Using Congestion Coherence to Enhance TCP over Wireless Links*

Chunlei Liu Raj Jain
Department of Computer and Information Science
Raj Jain is now at Washington University in Saint Louis, jain@cse.wustl.edu http://www.cse.wustl.edu/~jain/

June 4, 2001

Abstract

It is well known that TCP has low performance over wireless links. With the rapid development of wireless and mobile communications, there is a pressing need to enhance TCP performance over wireless links. In the past few years, a number of enhancements have been proposed. In this paper, we review these proposals and analyze the degradation of TCP performance due to errors in the wireless link. We propose an enhancement that meets several desirable requirements and relieves TCP from all degradations caused by wireless errors. Simulation results and analysis show this enhancement works much better than other enhancement proposals for all ranges of wireless error rates.

1 Introduction

Transmission Control Protocol (TCP) is the most widely used transport protocol in the network world. By providing reliable data transport and congestion control, TCP serves as the foundation of today's Internet. Historically, TCP was designed mainly for wired networks where transmission errors are rare and the majority of packet losses are caused by congestion. One of our contributions to TCP is pointing out that packet loss and the resulting timeout at the source are indications of congestion and the sources should reduce their traffic on timeout [1].

With the rapid development of wireless networks in recent years, TCP has been deployed on wireless networks. It has been discovered that TCP has poor performance over wireless links. This is because the assumption behind TCP congestion control algorithm — that the majority of packet losses are caused by congestion — is no longer true on the wireless links. When a wireless loss is treated as a congestion loss[1], the effective TCP transmission rate is cut by half. If wireless errors happen frequently, the effective transmission rate of the wireless link becomes almost zero.

A scenario of such transmission rate drop is illustrated in Figure 1. Before transmission error happens, the path between the sender and the receiver can support a window size of six packets. The sender transmits at this rate and a packet is lost on the wireless link due to a transmission error. When the destination sees the out-of-order packets, it follows TCP congestion control algorithm to send back duplicate ACKs. Upon the receipt of three duplicate ACKs, the sender assumes that congestion has happened in the network, retransmits the lost packet and reduces the window by half. As a result, the wireless link that can send six packets in a window is now sending only three, even though the network is not congested.

In the past few years, a number of enhancements have been proposed to improve TCP performance. These enhancements suggest modification at the sender, the base station or the receiver. Some of them try to hide the lossy characteristic of wireless link from TCP by performing buffering and retransmission at the base station. Others use extra traffic to find out the network congestion status. These methods are reviewed and evaluated in the next section of this paper. It is our conclusion that these methods either make unrealistic assumptions, do not meet certain implementation requirements, or apply only to a special network configuration.

*This research was supported in part by National Science Foundation grants 9980637 and 9809018.

[1] In this paper, we call packet losses caused by wireless transmission error "wireless losses", and packet losses caused by network congestion "congestion loss".

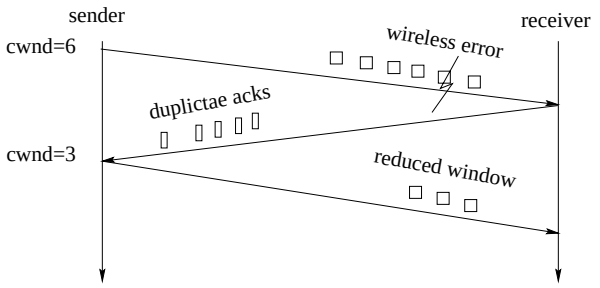


Figure 1: Window reduction due to wireless losses

In this paper, we identify the three ways that wireless errors degrade TCP performance, and propose a new enhancement approach. This new approach makes use of the sequential coherence of congestion marks to distinguish wireless losses from congestion losses. It requires only minor modifications in the mobile host, and applies to all network configurations. Analysis and simulation results show that this approach avoids the majority of retransmissions, unnecessary window reduction and timeouts, and thus significantly improves TCP performance. For all ranges of packet error rates, this enhancement works better than all other proposals.

2 Related Work

Since the problem of low TCP performance on wireless links was discovered, a number of enhancements have been proposed. In this section, we review the proposals reported in the literature. We briefly describe how these proposals work, and then discuss its advantages and drawbacks. At the end of this section, these proposals are compared with a set of implementation requirements we think desirable.

2.1 Review of Existing Proposals

I-TCP [2] is an early protocol that splits the entire path into two separate connections, a wired connection between the sender and the base station, and a wireless connection between the base station and the mobile host. TCP/IP runs independently on the two connections, so errors in the wireless connection do not affect the wired connection and congestion in the wired connection does not affect the wireless connection. The proposal is simple in concept and easy to implement. Its major drawback is that acknowledgments received by the sender do not mean that the packets have been received by the intended destination. When the mobile host moves to another cell or when the base station crashes, some acknowledged packets may never be received. So this proposal violates TCP's end-to-end semantics. Its second drawback is heavy buffering at the base station. Because the two connections run TCP at different rates, a large number of packets may pile up at the base station, or the wireless link may be idle.

Multiple Acknowledgments [3] uses two types of acknowledgments to distinguish losses in wired network and losses on wireless link. A partial acknowledgment ACKp indicates the base station has received the packet. A complete acknowledgment ACKc means the packet is received at its final destination. By using partial acknowledgments, unnecessary end-to-end retransmissions on the wired networks are avoided. Packets partially acknowledged but not completely acknowledged within a certain time will be considered as wireless losses and will be retransmitted. The main problem with this proposal is that it needs modifications at the fixed host and the base station. Virtually, this proposal requires all nodes that talk with the mobile host to change their TCP code. In practice, this is impossible. Extra traffic caused by partial acknowledgments is also a drawback of this proposal.

Control Connection [4] creates a control connection to measure the congestion status on the wired network. The control connection has the same path in the wired network as the first connection but terminates at the base station. Packets of the two connections are assumed to be routed in the same way and hence experience the same congestion in the wired network. If both connections have lost packets at the same time, then the lost packets are regarded as

congestion loss. If the control connection has no packet loss, then the lost packet in the first connection is regarded as a wireless loss. This method is based on unrealistic assumptions. First, the assumption that the two connections have the same route is questionable. Second, when congestion happens, packet losses are not dropped consecutively, the probability that both connections experience packet drops is small. Therefore, the correlation of packet losses between the two connections may be low. Third, in order to monitor the congestion status, the control connection need to run at a rate comparable to the first connection, and thus adds a significant overhead traffic to the network.

The **snooping protocol** [5] introduces a *snooping agent* at the base station. The agent observes and caches TCP packets going out to the mobile host, as well as acknowledgments coming back. By comparing the cached packets and acknowledgments, the agent is able to determine what packets are lost on the wireless link and schedule a local link layer retransmission. On the same time, duplicate acknowledgments corresponding to wireless losses are suppressed to avoid triggering an end-to-end retransmission at the source. Unlike the other proposals, the snooping protocol can exactly find out the cause of packet losses and take actions to prevent the TCP sender from making unnecessary window reductions. However, the snooping protocol requires the base station to cache TCP segments and keep per-connection state, so a heavy processing burden is added to the base station. Second, the base station needs to check TCP header to find the sequence number and the acknowledgement. When the packets are protected with IP security, this proposal does not work. Third, snooping protocol only works for traffic from the fixed host towards the mobile host. When the mobile host is a sender, the base station need to use the "explicit loss notification" discussed next.

Explicit Loss Notification (ELN) [6] uses a bit in the TCP header to communicate the cause of packet loss to the TCP sender. At the base station, a snooping agent monitors all TCP segments that arrive over the wireless link as well as acknowledgments from the wired network. ELN does not cache TCP segments, but keeps track of holes in the TCP sequence space. These holes correspond to segments that have been lost over the wireless link. When an ACK corresponding to a hole arrive from the wired network, the ELN bit in the ACK is set before being forwarded to the data sender. When the sender receives an ACK with ELN bit set, it retransmits the next segment but does not take any congestion control actions. ELN can detect the exact cause of packet loss and take correct action to prevent unnecessary window reductions, but suffers heavy processing. It cannot work with encrypted packets, and is only valid for traffic from the mobile host to the base station. In addition, ELN introduces unnecessarily long delay for retransmitted packets. When base station detects a wireless error, it does not ask for a retransmission immediately but waits for the duplicate ACKs coming back to trigger the retransmission.

Delayed Duplicate Acknowledgments (DDA) [7] algorithm attempts to imitate the behavior of the snooping protocol, but makes modifications at the receiver rather than the base station. It assumes a link level retransmission scheme is implemented. When out-of-order packets are received, the receiver sends duplicate ACKs for the first two out-of-order packets. If it gets more of them, the receiver defers ACKs for these packets for a duration d. If during this period, the next in-sequence packet arrives, the receiver discards the deferred duplicate ACKs and sends a new ACK. If the in-sequence packet does not arrive during this period, the receiver releases the deferred duplicate ACKs to trigger a retransmission. DDA is a simple proposal and requires modification only at the mobile host. However, it can not distinguish wireless and congestion losses, and increases the delay of retransmitted congestion losses.

Table 1 summarizes the proposals we reviewed above.

2.2 Enhancement Requirements

Although each of the above reviewed proposals solves the low performance problem in some ways, they are based on various assumptions and apply only to certain network configurations. If the assumptions are not realistic, the proposal may not be feasible in practice. If the proposal only apply to certain network configurations, then it is not a complete solution. In this section, we summarize a set of requirements that we think desirable in practice. These requirements are listed in the order of importance.

First, **the enhancement must maintain end-to-end semantics**. TCP is a *reliable* transport protocol. All packets acknowledged must have been received correctly at their destination. If any enhancement does not meet this requirement, it is *not* a reliable transport protocol.

Table 1: Comparison of proposals to enhance TCP over wireless links

	I-TCP	**Multiple Acks**	**Control Connection**	**Snooping**	**ELN**	**Delayed Dupacks**
Modify MH	no	no	yes	no	no	yes
Modify BS	yes	yes	no	yes	yes	no
Modify FH	no	yes	yes	no	no	no
Complexity	simple	moderate	moderate	high	high	simple
Add traffic	no	yes	yes	no	no	no
BS buffering	heavy	heavy	no	heavy	no	light
TCP states at BS	light	light	no	heavy	heavy	no
Two-way solution	yes	no	no	no	no	no
End-to-end semantics	no	yes	yes	yes	yes	yes
Interpret losses	no	yes	probably	yes	yes	no
Interpret dupacks	no	yes	probably	yes	yes	yes
Delay variations	small	small	large	small	large	small

Notes:

1. BS buffering: high if transport layer retransmission; light if link layer retransmission; no if retransmission is not needed.
2. TCP states at BS: high if BS needs to maintain a hole list; light if only window variables are needed.
3. Delay and delay variation: large if the corrupted packets are retransmitted end-to-end; small if retransmitted locally.

Second, **all modifications to the existing algorithm must be local**. Adding a wireless link to the network should not require the entire network to change. In the case of a mobile host, only the base station and the mobile host are under the control of the wireless link manager. Modifications beyond this scope are normally impractical. For example, when a wireless service provider offers a web browsing feature, it is his responsibility to improve TCP performance over the wireless link, but requiring all web sites that his customers visit to change is impossible.

Third, **the enhancement must apply to encrypted traffic**. As data security gets more important, a large portion of traffic is likely to be encrypted. Encryption protocols like IPSEC encrypt the entire IP payload so that intermediate nodes cannot see what is being carried. Only the destination has access to the content of IP packets. Enhancement solutions that need to see TCP header at the base station or at an intermediate router do not meet this requirement.

Fourth, **the enhancement should apply to two-way traffic**. Many enhancement proposals are designed for mobile receivers, and work only for traffic from the wired network to the mobile host. However, in almost all cases the mobile hosts also send data packets. Wireless errors affect downward traffic as well as upward traffic. Any enhancement that works only for one-way traffic is not a complete solution. In order to fully utilize the wireless resources, an enhancement that work for two-way traffic is needed.

Fifth, **the enhancement should apply to intermediate wireless links**. Many enhancement proposals presume the wireless link is the last hop of the connection and work only for this configuration. While this is true in many cases, other types of networks like ad-hoc networks, mobile networks and satellite links that have the wireless link as an intermediate link also have the low performance problem. It is desirable that the enhancement can apply to intermediate wireless links.

In addition to the above five requirements, it is desired to control the traffic, buffering and processing overhead. **The enhancement should keep the overhead traffic (both in the wired network and on the wireless link), the buffer requirement and the computing to minimal.**

Table 2 summarizes whether the current proposals meet the requirements and their traffic, buffering and computing

overhead. A $\surd$ means the requirement is met or the overhead is low, a $\times$ means the requirement is not met or the overhead is high. Surprisingly, none of current solutions meets all the requirements.

Table 2: Which solutions meet the requirements?

	I-TCP	**Multiple Acks**	**Control Connection**	**Snooping**	**ELN**	**Delayed Dupacks**
Semantic Requirement	$\times$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$
Local Requirement	$\surd$	$\times$	$\times$	$\surd$	$\surd$	$\surd$
Encryption Requirement	$\times$	$\times$	$\surd$	$\times$	$\times$	$\surd$
Two-way Requirement	$\surd$	$\times$	$\times$	$\times$	$\times$	$\times$
Intermediate Link Requirement	$\surd$	$\times$	$\surd$	$\times$	$\times$	$\surd$
Traffic Overhead	$\surd$	$\times$	$\times$	$\surd$	$\surd$	$\surd$
Buffering Overhead	$\times$	$\surd$	$\surd$	$\times$	$\times$	$\surd$
Computing Overhead	$\surd$	$\surd$	$\surd$	$\times$	$\times$	$\surd$

Notes: Both snooping and ELN are one-way solution, but they can be combined to provide a two-way solution.

3 Proposed Approach

As discussed above, errors cause the poor TCP performance over wireless links. A closer look at TCP traces reveals that wireless errors can degrade TCP performance in three ways:

End-to-end retransmissions: When a packet is lost in the wireless link, the packet has to be retransmitted. If the retransmission is done end-to-end, the extra traffic can add to the congestion and reduce the number of packets that can be sent through the bottleneck link. Extra traffic also causes longer delay.

Unnecessary window reductions: When wireless losses are retransmitted end-to-end, regular TCP algorithm treats the packet loss as an indication of congestion and thus reduces the congestion window size. Unnecessary window reductions cause big performance degradation because the effective transmission rate is cut to half.

Timeouts: The existing TCP algorithms (Tahoe, Reno or new Reno) can recover one packet loss in a window using duplicate acknowledgments. Two or more packet losses (can be congestion or wireless) in the same window will result in a timeout. Timeouts cause severe performance degradation because it takes many round trip times (RTT) to bring the normal transmission rate to normal value after a timeout. If packet loss is used for congestion control as in the current TCP algorithm, then any wireless loss within the RTT that contains a congestion loss always results in a timeout.

In order to improve TCP performance, an enhancement solution should eliminate the above three degradations. The first degradation can be eliminated by local link layer retransmission. The second degradation can be eliminated if a mechanism to distinguish between wireless and congestion losses is found. In order to eliminate the third degradation, we need to stop using congestion loss as a mechanism to adjust the window.

We find all current proposals eliminate only the first two degradations. They all use packet losses to deliver congestion feedback. Therefore, the third degradation is inherent and unavoidable.

In this paper, we use Explicit Congestion Notification (ECN) for congestion control and propose a mechanism to distinguish wireless and congestion losses. ECN avoids congestion losses by makes use of early congestion warnings. Avoiding congestion losses has two benefits on the performance. It avoids the end-to-end retransmissions, as well as timeouts caused by multiple losses in a window. In this way, we can eliminate all the three degradations of TCP performance.

Our proposal consists of the following three parts:

- *Assume that all network routers are ECN capable.*
- *Implement local link layer retransmission on the wireless link to avoid end-to-end retransmission of the wireless losses,*
- *Use congestion coherence to determine cause of packet loss at the mobile.*

Details of the proposed approach are given below.

3.1 Explicit Congestion Notification

The idea of marking packet header at the congested router to deliver binary feedback on congestion status can be traced to our *DECbit* scheme [8] in 1988. Compared with timeout and duplicate ACKs, explicit feedback delivers a direct and faster congestion signal to the sender.

Explicit Congestion Notification, the binary feedback scheme in TCP/IP, was first introduced by Floyd and Ramakrishnan [9]. ECN uses two bits in the IP header and two bits in the TCP header for ECN capability negotiation and feedback delivery. When the queue length exceeds a threshold and the incoming packet is labeled ECN-capable, the router marks the packet as *congestion experienced*. At the destination, the Congestion Experience bit is copied to the *ECN-echo* bit in the TCP acknowledgment and thus delivers a congestion notification back to the sender. Upon receiving the ECN-echo, the sender reduces its congestion window to alleviate the congestion.

ECN has been proved to be effective in avoiding unnecessary packet drops and in improving TCP performance. In RFC 2309 [10], it was recommended to be widely deployed as a router mechanism on the Internet, and was standardized by IETF in RFC 2481 [11] in 1999.

The marking policy determines how a packet is marked. Floyd and Ramakrishnan [9] recommend using Random Early Detection (RED) [12] as the marking policy. RED detects incipient congestion and randomly marks packets at a computed probability when the average queue size exceeds a threshold. Figure 2 shows the marking policy used in our simulation, which is a little different from that in [9]. We use the *actual* queue size instead of the average queue size. When the queue size is below a minimum threshold th_{min}, the incoming packet will not be marked. When the queue size is between th_{min} and a maximum threshold th_{max}, the incoming packet will be marked with a probability proportional to the queue size. When the queue size is greater than th_{max}, all incoming packets are marked. When the queue size is equal to buffer size, all incoming packets are dropped.

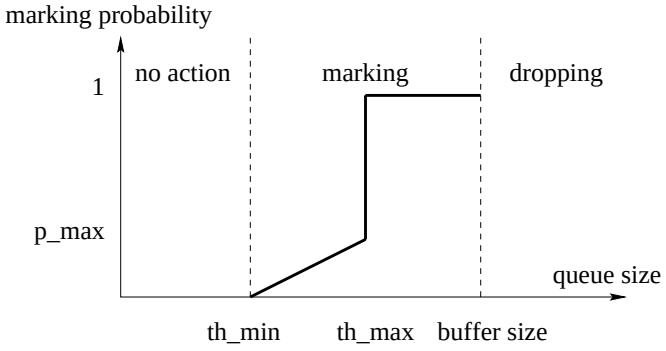


Figure 2: Marking and dropping policy used for congestion coherence

3.2 Local Link Layer Retransmission

When a packet is lost in the wireless media, it has to be retransmitted, either end-to-end or over the local wireless link. Compared to the end-to-end retransmission, local link layer retransmission not only avoids extra traffic in the wired

network but also reduces the delay. However, the delay caused by the local link layer retransmission can interfere with the end-to-end TCP retransmission clock. If the local link layer retransmission takes too long, the source may timeout and trigger an end-to-end retransmission before the acknowledgment of local retransmission is received. In designing the link layer retransmission scheme, it is important to transmit the failed packets with high priority to reduce the retransmission delay. One way of implementing this is to use the "insert from front" strategy. When a packet is detected to be lost, the link layer inserts the failed packet into the front of the transmission queue and transmits it when the media is available. In this way, the failed packet does not suffer queuing delay again.

An implication of local link layer retransmission is out-of-order packets. When a packet is lost and retransmitted, the TCP receiver will receive out-of-order packets and may respond duplicate ACKs. Both congestion losses and wireless losses cause out-of-order packets, and create holes in the packet sequence number space, but their consequences are difference. A hole caused by a wireless loss will be filled when the retransmission arrives, but a hole caused by a congestion loss will not be filled without a timeout or triggering an end-to-end retransmission. If the receiver knows the hole is a wireless loss, it should wait for the retransmission. If it knows the hole is a congestion loss, it should trigger the end-to-end retransmission right away.

The receiver needs a scheme to tell whether hole is a wireless loss or a congestion loss. Such a scheme is described in the next section.

3.3 Congestion Coherence

Table 3 illustrates two loss cases taken from a simulation trace. Packets 37 and 112 are lost, but the ECN marks of their neighboring packets are listed in the table. An "E" means the packet is marked "Congestion Experienced", a blank means it did not experience any congestion. By looking at these markings, is it possible to guess which packet is lost due to congestion and which is lost due to error?

seqno	mark
33	
34	E
35	E
36	E
37	lost
38	E
39	E
40	

seqno	mark
108	
109	
110	
111	
112	lost
113	
114	
115	

Table 3: Are these packets lost due to congestion or due to transmission error?

Packet 37 is a congestion loss and packet 112 is a transmission loss. This observation is based on the so called *congestion coherence* of ECN markings. It reflects the fact that congestion does not happen or disappear suddenly. Before congestion becomes so severe that a packet has to be dropped, some packets must have been marked. Similarly, after a packet is dropped, congestion does not disappear immediately. The queue size falls gradually and some packets are marked. As a result, congestion losses are normally preceded and followed by marked packets, see Figure 3.

In contrast, transmission errors normally happen independent of congestion. Neighboring packets of a wireless loss are usually not marked.

Therefore, being surrounded by marked packets is a tag of congestion losses. Congestion coherence can be used to distinguish congestion losses from wireless losses. We define the *coherence context* of packet n as packets $\{n-1, n+1, n+2\}$. The coherence context is said to be marked if any packet in the context is marked.

If a packet is found lost at the destination and its context is marked, duplicate acknowledgments should be sent right away to invoke fast retransmit and congestion control at the source. If the context is not marked, it is most likely

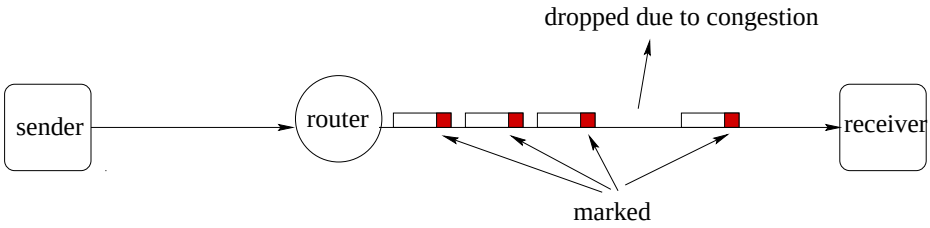


Figure 3: Congestion coherence

a wireless loss. Duplicate acknowledgments should be held to avoid invoking fast retransmit and congestion control at the source.

The same idea can be applied to the wireless sender case. When the wireless sender receives duplicate acknowledgments, it checks whether the coherence context contains an ECN-Echo. If yes, then the duplicate acknowledgments are most likely caused by a congestion loss, so the sender invokes the congestion control. Otherwise, the duplicate acknowledgments are most likely caused by a wireless loss. The sender ignores duplicate acknowledgments until the local retransmission succeeds.

3.4 Congestion Coherence TCP Algorithm

In our proposed approach, the modifications to the existing TCP algorithm are made in the wireless end. Based on the technique discussed above, this approach is named *Congestion Coherence*. Figures 4 and 5 show the modified receiving and sending algorithms.

- The TCP sink follows existing algorithm for sending new acknowledgments, first and second duplicate acknowledgments.
- When the third duplicate acknowledgment is to be sent, TCP sink checks whether the coherence context is marked. If yes, the acknowledgment is sent right away. Otherwise, it is deferred for w seconds, and a timer is started.
- If the expected packet arrives during the w seconds, a new acknowledgment is generated and the timer is cleared.
- If the timer expires, all deferred duplicate acknowledgments are released.

Figure 4: Congestion Coherence receiving algorithm

It should be noticed that the modifications to the receiving and sending algorithms are made on the same end. The implementation of Congestion Coherence at the wireless end hides the lossy characteristic of the wireless link from the other end. The wired end and intermediate routers, including the base station, need no modification. If the wireless link is in the middle, such as a satellite link or in an ad-hoc wireless network, the modifications can be made on any end, or both ends.

Revisiting Table 2, we can note that our approach meets all the requirements. The traffic overhead is zero because feedback is delivered using the IP and TCP headers and no extra packets are sent. The buffering is minimal, only those packets sent over the wireless link but not acknowledged yet are buffered. The computing complexity is very small, only a few lines of modification need to be added to the existing TCP code.

- The TCP sender follows existing algorithm for sending packets and updating congestion window upon receiving new acknowledgments, first and second duplicate acknowledgments.
- When the third duplicate acknowledgment arrives, the sender checks whether any acknowledgment in the coherence context is an ECN-Echo. If yes, the packet corresponding to the duplicate acknowledgments is sent right away and congestion window is reduced to half if a reduction has not been done in the previous RTT. Otherwise, the sender ignores the duplicate acknowledgement and a timer of w seconds is started.
- If a new acknowledgment arrives during the w seconds, the timer is cleared and new packets are sent as if the duplicate acknowledgments did not happen.
- If the timer expires, the packet corresponding to the duplicate acknowledgments is sent and congestion window is reduced to half if a reduction has not been done in the previous RTT.

Figure 5: Congestion Coherence sending algorithm

4 Mistake Scenarios

Congestion coherence provides an educated guess of the cause of packet losses. However, this method is based on probability and can make mistake in some cases. In this section, we discuss these mistake cases and analyze their performance impact.

The first mistake case happens when a wireless error occurs during congestion time. Since neighboring packets are marked, the wireless loss may be treated as a congestion loss. Therefore, congestion control is triggered at the source and an end-to-end retransmission is started. In this case, the end-to-end retransmission is unnecessary because local link layer retransmission will deliver the packet again. However, triggering the congestion control is needed because of the existence of congestion. As congestion control is more important than retransmitting a single packet, sending the third duplicate acknowledgment right away is the correct action.

The second mistake case happens when a burst of background traffic causes a congestion loss without marking the coherence context. In this case, our method regards the packet loss as a wireless loss and will wait w seconds. At the end of this w second, the retransmission of this packet does not come, so deferred duplicate ACKs are released to trigger an end-to-end retransmission. The cost of this mistake is a delay in the end-to-end retransmission and window reduction.

These two mistakes affect the performance, but results from our simulations show that the mistake rate is very small and the impact is minimal.

5 Simulation and Result Analysis

In order to compare our proposed method with other proposals, we performed a set of simulations with the ns simulator [13]. In this section, we present our network model, simulation scenarios, data collection method and results from analyzing individual simulation traces and aggregate measurements.

5.1 Simulation Model and Scenario Design

The simulations are performed on the simplified network model shown in Figure 6, where s_1, s_2 are the sources and d_1, d_2 are the destinations. The link between intermediate routers r_1 and r_2 is the bottleneck link. The link between r_2 and d_1 is a wireless link. The numbers beside each link represent its rate and delay.

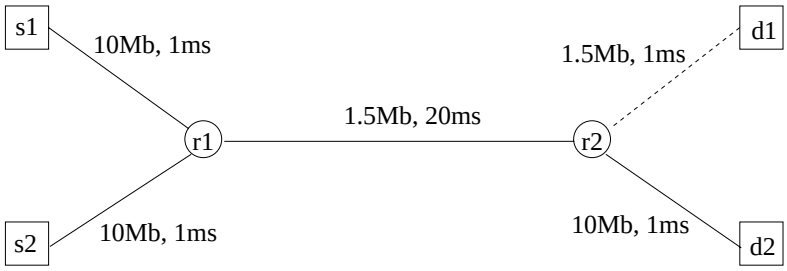


Figure 6: Simulation model

The experiment traffic is an FTP session from s_1 to d_1 using TCP Reno as the transport protocol. The background traffic is a UDP flow from s_2 to d_2 generated by an exponential on-off model. The mean burst period and the mean silence period are 100 ms, and the burst data rate is 500 kbps. Both TCP and UDP packet sizes are 1000 bytes, and TCP acknowledgments are 40 bytes long.

Link layer retransmission is implemented on the wireless link. Packets sent but not acknowledged at the link level in 40 ms are resent. Retransmitted packets are also subject to wireless errors at the same rate. The waiting time at the receiver is set at 81 ms so that packets delivered within two retransmissions are accepted. The packet error rate of the wireless link is varied to test the performance of various proposals under different loss scenarios.

To reflect the steady state measurement, the simulation time should be long enough to minimize the effect of the initial transient state. Longer simulation normally generates smoother aggregate results. Techniques to determine the simulation end time can be found in Chapter 25 of [14]. In our experiment, we tried various simulation time and found the results of 500 seconds show the essential features without noticeable difference from longer simulations, so all aggregate measurements are collected from 500-second simulations.

The proposals we compared include base TCP, DDA, Snoop and Congestion Coherence. When the mobile host is a sender, Snoop is replaced by ELN. In order to show ECN without congestion coherence does not work, we also compared plain ECN. We did not compare I-TCP, Multiple Acknowledgments and Control Connection because they violate the end-to-end and the local requirements and are considered unacceptable.

We experimented with various network configurations, including wireless link as the last hop (mobile receiver), as the first hop (mobile sender) and as intermediate links (e.g. ad-hoc or satellite link). Congestion Coherence works for all three configurations and has similar performance. Because most other proposals only work for the mobile receiver configuration, the comparison below is conducted only for this configuration.

5.2 Results and Analysis

Our first group of results, shown in Figure 7, is the TCP congestion window and queue length of each proposal. They are collected from 40-second simulation traces. The packet error rate in the simulation is 0.1. A calculation shows that the average window size of the wireless connection can be roughly 10 packets, but as shown in the figure, the window size of base TCP and ECN is reduced frequently. Their corresponding queue size graph shows the queue at the bottleneck link is almost always empty. Therefore, their link efficiency is very low. Snoop and DDA solve the problem of unnecessary window reductions caused by wireless errors. The window size is significantly increased and the bottleneck link is better utilized. Nevertheless, the spikes in the bottom of Snoop and DDA cwnd figure indicate these two methods suffer severe degradation from timeouts. Congestion Coherence is a thorough solution. Unnecessary window reductions and timeouts are avoided. The queue size figure shows it has high link efficiency.

The major metric to evaluate the enhancement proposals is *goodput*, which is defined as the number of packets successfully received and acknowledged by the mobile host, excluding the retransmitted packets. The goodput of the five proposals under different packet error rate is drawn in Figure 8. Base TCP performs reasonably well when packet error rate is very small, but as packet error rate increases, its performance degrades quickly. The performance curve

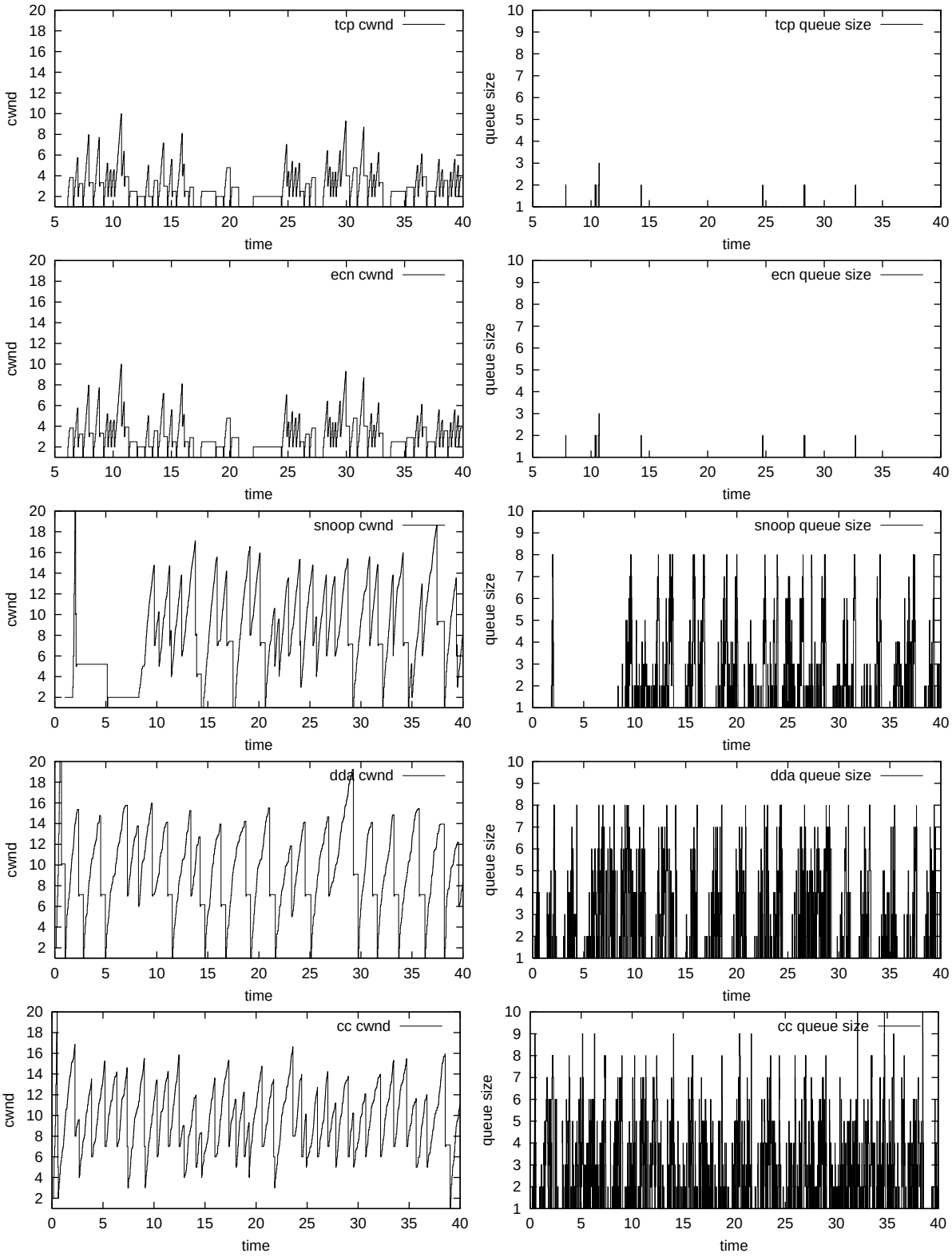


Figure 7: Congestion window and queue length for base TCP, ECN, Snoop, DDA and Congestion Coherence

confirms that TCP needs enhancement on wireless links. Plain ECN performs better than base TCP when error rate is very small, but its performance degrades quickly as the error rate increases. DDA does not degrade much with the error rate, but its performance under small error rate is low. Congestion Coherence proves to be better than all other methods for all ranges of error rates.

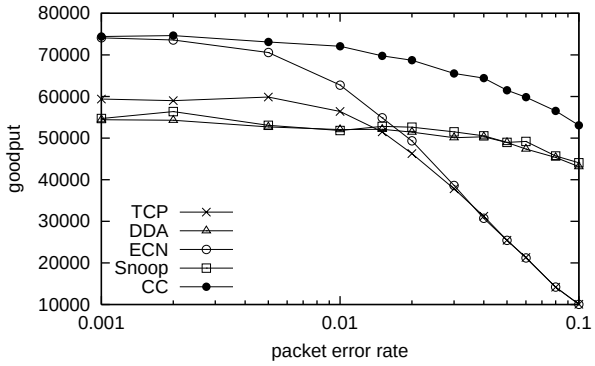
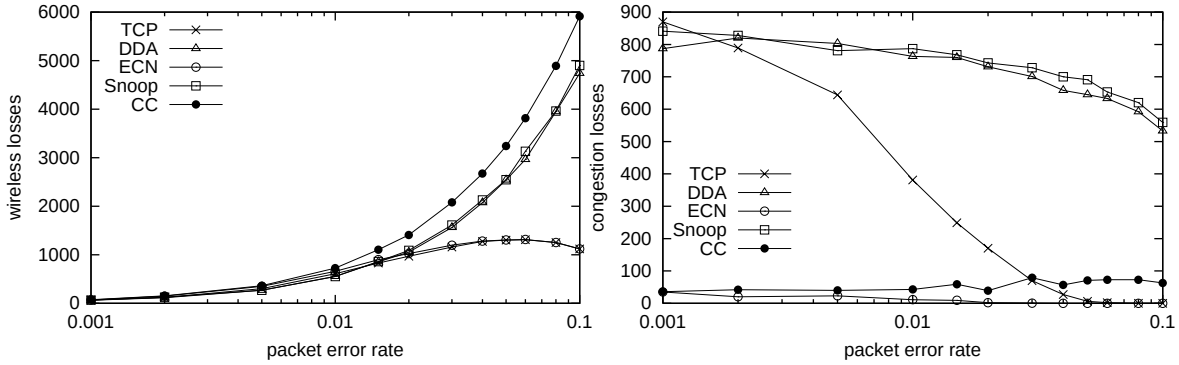


Figure 8: Goodput for the five methods

In addition to the goodput, we also analyzed the simulation trace and collected other data that helped us understand why one enhancement works better than another.



(a) Wireless losses

(b) Congestion losses

Figure 9: Wireless and congestion losses

Figure 9 (a) and (b) show the number of wireless and congestion losses. The number of wireless losses equals the total number of packets transmitted on the wireless link times the packet error rate. The number of congestion losses of base TCP, Snoop and DDA is significantly more than other methods because they use packet losses as a congestion control mechanism. As packet error rate increases, wireless losses reduce the congestion window so frequently that the window seldom grows to the level that a packet needs to be dropped. This explains the smaller number of congestion losses of base TCP in the right half of Figure 9(b). In contrast, methods using ECN do not suffer from congestion losses on a regular basis. Congestion losses happen only when bursts of background traffic generate so many packets that the buffer of bottleneck link cannot absorb. As analyzed in the beginning of Section 3, having fewer congestion losses helps to reduce end-to-end retransmissions and the chance of timeout.

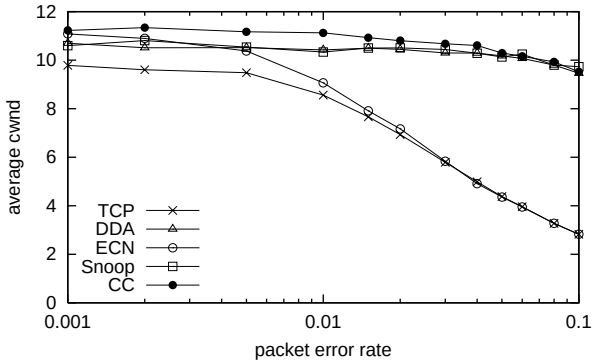


Figure 10: Average congestion window size

Figure 10 shows the average congestion window size. As packet error rate increases, wireless losses cause unnecessary window reductions in TCP and plain ECN, but the window size of Snoop, DDA and Congestion Coherence is not affected much by wireless errors. The slight drop in the right upper corner is caused by transmission errors in the retransmit packets.[2] This figure confirms that Snoop, DDA and Congestion Coherence solve the problem of unnecessary window reductions.

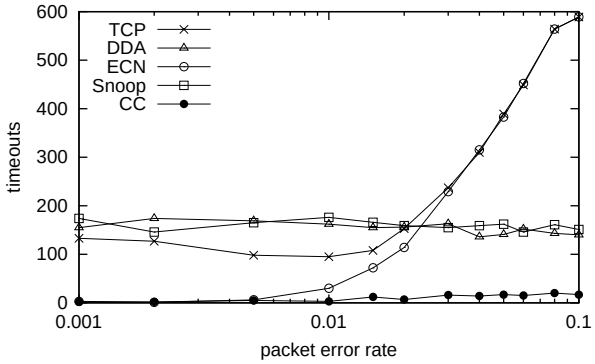


Figure 11: Number of timeouts

Figure 11 shows the number of timeouts that occurred during the simulation period. When packet error rate is small, TCP, Snoop and DDA have the largest number of timeouts because they use packet losses for congestion control. Their buffer occupancy at the bottleneck link can grow so high that bursts in background traffic can cause continual losses. Since two or more losses in a window causes a timeout. This translates to large number of timeouts. ECN and Congestion Coherence have very few timeouts because most of their congestion losses are avoided; background traffic causes occasional losses, but seldom become multiple losses in one window. As packet error rate increases, the number of timeouts in TCP and plain ECN increases dramatically because larger number of wireless losses increases the chance of multiple losses in one window. When the error rate is below 0.014, TCP has more timeouts than plain ECN. As the congestion window of TCP is reduced frequently by wireless losses (Figure 10) and congestion losses become fewer (Figure 9), TCP behaves almost identical to to ECN. The timeouts of Snoop and DDA are mainly caused by congestion losses, they remains constant for all packet error rates. Congestion Coherence has the smallest of all proposals. This figure is the evidence showing that only our proposal avoids the degradation caused by timeouts.

[2] In another simulation, when the retransmission method is replaced by perfect retransmission, i.e., no transmission error for retransmitted packet, wireless losses do not affect congestion window at all.

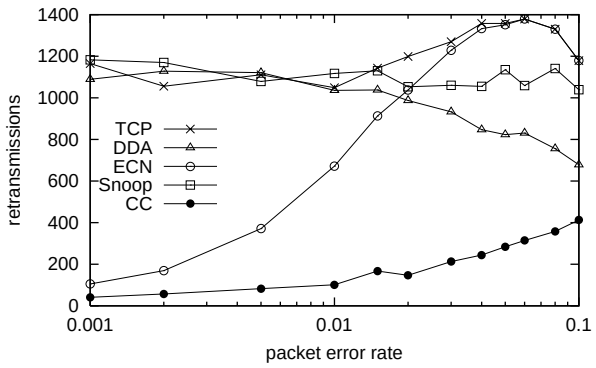


Figure 12: Number of end-to-end retransmissions

Figure 12 shows the number of end-to-end retransmissions. This number depends on the number of congestion losses, wireless losses and timeouts, as well as the enhancement method used. In fact, congestion losses in all methods are retransmitted. Wireless losses in TCP and plain ECN are retransmitted. When timeout happens, one full window of packets are retransmitted. Snoop and DDA avoid the majority of end-to-end retransmissions of wireless losses, but they still have a large number of retransmissions because of congestion losses and timeouts. Plain ECN reduces congestion losses, but cannot recover from wireless losses. All its wireless losses are retransmitted. Congestion Coherence avoids the majority of congestion losses, and waits for the local retransmission for wireless losses, so it has the smallest number of retransmissions.

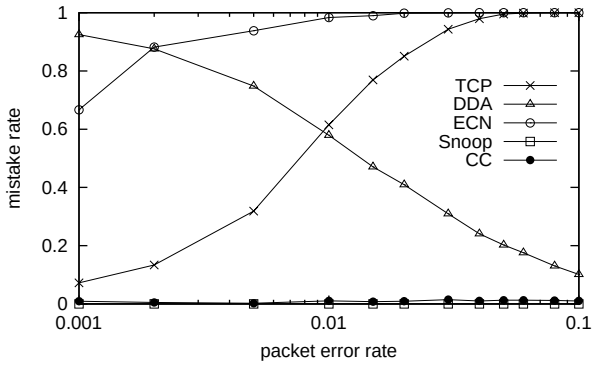


Figure 13: Mistake rate

Finally, the mistake rate in determining the cause of packet losses is shown in Figure 13. Both base TCP and plain ECN assume all losses are caused by congestion, so their mistake rate is the percentage of wireless losses in all losses. Plain ECN makes almost the same number of mistakes as base TCP, but it has much higher mistake rate because of its small number of congestion losses. DDA assumes all packet losses are due to wireless errors, so its mistake rate decreases when packet error rate increases. Snoop knows the exact cause of all packet losses by monitoring packets arriving at the base station, so its mistake rate is zero. Congestion Coherence takes advantage of congestion coherence and makes the right guess in most cases. As analyzed in Section 4, it makes mistake when very bursty traffic causes sudden packet losses without having neighboring packets marked. In our simulations, Congestion Coherence's mistake rate ranges from 0.06% to 1.2%. This rate is very small compared with other methods (except Snoop), and has minimal impact on the performance.

In summary, the simulation results reveal that Congestion Coherence avoids the majority of congestion losses and

is able to distinguish wireless loss from congestion losses. It is the only enhancement that avoids the three degradations of TCP performance over wireless links — end-to-end retransmissions, unnecessary window reductions and timeouts. Therefore, the performance of TCP is significantly improved for all ranges of packet error rates.

6 Discussions

This section is devoted to discussing a number of implementation details, alternatives and possible extensions for future study.

Coherence Context The coherence context used in our simulation is three packets, one before the lost packet and two after. We tried different size of the coherence context. It turned out that smaller coherence context tends to mistake congestion losses as wireless losses and causes more timeouts while larger coherence context tends to mistake wireless losses as congestion losses and causes more unnecessary end-to-end retransmissions. The other coherence context of size three, i.e., two before the lost packet and one after, gives similar result.

Location of Wireless Link Most enhancement proposals in the literature assume the wireless link is the last hop, congestion losses happen only between the fixed host and the base station, and wireless errors happen only between the base station and the mobile host. When the wireless link is a hop that connects two wired networks, like the satellite links, or when there are multiple wireless links as in an ad-hoc network, this assumption is no longer true. These solutions do not work in these cases. Our solution does not assume the location or the number of wireless links. As long as ECN is used in intermediate routers and the wireless links implement local retransmission, our solution will work.

Mark-Front Strategy It should be noticed that the mark on packets carries the congestion information of the route to the destination. The earlier the information is delivered to the sender, the more effective the sender's response can be. In our recent paper [15], we proposed to use the packet at the front of the queue, instead of the packet at the end of the queue, to carry the congestion information. This is called the mark-front strategy and has been shown to require smaller buffers, to generate higher throughput and to provide better fairness. In this paper, we assume marking and dropping are always performed on the packet in the front of the queue. Marking is done when a packet is leaving the queue and dropping is done when a packet enters a full queue.

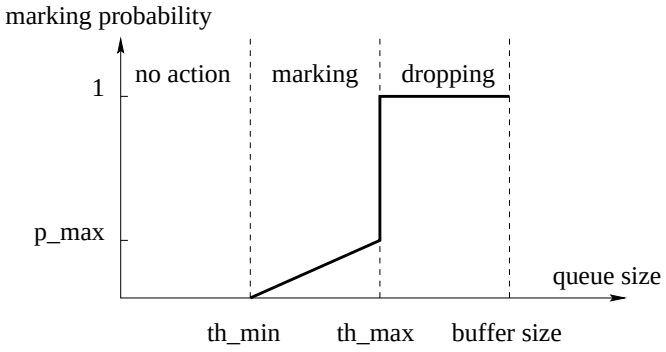


Figure 14: Early RED marking and dropping policy

Marking Policy The marking policy has a significant impact on congestion coherence. It is important to ensure that packets are not dropped without neighboring packets being marked. The randomness in deciding whether to mark a packet helps to desynchronize the TCP congestion windows among flows, but may jeopardize this insurance. Early versions of the RED algorithm, as depicted in Figure 14, yield low congestion coherence. The *gentle* option described in [16] and implemented in *ns2.17* improves the coherence. A better marking policy should keep a random

marking zone to help desynchronize TCP congestion windows among flows, and a deterministic marking zone to ensure congestion coherence, as shown in Figure 2.

Queue Length It should be noticed that it is the real queue length that is related to packet losses. Average queue length, because of its delay in reflecting the real congestion status, normally yields lower coherence and low goodput. Our simulations confirm this result. Discussing options of setting the RED parameters is beyond the scope of this paper, we will report them elsewhere.

7 Conclusion

In this paper, several proposed TCP enhancements over wireless links were reviewed. It was shown that these enhancement proposals do not meet the implementation and performance requirements. We analyzed the three degradations of TCP performance over wireless links and proposed a new approach called *Congestion Coherence*. This method makes use of the congestion coherence between consecutive packets to determine the cause of packet losses. By reducing the congestion losses and retransmitting the corrupted packets, Congestion Coherence significantly reduces end-to-end retransmissions, unnecessary window reductions and timeouts caused by wireless errors, and therefore improves the performance of TCP over wireless links.

References

[1] R. Jain, A timeout-based congestion control scheme for window flow-controlled networks, *IEEE Journal on Selected Areas in Communications*, Vol. SAC-4, No. 7, pp. 1162-1167, October 1986.

[2] A. Bakre, B. R. Badrinath, I-TCP: indirect TCP for mobile hosts, *Proceedings - International Conference on Distributed Computing Systems*, Vancouver, Canada, pp. 136–146, 1995.

[3] S. Biaz, N. Vaidya et al, TCP over wireless networks using multiple acknowledgements, *Texas A&M University, Technical Report 97-001*, http://www.cs.tamu.edu/faculty/vaidya/papers/mobile-computing/97-001.ps, January 1997.

[4] S. Banerjee and J. Goteti, Extending TCP for wireless networks, University of Maryland, College Park, http://www.cs.umd.edu/users/suman/docs/711s97/711s97.html.

[5] H. Balakrishnan, S. Seshan, and R. H. Katz; Improving reliable transport and handoff performance in cellular wireless networks; *Wireless Networks*, 1, 4, pp. 469 – 481, February 1995.

[6] H. Balakrishnan and R. H. Katz; Explicit loss notification and wireless web performance, in *Proceedings of IEEE Globecom 1998*, Sydney, Australia, November, 1998.

[7] N. H. Vaidya, M. Mehta, C. Perkins, G. Montenegro, Delayed duplicate acknowledgements: a TCP-unaware approach to improve performance of TCP over wireless, *Technical Report 99-003*, Computer Science Dept., Texas A&M University, February 1999.

[8] K. K. Ramakrishnan and R. Jain, A binary feedback scheme for congestion avoidance in computer networks , Proc. SIGCOMM'88, pp. 303-313, August 1988, selected as the best paper and published in *ACM Transactions on Computer Systems*, Vol. 8, No. 2, pp. 158-181, May 1990.

[9] S. Floyd, TCP and explicit congestion notification, *ACM Computer Communication Review*, V. 24 N. 5, p. 10-23, October 1994.

[10] B. Braden et al, Recommendations on queue management and congestion avoidance in the Internet, *RFC 2309*, April 1998.

[11] K. Ramakrishnan and S. Floyd, A proposal to add explicit congestion notification (ECN) to IP, *RFC 2481*, January 1999.

[12] S. Floyd and V. Jacobson, Random early detection gateways for congestion avoidance, *IEEE/ACM Transactions on Networking*, Vol. 1, No. 4, pp. 397-413, August 1993.

[13] UCB/LBNL/VINT Network Simulator - ns (version 2), http://www-mash.CS.Berkeley.EDU/ns/.

[14] R. Jain, *The Art of Computer System Performance Analysis*, John Wiley and Sons Inc., 1991.

[15] C. Liu and R. Jain, Improving explicit congestion notification with the mark-front strategy, *Computer Networks*, Vol. 35, No. 2–3, pp. 185–201, February 2001.

[16] S. Floyd, The gentle option in ns, http://www.aciri.org/floyd/red/gentle.html, March 2000.