An Extension to TCP HACK for Improving the Performance of TCP in Lossy Environments

Ying Xia Niu , Choong Seon Hong
School of Electronics and Information, Kyung Hee University
niuyx@networking.kyunghee.ac.kr, cshong@khu.ac.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 link. It presents an extension to TCP that enables TCP to distinguish packet corruption from congestion in losssy environments. TCP HACK performs well when the sender receives the special ACKs correctly, but if the ACKs are also lost much, the efficient of TCP HACK will not be prominent. In this paper we present an extension to TCP HACK, which can perform well even when the ACKs are much corrupted.

1 Introduction

Recent years, supporting Internet service over wireless network is a hot issue that has attractived many efforts. 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. Transmission control protocol (TCP) has been designed, improved and tuned to work efficiently on wired network 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 As a result, the assumption that packet loss is (mainly) due to congestion is no longer valid and the original TCP cannot work well in a heterogeneous network with both wired and wireless links.

In the following, we first summarize the existing proposed solutions, indicating their strengths and weaknesses, then introduce our proposal that can work well in lossy links where packets losses are in both forward and the reverse path.

2 Related Works

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 uses Performance Enhancing Proxies (PEPs) [7], the other class is end-to-end mechanisms that do not require TCP-level awareness by intermediate nodes. Protocols with PEPs have many drawbacks that should be considered for adoption in future versions of TCP. So in the following, we only introduces some protocols related to the second class.

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

TCP Reno [3] is on the top of Tahoe. It introduces a fast recovery algorithm to TCP Tahoe. After the fast retransmit 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. It doesn't set the congestion window to the smallest value like TCP Tahoe does. 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 [9] 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 [5], 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.

TCP HACK (Header Checksum Option) [6] 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 not congestion but packet corruption has taken place in the network. TCP HACK performs substantially better than both TCP SACK and NewReno in cases where burst corruptions are frequent. But the authors of [6] performed simulations where the errors were generated only to packets travel on the forward path. Packets on the reverse path was not corrupted

3 Proposed Scheme

The idea that proposes an extension to 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.

In TCP HACK, the receiver sends only one sequence number for corrupted data but sequence number can be recovered in one special ACK packet. 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 losses, so if the special ACK conveying the information that the packet was corrupted is lost, what the sender could do is to wait for the timeout.

So we propose to add a buffer in the TCP receiver, and save all the received packets that data are corrupted but the sequence numbers in headers can be recovered. Then in the HACK special ACK option, we acknowledge all these sequence numbers in the buffer. It should be mentioned that the TCP receiver does not receive these packets correctly yet.

3.1 Extension to TCP HACK options

In TCP HACK, it introduced two options: one is Header Checksum option and the other is the Header Checksum ACK option. In our proposal, we don't make any change in the first option (see Fig.1.), but the second option should be extended (see Fig.2.).

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

Fig.1. 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.2. Extended TCP header Checksum ACK option

3.2 Modification to TCP HACK

With our proposed extended TCP HACK, we add a buffer in the TCP receiver to cache the sequences of the segments that are data corrupted but header can be recovered, we call this buffer SeqBuffer. We should do modifications to the TCP HACK in the TCP sender when receiving a special ACK, and modify the TCP receiver when sending a special ACK. See Fig.

3. and Fig.4.

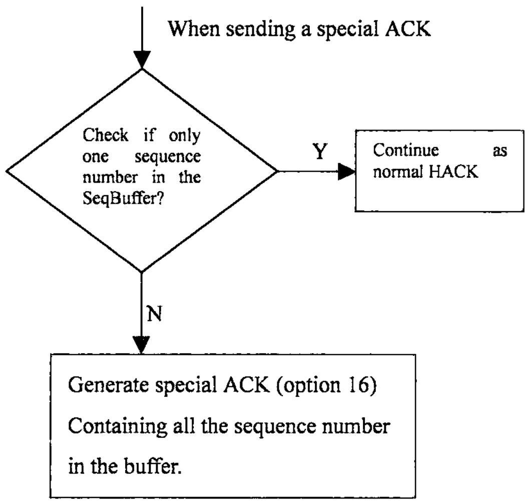
When sending a special ACK
Check if only one sequence number in the SeqBuffer?
Y
Continue as normal HACK
N
Generate special ACK (option 16) Containing all the sequence number in the buffer.

Fig.3. Modification to TCP receiver when sending a special ACK

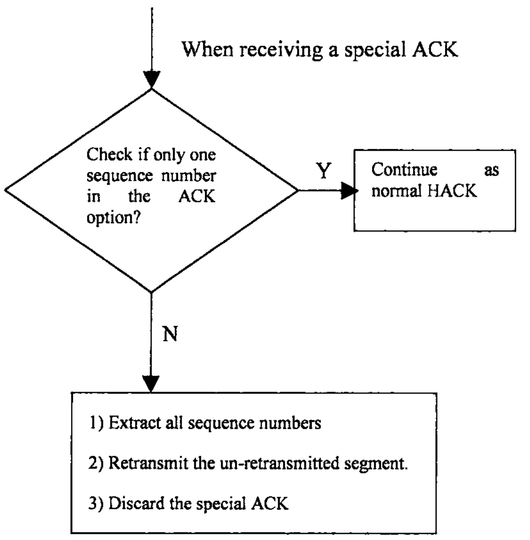
When receiving a special ACK
Check if only one sequence number in the ACK option?
Y
Continue as normal HACK
N
1) Extract all sequence numbers
2) Retransmit the un-retransmitted segment.
3) Discard the special ACK

Fig.4. Modification to TCP sender when receiving a special ACK

5. Drawbacks

The main drawback of our proposed protocol is 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.

6. Conclusions and Future Work

In this paper, we summarized the existing protocols, indicating their strengths and weaknesses, and 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 are also susceptible to packet corruption.

Simulations are being done to test our proposal by using OPNET modeler 8.0.

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

[6] 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

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

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

[9] S.Ployd, T.Henderson, "The NewReno Modification to TCP's Fast Recovery Algorithm". RFC 2582

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