers that have been saved in the s_buffer. So the TCP sender can distinguish that this special ACK was generated because of packet corruption. And with these recovered sequence numbers, the TCP sender can selectively retransmit these corrupted packets.

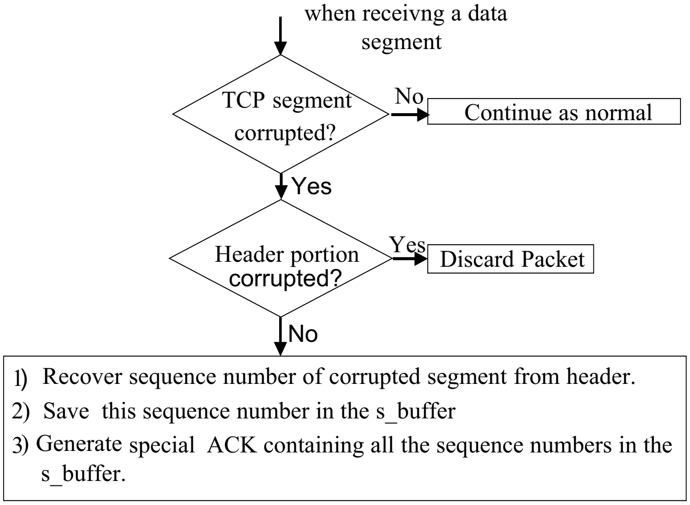
when receivng a data segment
TCP segment corrupted?
No
Continue as normal
Yes
Header portion corrupted?
Yes
Discard Packet
No
1) Recover sequence number of corrupted segment from header.
2) Save this sequence number in the s_buffer
3) Generate special ACK containing all the sequence numbers in the s_buffer.

Fig. 6. Modification to TCP receiver when receiving a packet

4.2.3 Modification to the ACK processing

When the TCP sender receives an ACK, first it checks if there is a HACK option (option 15) or an extended HACK option (option 16). If the received ACK is not a special ACK, the TCP sender will perform as normal TCP; otherwise, if it's a special ACK with option 15, TCP sender will work as TCP HACK, if the option kind is 16, the TCP sender does as follows:

1) Extract all the sequence numbers from the option of the special ACK
2) Retransmit Selectively these packets,
3) Discard the special ACK without further processing

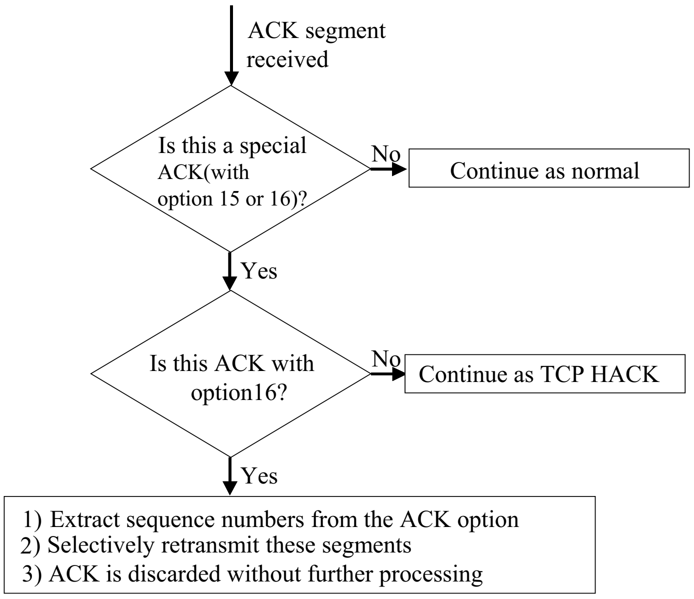
ACK segment received
Is this a special ACK(with option 15 or 16)?
No
Continue as normal
Yes
Is this ACK with option16?
No
Continue as TCP HACK
Yes
1) Extract sequence numbers from the ACK option
2) Selectively retransmit these segments
3) ACK is discarded without further processing

Fig. 7. Modification to the ACK processing

It should be mentioned that our extended TCP HACK could work together with normal TCP and TCP HACK. We implement the TCP HACK and extended TCP HACK with the original TCP. The user can choose to use any of them according to the transmission conditions.

4.3 Drawbacks of Our Proposal

There is one problem should be considered by using our proposed extended TCP HACK: the software overload. Using extended TCP HACK, more complex software will be on the sender and receiver sides, but while memory sizes and CPU performance permanently increase, the bandwidth of the air interface remains almost the same. Therefore the higher complexity is no real disadvantage any longer as it was in the early days of TCP.

4.4 Performance Evaluation

We carried our experiments by using OPNET modeler 8.0. The simulation model is shown as Fig.8. We ran our experiments by sending bulk data from the server to the client. The connect link between the server and client is a wired link but has been configured by different error bit rates to simulate the lossy wireless link. The link rate is set to 10M to simulate a wireless LAN and in order to avoid the congestion. The s_buffer is set to infinite not to be a limiting factor.

2%~15% packet loss was considered. The burst packets length is configured to 3 packets.

We also disabled the link layer CRC as TCP HACK did. So the corrupted TCP packet can arrive to the TCP stack.

We run experiments in two situations, one is corruption only on one direction and the other is corruption on both directions.

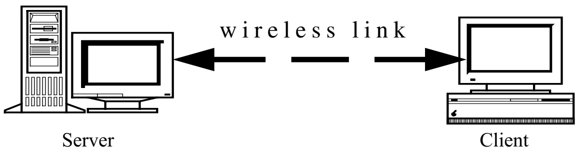

Fig. 8. Simulation model

4.4.1 Corruptions on One Direction

First, we run experiments in the condition that errors are only on the forward path; packets on the reverse path (the ACK packets) are not corrupted. Fig.9. shows the throughput for various packet loss rates. The result shows that TCP HACK and our extended TCP HACK all work better than the original TCP. When the packet loss rate is very high (more than 10%), the original TCP almost cannot receive correct packets, but using TCP HACK or extended TCP HACK, the receiver can still receive the packets

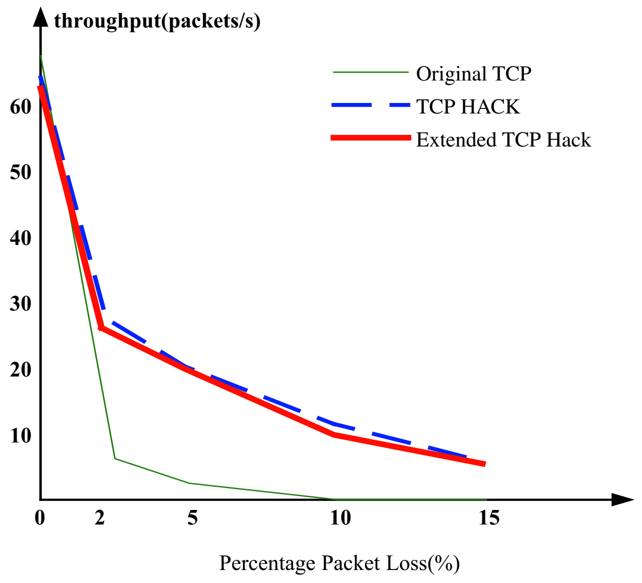

Fig. 9. Throughput for various packet loss rates when corruption on one direction

4.4.2 Corruptions on Both Directions

Next, we run experiments in the condition where corruptions on both directions (forward and reverse path). The acknowledgement packets have the same error rate as the data packets. From Fig.10 we can see that our extended TCP HACK performs much better in this condition. Because when there are many corruptions on the ACK path, the special ACKs in TCP HACK can't arrive at the sender, so the TCP HACK sender cannot get the loss information and cannot retransmit the corrupted packets.

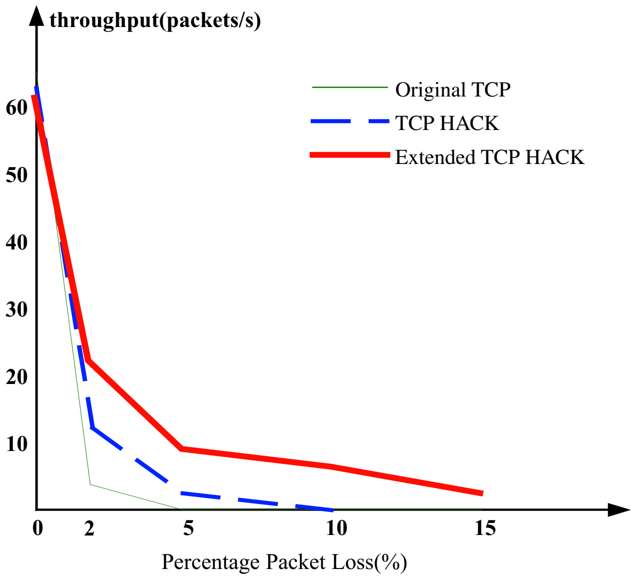

Fig. 10. Throughput for various packet loss rates when corruption on both directions

5 Conclusions and Future Works

In this paper, we analyzed the problems of original TCP over lossy links. Then we summarized the existing protocols, indicating their strengths and weaknesses. TCP HACK is one of these protocols, which has been proposed to improve original TCP over lossy links. We think that TCP HACK can be adopted in the future. But it has some drawbacks. So we proposed an extension to TCP HACK. Our proposal can enhance the TCP HACK in the situation where not only the data on the forward data are corrupted much, but also the ACKs on the inverse path susceptible to packet corruption.

Simulations have been done to test our extended TCP HACK protocol by using OPNET modeler 8.0. The results proved that our proposed protocol performs much better than original TCP in the cases that corruptions are big. In addition, the extended TCP HACK can improve the TCP HACK in the condition where packet loss not only on the forward path but also on the inverse path.

Works are being done to test the effectiveness of our proposal in situations where congestion, corruptions occur at the same time.

References

[1] Behronz A.Forouzan, "TCP/IP protocol suite" international editions 2000,271-311

[2] Jochen Schiller, "Mobile Communications" Pearson Education Limited 2000, 290-307

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

[4] Jiangping Pan,Jon W.Mark and Xuemin Shen, "TCP performance and its improvement over wireless links" GlobeCom2000

[5] Ajay Bakre and B.R. Badrinath, "I-TCP: Indirect TCP for mobile hosts", Tech. Rep., Rutgers University, May 1995.

[6] H.Balakrishnan, S.Seshan, Eamir, and R.H. Katz, "Improving TCP/IP performance over Wireless Networks", In Proc.1st ACM Conf. on Mobile computing and Networking, November 1995.

[7] Jian-Hao Hu,Kwan L.Yeung,Wiew Chee Kheong and Gang Feng, "Hierarchical Cache Design for Enhancing TCP over Heterogeneous Networks with Wired and Wireless Links" GlobeCom2000

[8] K.brown and S.Singh, "M-TCP: TCP for Mobile Cellular Networks", ACM computer Communications Review (CCR), vol.27, no.5, 1997

[9] N.Vaidya, Overview of work in mobile-computing (transparencies).

[10] Tom Goff, James Moronski, D.S.Phatak, "Freeze-TCP: a true end-to-end TCP enhancement mechanism for mobile environment" InfoCom2000

[11] M.Mathis, J.Mahdavi, S.Floyd, A.Romanow, "TCP selective acknowledgment options" IETF RFC 2018, 1996

[12] R.K.Balan, B.P.Lee,K.R.R.Kumar, L.Jacob,W.K.G.Seah, A.L.Ananda, " TCP HACK:TCP header checksum option to improve performance over lossy links", InfoCom2001

[13] Hari Balakrishnan, Venkata N. Padmanabhan, Srinvasan Seshan, Mark Stemm, Elan Amir, Randy H.Katz, "TCP Improvements for Heterogeneous Networks: The Daedalus Approach".

[14] Ramon Caceres and Liviu Iftode,"Improving the performance of reliable transport protocols in mobile computing environments" IEEE JSAC Special Issue on Mobile Computing Network,1994.

[15] James F. Kurose and Keith W. Ross, "Computer Networking" 2001 by Addison Wesley Longman, Inc. 167-260.

[16] Modeler & Radio powered by OPNET Simulation Technology. SIMUS Technologies. Inc.

[17] IT Decision Guru powered by OPNET Simulation Technology. SIMUS Technologies.Inc.

[18] IETF PILC WG homepage, http://www.ietf.org/html.charters/plic-charter.html

[19] J. Border , M. Kojo , J. Griner , G. Montenegro , Z. Shelby , "Performance Enhancing Proxies Intended to Mitigate Link-Related". IETF RFC 3135. June 2001.