LHP: An end-to-end reliable transport protocol over wireless data networks

Xia Gao, Suhas N. Diggavi, S. Muthukrishnan

Abstract—**The next generation wireless networks are posited to support large scale data applications. Implementing end-to-end TCP in such networks faces two problems. First, it is well known that TCP can not distinguish packet losses due to link failures and that due to network congestion. Second, TCP congestion control mechanism does not deal effectively with large amount of out-of-order packet retransmissions; this problem has received less attention in literature. In this paper, we present solutions to both these problems. In particular, we present a Link-Layer Header Protection (LHP) protocol which implements Explicit Loss Notification (ELN) in a simple, scalable manner, addressing the first problem. We also modify the congestion control mechanism to incorporate knowledge of ELN and packet loss pattern into retransmission decisions, solving the second problem. We combine both these solutions with TCP to present scalable, reliable end-to-end wireless transport protocol.**

I. Introduction

The operation of TCP in wired-wireless (hybrid) networks has been an important research issue in recent years, due to the impressive growth of WLAN and WPAN technologies. To duplicate the success of TCP in the wireline world at present in the wireless future, there are two conceptual and technical challenges in meeting this goal:

1. The first problem is that of localizing the source of packet losses. This is a well documented problem [1]. In the wireless network, a significant portion of the packet losses are due to wireless link failures resulting from channel fading and interference; hence, packet losses are more likely to be a wireless link failure rather than congestion in the network. TCP attributes packet losses to congestion and takes corrective measures: adjusting congestion windows and dropping data rates, which ultimately degrades performance in wired/wireless networks.
2. The second significant problem is to deal with the large amount of packets being transmitted out-of-order arising from retransmission. This problem has received scant attention in the literature. Since channel loss rates are significant in wireless networks, transport protocols have to handle many more out-of-order packets than in the wired case resent due to the channel corruption. Original TCP congestion control mechanisms such as fast retransmit, fast recovery and round trip time (RTT) estimation behave poorly in face of these out-of-order packets.

Many proposals have been made to address the first challenge [1], [2], [7], [9]; they rely on being able to isolate the cause of packet loss to link loss or the congestion loss. No fully developed protocol appears to have been presented to address the second challenge. Our experiments reveal this to be a serious problem and neglecting it would significantly degrade transport performance.

In this paper, we present solutions to both these challenges; when combined with TCP, this results in an ultimately simple, scalable and efficient end-to-end transport protocol. In particular:

1. To address the first challenge, we propose to *protect* the link layer header. More specifically, we propose a new end-to-end protocol called *Link-Layer Header Protection* (LHP) to disambiguate between congestion losses and link failures. The LHP protocol is based on very simple principle: if the header packet is received by the mobile terminal, even if the IP packet is in error, the receiver would know that there was a wireless link loss. This information can then be conveyed back to the sender using a special acknowledgment (ACK); this is an instance of Explicit Loss Notification (ELN) [1].
2. To address the second challenge, we redesign the TCP congestion control mechanism to take into account ELN and the packet loss pattern in order to deliver the packets. In particular, this mechanism determines the right time to trigger the fast retransmit and fast recovery, by judiciously incorporating both the knowledge of the source of packet loss (through ELN) and the packet loss pattern through an appropriate acknowledgment scheme.

The rest of the paper is organized as follows. In Section II, we discuss related work on reliable wireless transport schemes proposed in literature and put our work in perspective. Section III describes the LHP protocol and its properties. In Section IV we describe in detail the issue of packet re-ordering and present our scheme. Section V provides the main simulation results, in particular, the effects of TCP augmented with our two solutions. We conclude in Section VI with a brief discussion.

II. Related work

As mentioned above, the key to improve TCP performance over wireless link is to ensure TCP avoids congestion control measures for non-congestion losses. There is a large body of research aimed at ensuring this and these schemes can be classified into four broad categories : link layer (LL) protocols, split-connection protocols, end-to-end protocols[1], and explicit notification protocols.

LL protocols use techniques such as automatic repeat request (ARQ), forward error correction (FEC) or their combination to provide the transport layer protocol with a dependable communication channel, similar in characteristics to a wired one. By hiding any losses due to the channel error from the transport layer protocol, LL protocols result in little change, if any, to the TCP. *Split-connection protocols*, such as Indirect TCP [2] and [7], isolate the transport protocol from channel errors by splitting each TCP connection into two connections at the base station: ordinary TCP is used at the wireline link and protocols tuned specifically for wireless operations is used at the other end. *End-to-end protocols* assume that the underlying network can only provide best effort, unreliable packet delivery service. TCP Selective acknowledgment [6] and its complement, TCP forward acknowledgment [8] allow acknowledgment to have more information about the multiple non-contiguous blocks of successfully transmitted data segments. This information is then used by the sender to avoid retransmitting segments whose successful delivery at the other end is not evident if cumulative acknowledgment is used. *Explicit notification protocols*, such as Explicit Congestion Notification (ECN) [3] and Explicit Link Failure Notification [4], provide TCP senders explicit signals, informing either congestion [3] or corruption [4], to differentiate the reason of packet loss. More detailed discussion of each protocol and main design tradeoff are discussed in [1] and are not repeated here. Our algorithms discussed later overcome some of the limitations possessed by these protocols and are summarized in section III.

Recently, TCP header checksum option (HACK) [9] has been proposed; this is first known end-to-end ELN protocol. By putting extra header checksum bits in the option field of the TCP header, header can be checked the correctness and then used to produce ELN message even when the data portion of the packet is corrupted. However, two problems exposed in section I still exist. When the packet error rate is high, packet headers will have significant proba-

X. Gao is with DoCoMo Communications Lab USA. (Email: gao@docomolabs-usa.com). S. Diggavi and S. Muthukrishnan are with AT&T Shannon Labs. (Email: {suhas, muthu}@research.att.com).

bility of getting corrupted so that ELN packet can not be produced. On the other hand, when the channel error rate is low and congestion window is big, because HACK lacks appropriate mechanisms to deal with duplicate acknowledgments due to the corrupted packets, the duplicate acknowledgments from the current window will force the sender to unnecessarily enter fast retransmit phase. To alleviate this problem, HACK needs to work with an acknowledgment scheme such as SACK which recognizes packet loss patterns. This in turn leads to the problem of dealing with out-of-order packets during retransmission, the second challenge we mentioned earlier. As far as we know, the interaction of TCP congestion control and out-of-order retransmitted packets has not been raised in literature.

III. LHP PROTOCOL AND TCP MODIFICATION

In this section we will describe our link layer header protection protocol, its properties and some related issues.

A. Our LHP protocol

The link layer part of our LHP protocol works as follows:

1. Get the packet of size S from link layer queue, divide it into l *fragments*. Each fragment has size of S_l.
2. Set the maximum link time slot assignment of the packet to be $l + c$ slots, where each slot is able to transmit one fragment. c is a tunable parameter, which influences the wireless delay guarantee.
3. Link layer uses stop-and-wait mechanism to transmit each fragment.
4. Link layer starts with transmitting the first fragment, as the packet header. If the transmission is successful, it moves on to transmit next fragment. If the transmission is unsuccessful, it will retransmit the header packet until the transmission is successful or the maximum link time slots is reached.
5. Link layer will use the default mechanism to transmit the remaining fragments.

The LHP protocol is based on a simple insight: *if the IP packet header is received, one knows the flow to which the corrupted IP packet belongs*. In a nutshell, LHP does unequal error protection, and gives highest priority to the IP packet header in Step 4 of the above protocol.

B. Properties

We compare the LHP with other protocols mentioned earlier.

- *Accurate ELN detection.* By giving header extra protection, LHP provides significantly better ELN accuracy than [9] where IP header integrity is just checked, and ECN [3] where ELN can not always be derived from ECN packets. For example, if each IP packets are divided into 6 fragments (1500 bytes per IP packet and 256 bytes per link layer fragment), the probability for the transmission of header packet fails in all 6 slots is 1e-12 for a link-level packet error rate of 1% (which translates to a 6% IP packet loss rate).
- *Combination of link layer mechanism, ELN, and end-to-end transport layer protocol.* Such combination has proved to be one of the best for wireless transport protocols [1] and achieves better performance than pure end-to-end protocols such as [6] and [8]. Maintaining end-to-end semantics reduces the service load of base station compared with [2] and [7], and avoids the problem of understanding the TCP/IP header which may not be easy if IPsec is used to enhance the security of wireless communications compared with [1] and [4].
- *Portability.* Our link-level protocol does not need to be TCP-aware, and hence can be used with other transport protocols as long as the packet header is at the beginning of the packet compared with [1] and [4].
- *Scalability.* Unlike [1], each packet of different flows is treated the same way because position of the header in each packet is relatively fixed. Hence LHP does not need to keep per flow state.
- *Lower delay and bandwidth variation.* Moreover, based on the delay requirement of the application, extra time slot c can be statically or dynamically decided. If application is quite delay-sensitive, smaller c or even zero could be used. Or if application is loss-sensitive, higher c could be used. So channel bandwidth and round trip time deviations respectively in FEC and ARQ schemes could be alleviated. At the same time, adverse interaction between link layer and transport layer reliable mechanisms could also be avoided [5].

C. Issues

Two main concerns about the LHP protocol are the stop-and-wait retransmission and the packet fragmentation. Stop-and-wait is suitable when bandwidth-delay-product is low so the LHP protocol above is only good for LAN scenarios such as 802.11. To be used in the WAN scenario where bandwidth-delay-product is high, the LHP protocol needs to work together with active queuing schemes. In this way, the active queuing schemes can multiplex the transmission of different flows to avoid the waste of bandwidth and head-of-the-line blocking phenomenon. Other protection mechanisms, such as unequal FEC protection of the packet header, could also be used. Note that the main contribution of this part of our work is to propose the unequal error protection of packet headers so that ELN signal can be produced much more accurately. Then, the influence on congestion control of out-of-order packets due to the ELNs can be examined *independently*.

Packet fragmentation is another mechanism to combat transmission error. Because fragments with larger size have larger probability to be corrupted during the transmission but smaller fragment size produces more fragments for one packet which incurs more processing and signaling overhead, given a channel error rate, there usually exists an optimal fragment size which achieves maximal throughput. Hence, many of the wireless protocols are already equipped with packet fragmentation ability, which can be utilized by LHP. For example, 802.11, with the built-in stop-and-wait and packet fragmentation abilities, only needs small modification to support LHP protocol. The modification allows the partially received link layer packet to be uploaded to the network layer and then to the transport layer instead of being discarded. Then receiver transport layer could use the header option fields in the TCP ACK packets to send ELN back.

IV. CONGESTION CONTROL IN CASE OF OUT-OF-ORDER PACKETS

In this section we start with examples motivating the necessity of having a mechanism to deal with out-of-order packets. Then we describe our proposed scheme and explain its properties. We then summarize the proposed transport protocol along with the ELN scheme described in Section III.

A. Main Issues

The TCP control mechanism has two functionalities, "when to trigger the congestion control" and "what to send when congestion control algorithms have been triggered". Most of the later TCP enhancements such as SACK try to answer the second question by putting more information in the ACK sent by receiver to sender. Therefore, when the sender goes into fast retransmission period, it has enough information about which segments are lost during the transmission so that it is able to only retransmit lost packets. This is in contrast to beginning from the lowest unacked packets or the first unsent packet in current congestion window. When resent from the lowest unacked packet, many duplicates of successfully received packets are resent again (tahoe). When packets are sent from the right edge of the congestion window, the recovery speed of mul-

tiple drops will be slowed down to 1 packet per RTT (reno, newreno). While mechanisms such as SACK gives good answers to the second problem associated with recovery speed and duplicate retransmission, they use the same techniques, fast-retransmission and timeout, to trigger the congestion control because they assume all packet losses and therefore the duplicate ACKs are due to the congestion. In the presence of large link-level loss rates of the wireless networks, the side-effect of large out-of-order packet retransmission due to the ELNs makes it necessary to reconsider the first question of *when to trigger the congestion control*.

B. Examples

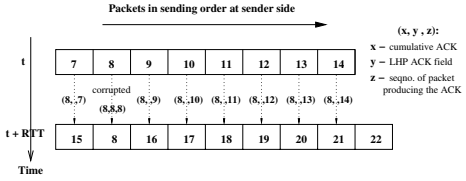

Fig. 1. One corruption loss, no congestion loss

Fig.1 shows the functionality of LHP. Assume the sender has congestion window size (cwnd) of 8 packets and is in the congestion avoidance period. Packet 8 gets corrupted during the transmission and the receiver sends back special ACKs to inform the sender to retransmit the packet. The format of the ACK is shown in Fig.1 and only first two parts, i.e., the cumulative ACK fields and LHP fields, are used in this part. Later, when the ACKs for the following packets 9, 10, 11 arrive, although all of their cumulative ACK fields are packet 8, the sender can avoid unnecessary fast retransmission by utilizing the information that packet 8 is just corrupted.

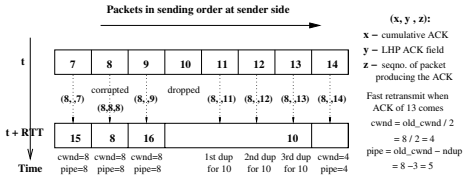

Fig. 2. One corruption loss, one congestion loss

While Fig.1 demonstrates two main benefits of using LHP, to allow retransmission of corrupted packets and to avoid unnecessary fast retransmission, Fig.2 shows its main limitation. Compared to Fig.1, Fig.2 has packet 10 dropped because of congestion. Because cumulative ack fields of ACK 11, 12, 13 are used for packet 8, the loss of packet 10 cannot be detected and recovered by fast retransmission. This motivates the usage of mechanisms such as SACK or SMART which let sender know the receiver packet pattern. The third fields in the special ACK, is used for this purpose. By knowing that ACKs have been triggered by packet 11, 12, and 13, the sender can infer that packet 10 is lost and enters the fast retransmission phase correctly. Therefore, SACK in combination with LHP can allow the sender to make the right congestion control and transmission choice.

Up to now, the updated congestion control mechanism can deal with the 4 combinations of packet corruption and congestion: *(a) only congestion (b) only corruption (c) corruption happening before congestion and (d) corruption happening after congestion.* Case (a) and (b) are handled separately by ordinary fast retransmit mechanism and LHP. Case (c) is handled as described above. Case (d) can be handled easily by using ordinary fast retransmit mechanism first and then LHP later. However there is one more observation shown in Fig. 3 suggests that further treatment is needed to tackle the interaction between congestion and corruption.

In Fig.3 packet 8 gets corrupted in the first transmission so LHP will enable the retransmission of packet 8 one RTT later. Then

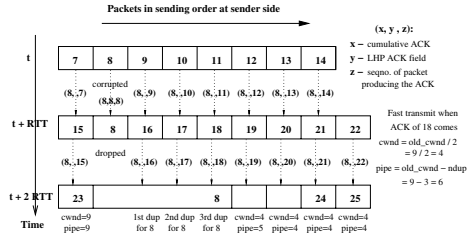

Fig. 3. Retransmission of corruption loss packet having congestion

if packet 8 encounters congestion and gets dropped, how can the sender distinguish this from packet 8 corruption in the previous round and make right decision on whether or not to enter the fast retransmission phase. Notice that ACK of packet 15 can not be treated as the duplicate ACK for 8 although it acks 8. This is because packet 15 was sent before packet 8 in the second round, so that the ACK of 15 should always ACK packet 8. On the contrary, the ACKs of packet 16, 17 and 18 should be treated as the duplicate ACKs for 8 because they are sent out after packet 8 in the second round, since their ACK reflect the information of packet 8. Then as shown in Fig.3, ACK of packet 18 can trigger the duplicate ACK mechanism and leads to the retransmission of packet 8. This key observation motivates the need for the sender *to keep the sending order of packets* in order to make this judgment. In the following section we will propose a simple algorithm to fulfill this goal.

Three examples shown above illustrates the problems incurred by out-of-order packets. Notice that ordinary TCP always sends out packets in sequential order in slow start or congestion avoidance period. Only when congestion happens and TCP goes into fast recovery period, does TCP send out-of-order retransmission packets. *So when ordinary TCP needs to make the judgment about whether to go into fast recovery period, it does not take the influence of out-of-order packets into account.* So the basic assumption is the out-of-order is rare and if it does exist, the three duplicate ACK will have enough redundancy to deal with it. Our observations show that this assumption is no longer valid in wireless domain given the frequent interaction between out-of-order packets and TCP congestion control mechanisms.

C. New fast retransmit and fast recovery mechanisms for lossy wireless links

Previous examples have shown the necessity to keep the information of sending order of packets besides the LHP and SACk fields for the sender to make the right choice when facing different combination of corruption and congestion. In Fig.4 we show the data structure consisting of a single list of packets in the order of sending time, to keep the necessary information. Each entry of the list has two data fields besides the pointer: sequence number of the packet and the duplicate ACK counter for this packet. The left side of Fig.4 shows the ACK information received by the sender. The right hand of Fig.4 shows the current list after the ACK is processed. We will illustrate our algorithm through this example.

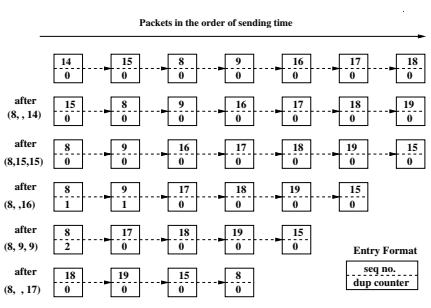

Fig. 4. Update of packet sent queue

The original list has packet 14, 15, 8, 9, 16, 17, and 18. Packet 8 and 9 are out of order because of retransmission. If the first ACK of packet 14 is received, the sender knows it is the normal ACK because packet 8 is under retransmission. So the sender removes the packet 14 from head and sends out a new packet 19. The new entry of packet 19 is inserted at the end of the queue. Next ACK of packet 15 is an LHP ACK which shows that packet 15 has corrupted at the link. So packet 15 gets retransmitted immediately. The entry of packet 15 is first removed from the head of the list and then is inserted at the end. Then ACK of packet 16 is received and by examining the figure we can see the packet 16 is an out-of-order packet because packet 8 and 9 were sent before packet 16 but have not received an ACK. The algorithm searches the list from the head to the position of the packet and for each entry it goes through, the duplicate counter is increased by one. So now, entries of packet 8 and 9 have duplicate counter of 1. Since this is a duplicate ACK, no new packet is sent out. Next, the (out-of-order) LHP ACK of packet 9 is received. Hence, duplicate counter of packet 8 will be increased to 2. Since this is a duplicate ACK, the retransmission of packet 9 is postponed. Finally when ACK of packet 17 reaches the sender, the duplicate counter of packet 8 reaches 3 which is the threshold to begin the fast retransmit. From this example one can see that this algorithm works robustly in all the scenarios. And the space requirement is linear in the size of congestion window. And when packets are transmitted in order, the only operations are the removal from the head and the insertion at the end which need constant time. So overall the space and amortized time complexity of this mechanism is small.

By maintaining such a list, another shortcoming of SACK can be overcome. In SACK, when the retransmitted packet got lost due to congestion or corruption, the sender has to rely on the timeout to detect such scenario [6]. But using the proposed list structure we can find such a gap and do the necessary retransmission.

D. Summary of our algorithm

The main goal of our algorithm is to *bring TCP's congestion control behavior closer to the goal of Additive Increase Multiplicative Decrease (AIMD) for wireless domain and decouple the interaction between congestion and corruption*. To achieve this, our algorithm has two interacting parts, i.e., the LHP which allows the TCP receiver to accurately produce ELN messages for the sender to retransmit the lost packets, and enhanced TCP congestion control algorithm to eliminate the side-effect of large number of out-of-order packets. Besides the changes we described above, after examining each congestion control components, we make some necessary changes such as the estimation of SRTT (smoothed RTT) and RTO, and usage of $pipe$ variable similar to [10] to track the packets on-fly in both cases of corruption and fast recovery . Except for these changes, we keep the other basic components, such as slow start and time-out, unchanged. More details and the pseudo-code of the algorithm can be seen in [11].

V. Performance Evaluation

In this section we briefly introduce the main simulation results of LHP and new congestion control algorithm because of page limitations. More extensive results can be found in [11].

A. Simulation Setup

The simulation is done in the ns-2 simulator , where we add the necessary components including link layer fragmentation and reassembly block, link layer retransmission block, LHP agent and sinker.

The simulation topology is the standard single bottleneck scenario where competing flows have the same RTT delay. This eliminates unfairness among TCPs with different RTT. In the scenario shown in Fig.5, the link between the wireline router $R1$ and base station $B1$ is the bottleneck. The congestion occurs at router $R1$ and corruption occurs on the link between $B1$ and mobile hosts Mi.

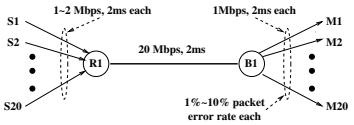

Fig. 5. Simulation topology

We use IP layer packets having a fixed size of 1500 bytes including TCP/IP header. The link layer has fragment packet size of 256 bytes and hence each IP packet has six link layer time slots. The link between $R1$ and $B1$ has fixed bandwidth 25Mbps and delay 2ms. The wireless link between $B1$ and $\{Mi\}$ has fixed bandwidth 1Mpbs and delay 2ms but variable link layer bit error rate (BER). The link between $\{Si\}$ and $R1$ has fixed delay of 2ms but a variable link bandwidth allowing for different congestion conditions to occur.

We compare the performance of our proposed protocols with various versions of TCP proposed in literature, these include TCP-RENO, SACK, HACK. The modified link layer transmission scheme fragments the IP layer packets according to the link layer packet size, and uses stop-and-wait mechanism to transmit the packet header until the maximal allowable delay is reached. This maximal delay is a parameter that can be further tuned to suit the application. The link layer receiver is designed to check the integrity of the received fragment and send back an ACK to the sender. Using the LHP protocol, the fragments are assembled and information about the packet integrity is passed on to the upper layers. We call **new-LHP** the complete protocol which utilizes both the ELN mechanism of LHP (from Section III) and the control mechanisms introduced in Section IV. We call **simple-LHP** the protocol that has the same ELN and congestion control mechanism except the ability shown in example 3 in Section IV. Then when a retransmitted packet (due to lhp packet or congestion) gets lost again, **simple-LHP** will timeout. On the contrary, **new-LHP** will try to recover it.

The LHP sink block is the same for both simple-LHP and the new-LHP. Upon receiving the packet from link layer, the receiver uses the LHP option field to provide the ELN mechanism and informs the sender about the sequence number and packet length of the corrupted packet. The SACK option is used in the manner specified in [6]. The HACK sender doesn't has the similar ability of SACK to find out the gap of receiving packets.

We are going to present two main sets of results. In Section B we are going to present the performance of different TCP versions in the presence of only link losses. In Section C we compare the performance of our proposal with the various TCP versions in the presence of both congestion and link losses.

B. Behavior with link losses

In this set of simulations we examine just the effect of link losses on each of the protocols without congestion. The experiment runs for a period of 200 seconds and we average over 10 runs for each link layer packet frame error rate (FER) of the IP packets. We used a uniformly distributed packet error model for these simulations. And we examine for each protocol two metrics for comparison: throughput and goodput. By definition, throughput is the number packets sent out by the sender and the goodput is the number of unduplicated packets received by the receiver.

In Fig.6 we compare the throughput of various versions of TCP as the FER varies from 0-3%. As expected the methods such as

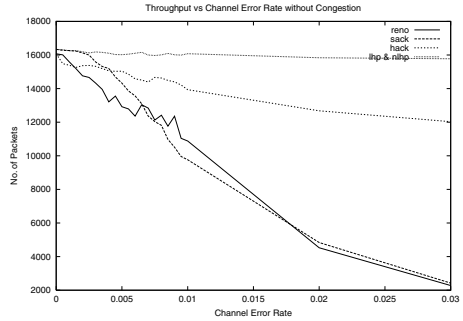

Fig. 6. Throughput vs Packet Frame Error Rate (FER) of different TCP

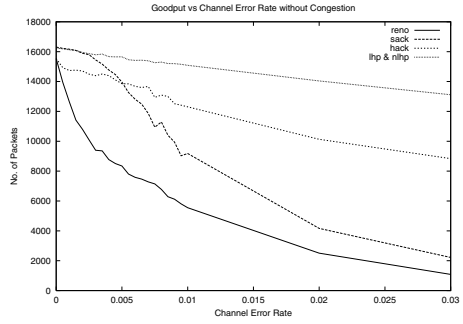

Fig. 7. Goodput vs Packet FrameError Rate (PER) of different TCP

SACK and TCP-Reno that do not have an ELN mechanism perform quite poorly even for small FER. The ELN mechanism in HACK allows them to tolerate some link losses, but the performance still deteriorates significantly when FER increases. In contrast for the entire range of FERs shown, the proposed LHP schemes show little degradation in throughput and clearly outperform other mechanisms in this regime.

When FER is 1%, both LHP schemes perform 13.3% better than HACK,61.2% better than SACK, and 45.9% better than RENO. Note that both simple-LHP and new-LHP perform similarly in the presence of just link losses. The difference in their performances becomes apparent in Section C where both congestion and link losses are encountered.

The goodput of the TCP mechanisms is shown in Fig.7. There are two main points to note in this set of experiments. First, the loss in goodput of the LHP schemes is small, and is mainly due to retransmission, not because of congestion control mechanisms. Therefore, these schemes again outperform the other TCP mechanisms. Secondly SACK's goodput is 65.3% better than RENO in Fig.7 despite that RENO's throughput is 10.3% better than SACK in Fig.6 when FER is 1%. This shows RENO has send out more duplicate packets.

C. Throughput behavior with congestion and link losses

In this section we present our comparison of different TCP when both congestion and corruption occur. The simulation are composed of 25 senders, 5 for each version of TCP, i.e, reno, sack, hack, simple-lhp and new-lhp. The bandwidth of the link between Si and $R1$ is varying from 1.01 Mbps to 1.600 Mbps which varies the congestion ratio from 1.0 to 1.6. The wireless FER is varied from 0% to 2% as well. The simulation runs for 200 seconds and the result is averaged over several flows of each TCP version.

Fig.8 shows the influence of different link loss rates on the TCP goodput when congestion ratio is fixed to 1.1 and Fig.9 shows the influence of different congestion ratio on the TCP goodput when FER is fixed to 0.5%. In both Figures lhp versions performs much better than other TCP versions. And in Fig.8 new-lhp performs 1 to 2% better than simple-lhp when FER changes from 0.5% to 3%. In Fig.9 new-lhp performs 0.9% to 4.3% better than simple-lhp when congestion ratio changes from 1 to 1.6. Considering the only difference between two lhp versions is the mechanism to detect the congestion of retransmission packets, this improvement is not trivial. The reason that goodputs in Fig.9 change very slowly when congestion ratio changes dramastically is that TCP traffic is elastic so 1.6 congestion ratio doesn't mean 60% packets get dropped.

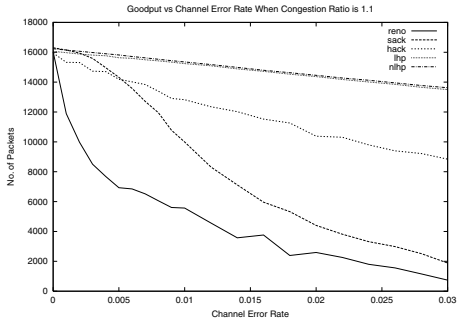

Fig. 8. Influence of link loss rate on the TCP goodput when congestion ratio is 1.1

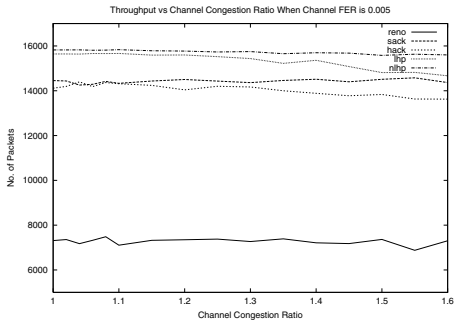

Fig. 9. Influence of congestion ratio on the TCP goodput when channel FER is 0.5%

VI. Discussion

In this paper we exposed the limitations of current transport protocols in hybrid wireless-wireline networks. We have shown that it is important to tackle both the the problem of localizing the source of packet loss (congestion/link failure) and also the existence of a large number of out-of-order packets, which arise due to packet retransmission. We have proposed a simple, scalable transport protocol that provides a solution to both these problems. Through simulations, we have demonstrated that this protocol can give significant gains in data throughput over existing protocols.

References

[1] H. Balakrishnan, V. Padmanabhan, S. Seshan, and R. H. Katz, "A comparison of mechanisms for improving tcp performance over wireless links," *IEEE Transactions on Networking*, December 1997.

[2] A. Bakre and B. R. Badrinath, "I-TCP: Indirect TCP for mobile hosts," in *Proc. 15th Int. Conf. Distributed Computing System (ICDCS)*, May 1995.

[3] S. Kunniyur and R. Srikant, "End-to-end congestion control: utility functions, random losses and ECN marks," in *Proc.INFOCOM'2000*, 2000.

[4] G. Holland and N. Vaidya, "Analysis of TCP performance over mobile ad hoc networks," in *Proc. of Mobicom*, 1999.

[5] A. DeSimone, M. C. Chuah, and O. C. Yue, "Throughput performance of transport-layer protocols over wireless LANs," in *Proc. of Globecom*, December 1993.

[6] M. Mathis, J. Mahdavi, S. Floyd and A. Romanow"TCP selective acknowledgement options," in *RFC 2481*, October 1996.

[7] K. Brown and S. Singh, "M-TCP: TCP for mobile cellular networks," in *ACM Computer Comm. Rev. (CCR)*, vol. 27, no. 5, 1997.

[8] M. Mathis and J. Mahdavi, "Forward acknowledgement: Refining TCP congestion control," in *Proc. of ACM SIGCOMM*, 1996.

[9] R. K. Balan, etc., "TCP HACK: TCP header checksum option to improve performance over lossy links," in *Proc. IEEE INFOCOM'2001*, 2001.

[10] K. Fall and S. Floyd, "Simulation-based comparisons of Tahoe, Reno, and SACK TCP", in *Computer Communication Review*, October 1994.

[11] X. Gao, "QoS Management in Packet Cellular Network", Ph.D thesis, University of Illinois at Urbana-Champaign, 2002.