Wireless Networks 7, 221–236, 2001
© 2001 Kluwer Academic Publishers. Manufactured in The Netherlands.

Use of TCP Decoupling in Improving TCP Performance over Wireless Networks

S.Y. WANG [*]
Department of Computer Science and Information Engineering, National Chiao Tung University, Hsinchu, Taiwan

H.T. KUNG [**]
Division of Engineering and Applied Sciences, Harvard University, Cambridge, Massachusetts, USA

Abstract. We propose using the TCP decoupling approach to improve a TCP connection's goodput over wireless networks. The performance improvement can be analytically shown to be proportional to $\sqrt{\mathrm{MTU}/\mathrm{HP\_Sz}}$, where MTU is the maximum transmission unit of participating wireless links and HP_Sz is the size of a packet containing only a TCP/IP header. For example, on a WaveLAN [32] wireless network, where MTU is 1500 bytes and HP_Sz is 40 bytes, the achieved goodput improvement is about 350%. We present experimental results demonstrating that TCP decoupling outperforms TCP reno and TCP SACK. These results confirm the analysis of $\sqrt{\mathrm{MTU}/\mathrm{HP\_Sz}}$ performance improvement.

Keywords: TCP, wireless networks, performance

1. Introduction

TCP [28] is the most widely used protocol for providing reliable and in-sequence transport service between two hosts over the Internet. Recently, wireless networks have grown significantly, and as a result, studying and improving TCP's performance over wireless networks has become an active area of research.

The current TCP congestion control design does not allow TCP to perform well over unreliable wireless links. When one packet of a TCP connection is lost, the sending rate of the TCP connection must be reduced by at least 50%. (The rate reduction for TCP reno and TCP SACK is 50%. For TCP tahoe, it can be much higher than 50%. This paper assumes that TCP reno is used.) When multiple packets in a TCP's congestion window are lost, the TCP connection is very likely to time-out for more than one second. This congestion control design works well to prevent congestion in a network in which links (such as fiber optics) have very small bit-error-rates (BERs) (e.g., 10^{-12}). In such an environment, packet losses mostly result from packet dropping due to router buffer overflow during congestion. However, in a network in which links (such as wireless links) have large BERs (e.g., 10^{-6}), and therefore, packet losses may result from both packet dropping due to congestion and packet corruption due to link errors, TCP's congestion control design will mistakenly and unnecessarily reduce a TCP connection's sending rate when its packets get corrupted and lost due to link errors. The result of these wrong control decisions is that the achievable throughput of a TCP connection over wireless networks with large BERs can be very poor.

To achieve improved throughput and at the same time implement congestion control in a lossy wireless network, a TCP connection needs to distinguish packet losses caused by congestion from those caused by corruption. For packet losses caused by congestion, the TCP congestion control should reduce the TCP connection's sending rate to help remove the current congestion. For packet losses caused by corruption, the TCP congestion control should not be invoked. That is, the lost packets should be retransmitted but the current sending rate should not be reduced. Recall that TCP's congestion control uses an additive-increase (when there is no congestion loss) and multiplicative-decrease (when there is a congestion loss) algorithm [10,21] to control a TCP connection's sending rate. When a packet is corrupted and lost, the sending rate of the TCP connection should still keep increasing, rather than being reduced, until one of its packet is lost due to congestion.

However, to distinguish packet losses caused by congestion from those caused by corruption is challenging. Many approaches have been proposed aiming at improving TCP performance over wireless networks. Section 2 will briefly review these approaches.

This paper proposes the TCP decoupling approach to improve TCP's performance over lossy wireless networks. Instead of attempting to distinguish packet losses caused by congestion from those caused by corruption, this approach uses tiny (40 bytes) TCP/IP header packets to implement TCP congestion control for a stream of large data packets, which are responsible for carrying the user application's data. Since the packet-error-rate (PER) is proportional to the packet size [13,14] and the packet size of these tiny header packets is small, the chance that a tiny header packet gets corrupted and therefore TCP's congestion control gets wrongly triggered becomes small as well. As a result, TCP's

[*] This work is based on the author's Ph.D. thesis [31] at Harvard University.

[**] The first author's Ph.D. thesis advisor.

goodput improves in the TCP decoupling approach, and the performance improvement can be analytically shown to be proportional to $\sqrt{\mathrm{MTU/HP\_Sz}}$, where MTU is the maximum transmission unit of participating wireless links and HP_Sz is the size of a packet containing only a TCP/IP header.

The rest of this paper is organized as follows. Section 2 surveys previous work. Section 3 analyzes how a nonzero BER limits the maximum throughput that a TCP connection can possibly achieve. Section 4 describes the design and implementation of our TCP decoupling approach. Section 5 presents the reliable decoupling socket approach, which is a direct application of the TCP decoupling approach for improving TCP performance over lossy wireless networks. Section 6 discusses the strategies that are currently used in the error control of the reliable decoupling socket approach. Section 7 explains why the reliable decoupling socket approach can achieve a significant improvement on TCP's goodput. Section 8 presents experimental results and shows that the TCP decoupling approach outperforms TCP reno and TCP SACK. Section 9 compares the TCP decoupling approach to other approaches and points out its advantages over them. Section 10 discusses some future work that can further improve the performance of the TCP decoupling approach. Finally, section 11 concludes this paper.

2. Related work

This section briefly summarizes some approaches that have been proposed to improve TCP performance over wireless networks.

2.1. Link-layer schemes (e.g., [1,14]*)*

Forward error correction (FEC) schemes can be used to reduce the effective BER of a wireless link at the expense of reduced bandwidth and a requirement for high processing power to encode and decode packets. Automatic Request-Repeat (ARQ) can be used to automatically retransmit lost packets at the link layer to hide packet loss from the sender of a TCP connection at the expense of increased delay and delay variations, and introduced packet reordering. These two schemes can be combined to improve the quality of a wireless link.

2.2. Snooping protocol (e.g., [6]*)*

If only the last hop to a mobile host is a wireless link, a TCP-aware agent can be run on the base station to snoop passing TCP packets and do some local control. For example, by caching recently transmitted TCP packets sent to a mobile host and monitoring the acknowledgment packets returning to the sender of a TCP connection, the snooping agent can quickly resend a cached copy of a lost packet to the mobile host if it observes that more than three duplicate acknowledgment packets are sent back to the sender of a TCP connection. Snooping protocols of this kind have some drawbacks. First, the agent must be TCP-aware. As a result, this scheme is protocol dependent and can not work for other existing protocols or future protocols when they become available. Second, the snooping performance overhead is high. Third, although a lost packet can be retransmitted locally by the base station, the generated three duplicate acknowledgment packets still reach the sender of the TCP connection and cause the sender to unnecessarily reduce its sending rate by 50%.

2.3. Split connection (e.g., [4]*)*

If only the last hop to a mobile host is a wireless link, a TCP connection to a mobile host can be split into two connections. The first one starts at the sender of the TCP connection and ends at the base station. The second one starts at the base station and ends at the mobile host. Since the second TCP connection is explicitly used for the wireless link where packet losses are solely due to corruption, not congestion, it can be fine tuned to improve TCP performance on the wireless link. A shortcoming of this kind of schemes is that the end-to-end semantic of TCP is violated. For example, this will cause connection re-establishment problems when the mobile host switches to other base stations during a TCP connection's life time.

2.4. Explicit loss notification (e.g., [5,29]*)*

Like the snooping protocols described earlier, a TCP-aware agent is run on the base station to watch passing TCP packets to deduce that there may be a packet lost due to corruption. It then sets a special bit in the returning acknowledgment packets to notify the sender of a TCP connection that the recent packet loss may be a result of corruption, not congestion. When detecting this bit, the sender will not reduce its sending rate by 50%. This scheme is an improved version of snooping protocols but still has some drawbacks. First, the agent must be TCP-aware and, as pointed out above, this scheme is protocol dependent and can not work for other existing protocols or future protocols. Second, the snooping performance overhead is high. Third, the TCP congestion control at the sender of a TCP connection needs to be modified and becomes more complicated.

2.5. Modifications to TCP (e.g., [2]*)*

TCP SACK can be used to recover from multiple packet losses in a window without timing-out. The same TCP congestion control algorithms, but with different parameters, can be used to transmit data more aggressively. For example, at the TCP receiver, "ack every other packet" can be changed to "ack every packet" to increase a TCP connection's ramp up speed. A drawback of this approach is that these modifications to TCP may result in a TCP protocol that is too aggressive and thus harmful to congestion control in a network.

3. Effect of bit-error-rate on a TCP connection's maximum achievable throughput

This section analyzes the effects of a nonzero BER on the maximum bandwidth an idealized TCP connection can possibly achieve. An idealized TCP connection is defined as a TCP connection whose fast retransmit and recovery mechanism always works on packet losses and never times-out.

Consider a typical BER value of $3 \cdot 10^{-5}$ for a wireless link [11]. Assume that, just for this analysis, the packet size PS is a typical MTU of 576 bytes, and that bit errors are uniformly distributed in packets [11]. Then the Packet Error Rate is approximated as

$$\mathrm{PER} = \mathrm{BER} \cdot \mathrm{PS} = 3 \cdot 10^{-5} \cdot 576 \cdot 8 = 0.14. \qquad (1)$$

This means that on average one corrupted packet is expected to occur in every 1/PER = 7.2 packets.

We calculate the Maximum Allowable Window (W) in packets and the Maximum Allowable Throughput (MAT) in bits per second assuming that there is no TCP time out. That is, this analysis assumes that a corrupted packet can always be recovered by the fast retransmit and recovery mechanism. When a packet is corrupted and lost, TCP's congestion control will cut its current congestion window size W to $W/2$, and then increase the congestion window size by one packet every round-trip time (RTT) until one packet is corrupted and lost again. Since bit errors are uniformly distributed over the stream of TCP packets, the number of transmitted packets between two packet corruptions (called a "cycle" in the following discussion) is roughly the same. Then, there exists a W such that, in each cycle, the TCP connection's window size will grow from $W/2$, $(W/2)+1, \ldots,$ to $(W/2+W/2)$, then drop back to $W/2$. In each cycle, the total number of packets transmitted between two packet corruptions (losses) is, thus, $W/2 + (W/2+1) + \cdots + (W/2+W/2) = (3/8)W \cdot W + 3W/4$.

Therefore,

$$\frac{1}{\mathrm{PER}} = \frac{W}{2} + \left(\frac{W}{2}+1\right) + \cdots + \left(\frac{W}{2}+\frac{W}{2}\right) = \frac{3}{8}W \cdot W + \frac{3W}{4}. \qquad (2)$$

Based on equation (2), given a PER value, W can be solved. For example, when PER is 0.14, W is about 4.

Note that the congestion window grows by one packet per RTT. Thus, a total of $(3/8)W \cdot W + 3W/4$ packets are sent over $(W/2) \cdot \mathrm{RTT}$ time as depicted in figure 1. This implies that

$$\mathrm{MAT} = \left(\frac{3}{8}W \cdot W + \frac{3W}{4}\right) \cdot \mathrm{PS} \cdot \frac{8}{(W/2) \cdot \mathrm{RTT}} = \frac{3}{4} \cdot W \cdot \mathrm{PS} \cdot \frac{8}{\mathrm{RTT}} + \frac{3}{2} \cdot \mathrm{PS} \cdot \frac{8}{\mathrm{RTT}}, \qquad (3)$$

where RTT is the round-trip time, in seconds, for the TCP connection.

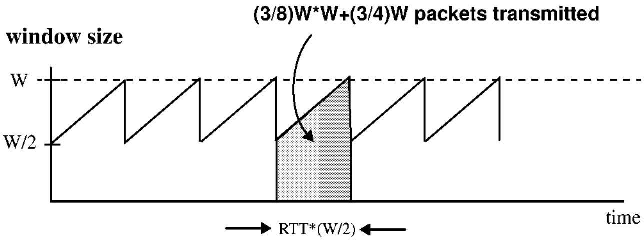


Figure 1. TCP's saw-tooth window growing and shrinking.

Based on equation (3), for example, when $W = 4$, $\mathrm{RTT} = 0.540$ and $\mathrm{PS} = 576$,

$$\mathrm{MAT} = 26 \text{ kbps}. \qquad (4)$$

Using a retransmission packet loss detection algorithm, Samaraweera and Fairhurst [29] reported that their method can achieve an optimal throughput of about 26 kbps under similar assumptions about BER, packet size and RTT. Their empirical results are consistent with equation (4) of our analysis.

The MAT value of equation (3) results from link errors rather than network congestion. Thus, the MAT value will hold even when there is no congestion in the network and the link bandwidth is infinitely large. The severity of the problem increases when RTT is large, as in the case of satellite communications [2,30]. This poor TCP throughput presented in the analysis is a consequence of incorrectly applying TCP congestion control algorithms to a situation where packet losses are due to link errors, rather than congestion.

4. The TCP decoupling approach

In the TCP decoupling [23,31], TCP congestion control is applied to a data packet stream without actually transporting data packets over a TCP connection. A TCP connection using the same network path as the data packet stream is set up separately and the transmission rate of the data packet stream is then associated with that of the TCP packets. Since the transmission rate of these TCP packets is under TCP congestion control, so is that of the data packet stream. Because the data packet stream is not transported on a TCP connection, the regulated data packet stream needs not be subject to properties caused by TCP error control such as automatically retransmitting lost packets at the TCP sender and delaying already arrived out-of-order packets in the TCP receiver's assembly queue. These properties may not be desirable for all data packet streams.

Because the TCP decoupling approach decouples TCP congestion control from TCP error control and allows them to be separately and independently performed, this approach has several important applications [31]. The first application is to provide TCP trunking service [23] for MPLS label-switched paths [9] and ATM virtual circuits [26]. The second is to provide a 100% TCP-friendly congestion control scheme for multimedia streaming applications. The third is to improve TCP performance over wireless networks as described in this paper.

The TCP decoupling approach has been implemented on FreeBSD 2.2.8. Extensive experimental results have been collected on a network testbed consisting of up to sixteen 400 MHZ Pentium PCs at Harvard University.

This paper focuses on the application of the TCP decoupling approach in improving TCP performance over wireless networks. This section presents the design and implementation of the TCP decoupling approach in detail.

4.1. Notation and terminology

This section defines some notations and terminologies which will be used throughout the paper.

- *Circuit.* A routing path over which a stream of data packets will be transmitted. Data packets to enter a circuit will be emitted into the circuit at certain rates by the sending node of the circuit. Once a data packet is emitted into the circuit, it will be forwarded by intermediate routers on the circuit path as soon as they can.
- *TCP circuit.* A circuit whose sending node uses TCP congestion control to control the emission rate of data packets into the circuit.
- *Sender and receiver of a TCP circuit.* The sender of a TCP circuit is composed of a GMB sender (section 4.5.2) and one or more senders of control TCP(s) set up for the TCP circuit. The receiver of a TCP circuit is composed of one or more receivers of control TCP(s) set up for the TCP circuit.
- *Control TCP*. A control TCP is a TCP connection set up between the sending and receiving node of a TCP circuit to regulate the emission rate of a data packet stream flowing into the circuit. The version of control TCP used in the current implementation of TCP decoupling is TCP reno. It can be any other version (e.g., TCP SACK [25]).
- *Sender and receiver of a control TCP*. The sender of a control TCP is the TCP processing module at the sending node of the control TCP. The receiver of a control TCP is the TCP processing module at the receiving node of the control TCP.
- *Header packet.* A header packet is generated and sent by the sender of a control TCP to the receiver of the control TCP. A header packet contains only a TCP/IP header, and has no data payload.
- *Control packet.* A control packet is either a header packet or an acknowledgment packet generated and sent back by the receiver of the control TCP to the sender of the control TCP.
- *Data packet (user packet).* A data packet is a packet in a network which is not a control packet. Since a data packet is normally generated by a user application program, it is also called a "user" packet in the paper.
- *GMB*. Guaranteed minimum bandwidth, in bytes per unit time.
- *VMSS*. "Virtual maximum segment size" in bytes. It is a configurable parameter.
- *HP_Sz.* Header packet size in bytes. This paper assumes HP_Sz = 52 because typically a header packet contains a 40-byte TCP/IP header and a 12-byte TCP timestamp option.

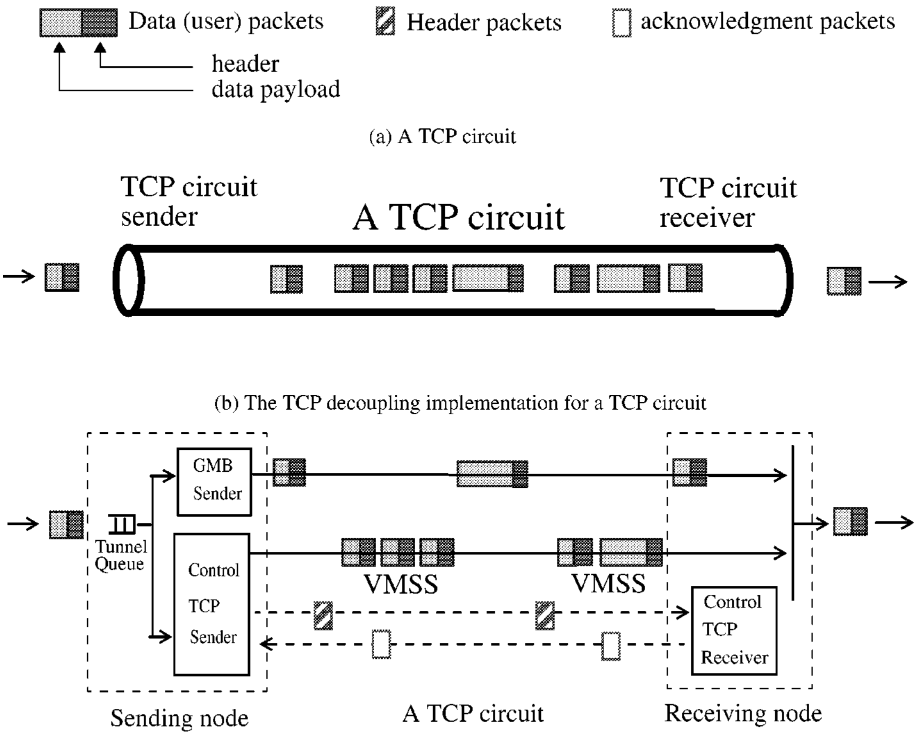
4.2. Overview of the TCP decoupling approach

The TCP decoupling is an approach to implement a TCP circuit. Figure 2(a) depicts a TCP circuit. A TCP circuit can be used as an edge-to-edge TCP trunk [23] or an end-to-end connection for wireless communication and multimedia streaming application [31]. The sending node and receiving node of a TCP circuit thus can be a router or a host.

Figure 2(b) shows how a TCP circuit is internally implemented by the TCP decoupling approach. A TCP circuit is composed of a GMB sender at its sending node and one or multiple control TCPs between its sending and receiving nodes. Data packets flowing into the TCP circuit are first directed to and stored in the tunnel queue at the sending node. The GMB sender is used when a TCP circuit is allocated a certain GMB along its path. The GMB sender unconditionally sends data packets in the tunnel queue into the TCP circuit at the GMB rate. In figure 2(b), one control TCP connection is set up between the sending node and the receiving node of the TCP circuit to probe for the available bandwidth for the data packet stream beyond the TCP circuit's allocated GMB.

Each control TCP sends out its header packets under TCP congestion control algorithms when there are data packets in the tunnel queue. These header packets each contain only a TCP/IP header and no data payload. They only need to contain a TCP/IP header because, in order to implement and use TCP congestion control, the control information exchanged and carried by the packets of a TCP connection actually is all contained in the TCP headers of these packets, and the content of the TCP data payloads of these packets are totally irrelevant and can be empty. For each transmitted header packet, the control TCP on the sending node emits data packets in the tunnel queue into the TCP circuit totaling up to VMSS bytes. The sending rate of the data packet stream, thus, is proportional to the sending rate of the header packets. Since data packets traverse the same routing path as header packets (this assumption is discussed in section 4.3 later), they will experience the same congestion level at the same place at the same time. Suppose that congestion occurs and buffer eventually overflows in a router on the path, which results in dropping of header packets, the sender of the control TCP will reduce the sending rate of its header packets, which also results in a proportional reduction in the sending rate of the data packet stream. By this method, the TCP decoupling approach achieves the goal of using TCP's congestion control to regulate the transmission rate of a data packet stream for utilizing available bandwidth.

In contrast with the traditional TCP approach in which data packets need to be carried (encapsulated) by TCP packets and thus be coupled with TCP/IP headers, in the TCP decoupling approach, data packets are transmitted as inde-



(a) A TCP circuit

(b) The TCP decoupling implementation for a TCP circuit

Figure 2. The TCP decoupling approach for implementing a TCP circuit.

pendent packets from header packets and their packet format and packet content remain unchanged. The data packet stream does not suffer from the properties caused by TCP's error control as TCP's error control is applied to the header packets only, not to the data packets.

4.3. Assumption of the TCP decoupling approach

One assumption required by the TCP decoupling approach is that the routing path taken by the data packets should be the same as the routing path taken by the header packets which control them. Obviously, if header packets take a different path than data packets, header and data packets will not experience the same congestion level at the same place at the same time in a network and, as a result, the TCP decoupling approach may fail.

However, in the current Internet, we can argue that when the TCP decoupling approach is used for end-to-end applications, the problem of using different routing paths for header and data packets is not likely to happen, and when the TCP decoupling approach is used for edge-to-edge applications, there exist solutions for it. The reasons are presented as follows.

First, for end-to-end applications such as wireless communication and multimedia streaming applications, data and header packets have the same IP destination addresses. When routing tables change, a header packet may take a different routing path than its associated data packet(s). Although this problem may happen, it affects only one pair of a header and its associated data packets totaling up to only VMSS bytes. The route change problem is expected to happen infrequently as the Internet routing protocol OSPF [20,27] only updates routing tables every 30 s or so to avoid route flapping. Another concern is about multi-path routing, which splits the load of a packet stream onto multiple routing paths for load-balancing purposes. This problem should be minimum as network researchers now understand that the minimum granularity of load-balancing should be a flow – a packet stream with the same source and destination IP addresses, otherwise a TCP connection's throughput [7] and the quality of a UDP audio/video stream will suffer due to excessive packet reordering. (For applications using TCP, packet reordering causes duplicate acknowledgment packets, which unnecessarily trigger TCP's fast retransmit algorithm, which in turn unnecessarily reduces the sending rate of a TCP connection. For multimedia applications using UDP, packet reordering increases the required buffer size to store and rearrange out-of-order packets before they can be played back at the receiving host, which also adds unnecessary delays to the playback time and degrades the quality of real-time applications such as IP phone.)

Second, consider the edge-to-edge application such as TCP trunking where the data packet stream is an aggregate stream composed of many flows each with its own different IP source and destination addresses. A TCP trunk can be associated with a layer-2 ATM [26] or Frame Relay [19] virtual circuit, or an MPLS label-switched path [3,9] to make sure that its header and data packets all take the same routing path. These TCP trunks are intended to be used in the backbone networks [23], where ATM and Frame Relay virtual circuits and MPLS label-switched paths are provided for engineering traffic. Running a TCP trunk on top of such a virtual circuit or label-switched path is feasible and well-suited.

4.4. Design goals (or properties) of the TCP decoupling approach

The design goals of the TCP decoupling approach are listed as follows:

(1) Arrivals of data packets at the sending node of a TCP circuit trigger transmissions of control (header) packets.

(2) Do not automatically retransmit lost data packets.

(3) Do not introduce packet reordering to a data packet stream.

(4) Do not introduce extra transmission delay to a data packet other than that caused by TCP's congestion control.

(5) Do not modify the content of a data packet.

(6) Do not increase the length of a data packet.

(7) Low bandwidth overhead for control packets.

(8) Simple and efficient implementation (for high throughput).

(9) Easy to set up, configure, and use.

Goal (1) is desirable because generating and sending control packets when there are no data packets to send unnecessarily waste network bandwidth. Goal (2) is desirable because different applications have different reliability requirements for their packet transfer (e.g., FTP requires reliable data transfer but video-conferencing can tolerate unreliable transfer), retransmitting data packets, if required, should be handled by the application program or some reliable protocol at the sending host. Goals (3) and (4) were discussed earlier. Goal (5) is desirable because modifying a data packet's content needs several read/write operations and a recomputation of the IP checksum, which will slow down the forwarding throughput. Goal (6) is desirable because increasing a packet's length may cause packet fragmentation when the resulting length exceeds the MTU (maximum transmission unit) of some link on which the packet need to traverse. Goal (7) means that control packets should not consume too much bandwidth. Goal (8) is desirable because a simple implementation leads to a low-cost and robust implementation, and an efficient implementation can provide high forwarding throughput on high-speed links such as OC-192 (10 Gbps) links.

The design and implementation of the TCP decoupling approach meet all of these goals. Therefore, these listed design goals are also the general properties of the TCP decoupling approach. In addition, the TCP decoupling approach can allocate bandwidth among competing TCP circuits in a fine-grain way by using different values for the VMSS of competing TCP circuits. This property will be discussed in section 4.8.1.

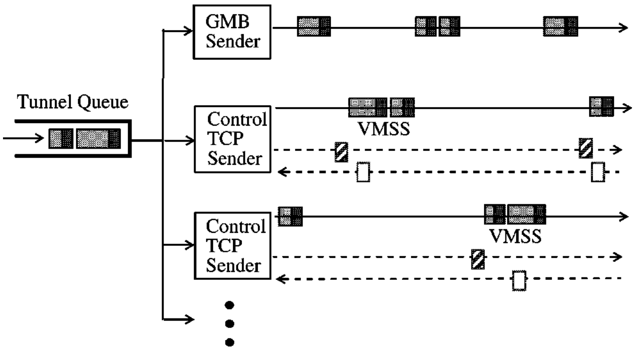


Figure 3. The architecture of the sending node of a TCP circuit implemented in the TCP decoupling approach.

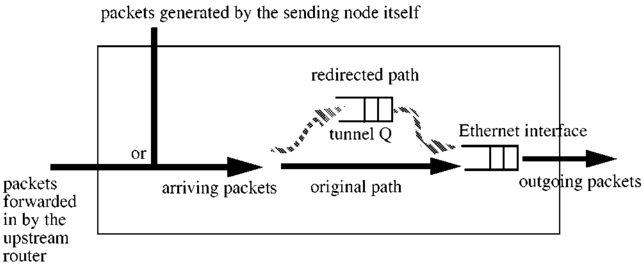


Figure 4. Redirection of arriving packets to a tunnel network interface queue, from which they will be sent out through a physical network interface (e.g., an Ethernet interface) later.

4.5. The TCP decoupling mechanism on the sending node of a TCP circuit

Figure 3 depicts the architecture of the sending node of a TCP circuit in the TCP decoupling approach. Each component will be presented in detail in the following sections.

Data packets that are to be sent into a TCP circuit are first redirected to and enqueued in a tunnel network interface queue. Later on, from the tunnel queue these packets will be dequeued and forwarded by either the GMB sender or a control TCP sender. A tunnel network interface is a pseudo network interface that does not have a real physical network attached to it [18]. Its functions, however, from the kernel's point of view, are no different from those of a normal Ethernet network interface. The tunnel interface queue serves as an input queue for temporarily holding data packets not yet forwarded out. Although using any software queue in the kernel also works for serving as an input queue, using a tunnel network interface queue has an advantage. The advantage is that, since from the kernel's point of view a tunnel network interface is like a physical network interface, redirecting arriving data packets to a tunnel interface queue can be done simply by changing just one routing entry in the sending node's routing table. When allowed by the GMB rate or TCP congestion control, the GMB sender or a control TCP sender dequeues the first redirected data packet and calls the kernel's IP packet forward function (`ip_forward()`) to forward it out. Figure 4 depicts the data packet redirection scenario.

4.5.1. Control TCP sender

The control TCP sender is the sender of a TCP connection set up between the sending and receiving nodes of a TCP circuit. In contrast with the normal usage, the control TCP sender is not an active process running at the user level. Instead, it is the socket which represents the sending endpoint of the TCP connection and the TCP processing functions, both of which are passive and reside in the kernel. The control TCP sender generates and transmits header packets, transmits data packets, and receives acknowledgment packets. All of these operations are automatically performed by the TCP processing functions, which are called by the network interrupt service routine, which in turn is invoked when a packet arrives. Since every operation is performed inside the kernel without context switching overhead between the kernel and user space, the control TCP sender operates efficiently and supports high speed forwarding.

To set up a control TCP connection between the sending and receiving nodes, like the normal usage, a user-level process at the sending and receiving nodes is run up. These two processes use the standard socket system calls such as `connect()` and `accept()` to conduct TCP's 3-way handshaking connection set up procedure. After the TCP connection is set up, the process on the sending node becomes idle and is not involved in sending header and data packets and receiving acknowledgment packets from the receiving node. Similarly, the process on the receiving node also becomes idle.

The socket send buffer [33], which is automatically allocated to the control TCP sender by the UNIX system as in the normal TCP usage, is not used in the TCP decoupling approach. This is because in the TCP decoupling approach, there is no physical data for the control TCP sender to send. Instead of working on and transporting a data byte stream formed by application data when they are written into a TCP socket send buffer as in the normal TCP usage, the control TCP sender works on and transports a "virtual data byte stream", which does not physically exist. Each packet transmitted by the control TCP sender, thus, is a packet consisting of only the TCP/IP header and contains no physical TCP data payload. They are, thus, called "header packets" in this paper. These header packets, together with the acknowledgment packets sent back by the control TCP receiver, are called "control packets" as their existence is solely for congestion control purposes, rather than for data carrying. The information carried in the TCP header of a header packet thus identifies some contiguous bytes of the virtual data byte stream that the header packet is supposed to carry, although physically they are not carried in the header packets.

The operations on the virtual byte stream closely correspond to the operations on the byte stream formed by the data packets entering the tunnel queue. When VMSS contiguous bytes of the virtual data byte stream have just been "transported" by the control TCP sender under its TCP congestion control, the corresponding VMSS bytes of the data packet stream can now be physically forwarded. Since data packets in the tunnel queue may have many different sizes, forwarding VMSS bytes of the data packet stream actually translates to the forwarding of as many data packet as until these VMSS byte credits are exhausted. (Note that since each packet should be transmitted atomically and cannot be cut arbitrarily for transmission, sometimes credits may be left or overused by a little amount. These left credits or debits will be carried over to the next time when another VMSS bytes credits are gained again.)

One exception to the correspondence between the virtual byte stream and the byte stream formed by the data packets entering the tunnel queue is that, in case of a header packet loss, to keep TCP congestion control algorithms going, the control TCP sender must retransmit the lost "virtual" data until it is finally received by the receiving node (actually it is the lost header packet that matters). However, this retransmission operation does not result in a retransmission of the corresponding data of the data packet stream. Instead, using these new VMSS credits, more data packets totaling up to VMSS bytes are dequeued and forwarded out from the tunnel queue. This design is both desirable and simple. This design is desirable because of no automatic retransmission of lost data packets, as explained in section 4.4. This design is simple in the sense that now, since data packets need not be retransmitted, the buffer space occupied by them can be released as soon as they are dequeued from the tunnel queue and forwarded out. There is no need to keep them in the tunnel queue as is the case for a normal TCP socket send buffer. This design, therefore, allows for a simple first-in-first-out buffer system for the tunnel queue.

For each header packet sent by the control TCP sender, the "CONTROL" bit is set in the Type-Of-Service (TOS) field of its IP header to allow the routers on the TCP circuit's path to distinguish header packets from data packets and thus be able to give them different treatments. Section 4.7 discusses the useful "lossless" property enabled by the use of this bit.

To meet the design goal of "arrivals of data packets trigger transmissions of control packets" as explained in section 4.4, the generation and sending of header packets are enabled only when there are data packets in the tunnel queue and the control TCP's congestion control allows.

The sender of the control TCP uses the difference between its current congestion window size and the number of its current outstanding (not acknowledged yet) virtual bytes in the network as the credit to decide when it can forward more data packets and send them to the network. When the credit is decreased to zero or below zero, no more data packets can be forwarded and sent to the network. Following the normal TCP design, when a control TCP is initially set up or when it times out, the control TCP's congestion window size is set or reset to VMSS bytes. As a result, the control TCP sender always has VMSS bytes credits to transmit up to VMSS bytes data packets when it initially starts or restarts. For every VMSS bytes worth of data packets which have been forwarded and sent to the network, the control TCP sender sends out a header packet as if the header packet were coupled with these data packets as is performed in traditional TCP. The transmissions of these data packets precedes the transmission of their associated header packet to meet the design goal of "data packets triggers control packets". The control information carried in the TCP headers of these header packets are exactly the same as the control information that would have been generated and carried if each header packet physically carries a VMSS-byte TCP data payload from a physical byte stream. The outcomes of these header packets, either successfully received and acknowledged or lost in the network, will cause the control TCP sender to adjust its congestion window size.

Multiple control TCPs can be set up between the sending and the receiving nodes of a TCP circuit to work together on the same data packet stream flowing into the TCP circuit. The senders of these control TCPs dequeue and forward packets from the same tunnel queue as soon as their TCP congestion controls allow them to send more data into the network. Using multiple control TCP connections is for two different purposes. First, using multiple control TCPs can smooth the achieved bandwidth usage of the TCP circuit (thus, the data packet stream flowing in it). If only one control TCP is used, since TCP reduces its sending rate by 50% when any of its packets gets lost, the transmission rate of the data packet stream, which is regulated by the control TCP, will also undergo a similar rate reduction. Suppose that there are now M control TCPs. Then a 50% bandwidth reduction from any of them will only result in a reduction of the total bandwidth of the data stream by a factor of $(1/2)/M$. This smoother bandwidth change is important for the TCP trunking [23] application for which it is desirable that, in the backbone network, a trunk's achieved bandwidth not vary too much and too quickly for stability concerns. Second, using multiple TCP connections is a way of allocating available bandwidth. It is well known that, under ideal situations (e.g., when all TCP connections have the same RTT), TCP exhibits per-flow fairness property [15,16]. That is, when there are N greedy TCP flows with about the same RTT contending for available bandwidth, each one will roughly achieve $1/N$ of the available bandwidth. Using this property, a data packet stream regulated by N control TCPs can roughly achieve N times bandwidth of a data packet stream regulated by only one control TCP. Experimental results presented in section 4.8.2 demonstrate this property.

The architecture of the sending node of a TCP circuit, as shown in figure 3, has several useful properties. First, despite that multiple senders (one GMB sender plus one or multiple control TCP senders) can dequeue and forward data packets from the tunnel queue, the design maintains the packet order of the data packet stream when it flows through the sending node. That is, data packets in the tunnel queue are forwarded out in exactly the same order as they enter the tunnel queue. This in-sequence forwarding is achieved by the design that when a control TCP sender decides to dequeue and forward a data packet from the tunnel queue, the control TCP sender must already have gained at least VMSS "credits" to forward a data packet. In case when a control TCP sender wants to dequeue a data packet and its current number of credits is less than the size of the first packet in the tunnel queue (this situation may happen when VMSS is configured to be smaller than the MTU of links), the control TCP sender simply returns and waits for more credits. No data packets will be queued in a control TCP sender as there is no need to queue data packets and there is no queue in a control TCP sender. A data packet thus is sent to a network interface as soon as it is dequeued from the tunnel queue. Second, all operations (e.g., enqueueing data packets, dequeueing data packets, sending header packets, and receiving acknowledgment packets) are triggered and performed automatically in the kernel when packets arrive. This all-in-kernel design and implementation result in a high throughput system. Third, the format and content of data packets remain untouched and unchanged when they flow through the sending node.

4.5.2. GMB (guaranteed minimum bandwidth) sender

Consider the case when a data packet stream requires a guaranteed minimum bandwidth (GMB) of X bytes per ms. Assume that via bandwidth provision and connection admission control, the network guarantees to deliver this required bandwidth for the data packet stream over its routing path. This section describes how the sender of a TCP circuit sends data packets at the GMB rate while being able to send additional data packets under TCP congestion control when extra bandwidth is available.

A TCP circuit has a GMB sender at the sending node of the TCP circuit. The GMB sender is equipped with a timer and unconditionally sends some number of data packets from the tunnel queue each time the timer expires. (In the current TCP decoupling implementation, the timer is set to be 1 ms.) When sending out data packets, the GMB sender needs not send out header packets as a control TCP sender does. Since the data packet stream has been allocated a certain bandwidth as its GMB, there is no need for the GMB sender to send out header packets to probe for available bandwidth. When the timer expires, if there are data packets in the tunnel queue, the GMB sender will send some of them under the control of a leaky bucket algorithm. The objective here is that, for any time interval of Y ms, if there is a sufficient number of bytes to be sent from the tunnel queue, the total number of bytes actually sent by the GMB sender will approach the target of $X \cdot Y$.

For each expiration of the GMB timer, the GMB sender will try to send all the data packets it is supposed to send. If there are still some data packets left in the tunnel queue, they will be sent out under the congestion control of the control TCP sender(s). In this manner, the data packet stream will always receive its GMB under the control of the GMB sender, and at the same time dynamically acquire additional bandwidth under the congestion control of the control TCP(s).

4.6. The TCP decoupling mechanism on the receiving node of a TCP circuit

Figure 5 depicts the architecture of the receiving node of a TCP circuit implemented in the TCP decoupling approach. A control TCP receiver is the receiver of a TCP connection set up between the sending and receiving nodes of a TCP circuit. In contrast with the normal usage, the control TCP receiver is not an active process running at the user level. Instead, it is the socket which represents the receiving endpoint of the TCP connection and the TCP processing functions, both of which are passive and reside in the kernel. The control TCP receiver receives header packets sent by its corresponding control TCP sender, and for each received header packet, the control TCP receiver views it as a TCP packet carrying VMSS-byte data payload, although physically there is no data payload coupled with the header packet. The control TCP receiver processes received header packets and acknowledges their receipt by sending out acknowledgment packets using the normal TCP cumulative acknowledgment scheme. Since there is no real data payload carried in these received header packets, the control TCP receiver needs not do a checksum test on the data payload, nor does it need to insert any data to its socket receive buffer. Receiving header packets and sending back acknowledgment packets are automatically performed by the TCP processing functions, which are called by the network interrupt service routine, which in turn is invoked when a packet arrives. Since every operation is performed inside the kernel without context switching overhead between the kernel and user space, the control TCP receiver operates efficiently and supports high speed forwarding.

To set up a control TCP connection between the sending and receiving nodes, like the normal usage, a user-level process at the receiving node is run up. This user-level process works with the user-level process at the sending node to conduct TCP's 3-way handshaking connection set up procedure. After the TCP connection is set up, the user-level process at the receiving node becomes idle and is not involved in receiving header and data packets and sending acknowledgment packets to the sending node.

Multiple control TCP receivers can be used (as depicted in figure 5), each corresponding to a control TCP sender at the sending node, to achieve the properties enabled by using multiple control TCP connections described in section 4.5.1.

The design of the architecture of the receiving node of a TCP circuit has many useful properties. First, arriving data packets, either sent under the control of the GMB sender or the control TCP sender(s) at the sending node, are forwarded automatically by the kernel based on the IP destination addresses contained in their own TCP/IP headers. These data packets are forwarded in exactly the same way they would be forwarded in a normal router. No further processing on these data packets is needed. The control TCP receiver is not involved in the forwarding of these arriving data packets. (Actually, the control TCP receiver does not even know when a data packet will arrive, nor does it know when a data packet has been forwarded out.) This design makes forwarding a data packet as fast as when the TCP decoupling approach is not used and results in a low-latency and high-throughput system. Second, the design maintains the packet order of a data packet stream when it flows through the receiving node. Since data packets arrive in a sequential order and, as described above, each one can be forwarded out immediately, data packets thus will be forwarded out in exactly the same order as they arrive. Third, the content and format of data packets remain untouched and unchanged by the control TCP receiver when they flow through the receiving node.

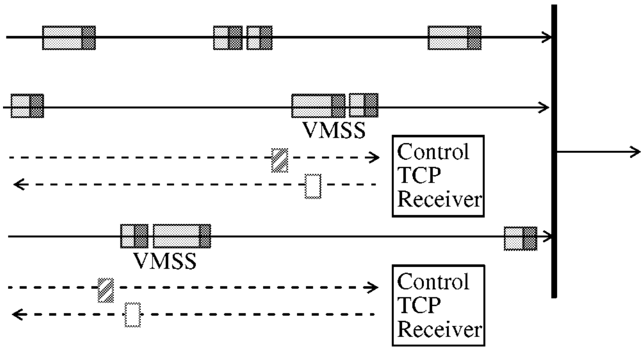


Figure 5. The architecture of the receiving node of a TCP circuit implemented in the TCP decoupling approach.

4.7. Router buffer management scheme for the TCP decoupling approach

In the TCP decoupling approach, the requirement for a router's buffer system can be as simple as a single FIFO queue shared by both data and header packets. A single FIFO queue allows for a simple and low-cost buffer system and preserves the order of arriving packets. The TCP decoupling approach can offer a unique and useful property which we call the "lossless" property. This property prevents data packets from being dropped during congestion and can be achieved when the buffer management system of every router employs a special packet dropping method that drops header packets first before dropping data packets when congestion occurs. If the "lossless" property is not required or there are problems with deploying special packet dropping methods in routers, the common FIFO or RED [17] packet dropping method can be used in the routers.

The "lossless" property is useful in a wired network (e.g., an optical network) with very small BERs such as 10^{-12}. The reason is that as long as we can prevent data (user) packets from being dropped inside routers due to congestion, they can be guaranteed with a high probability for their successful arrivals at their destinations. The "lossless" property, however, is not very useful in a lossy wireless network with large BERs such as 10^{-5} because even though data packets can be prevented from being dropped due to congestion, they can still be dropped due to link errors. For this reason, when the TCP decoupling approach is applied to improve TCP per-

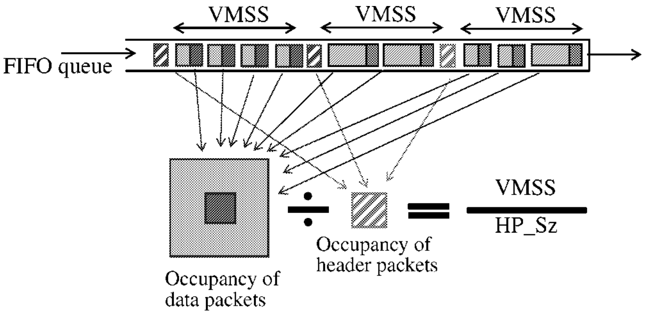


Figure 6. A FIFO buffer in a router occupied by both data and header packets.

formance over wireless networks, routers can just use the common FIFO or RED packet dropping methods.

For completeness, in this paper, we will present the special packet dropping method. However, this method will only be briefly presented. A more detailed presentation can be found in [23,31].

4.7.1. *The special packet dropping method*

To prevent data packets from being dropped inside a router when congestion occurs, the router's buffer management system uses the following two principles:

- When the FIFO queue buildup occurs, drop some incoming header packets early enough so that their control TCP senders can reduce their rates of sending data packets in time.
- Allocate sufficient buffer space for data packets to accommodate temporary buffer usage fluctuation caused by the control delay and possible arrival of new flows.

Figure 6 depicts the router buffer architecture which uses a single FIFO queue. It shows that, when there is no GMB traffic (the data packets sent under the control of GMB senders), the buffer space occupied by data packets is proportional (VMSS/HP_Sz times) to that occupied by header packets. Because of this property, controlling the maximum number of bytes of data packets in the FIFO can be achieved by limiting the maximum number of header packets in the FIFO (because the size of each header packet is the same, instead of limiting the maximum number of bytes occupied by header packets, we can simply limit the maximum number of header packets in the FIFO). Thus, by properly controlling the maximum number of header packets so that the total buffer usage of the header and data packets is always below the provisioned buffer size, the TCP decoupling approach can achieve the "lossless" property for data packets.

The maximum number of header packets in the FIFO is controlled by dropping them when the number exceeds a certain threshold. Since header packets are generated by control TCP(s), dropping header packets will cause the senders of their corresponding control TCPs to reduce their sending rates. As a result, the buffer occupancy of header packets will drop below the threshold again and thus be maintained near the threshold.

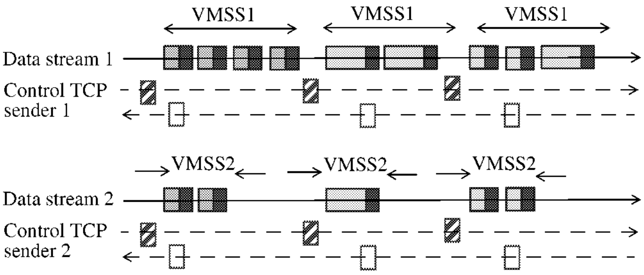


Figure 7. Although control TCP sender 1 and sender 2 achieve the same bandwidth for their header packets, the ratio of the achieved bandwidth for data stream 1 to that of data stream 2 is VMSS1/VMSS2.

4.8. *Discussions about the TCP decoupling approach*

4.8.1. *Allocating different bandwidths to TCP circuits by configuring VMSS*

Control TCPs can use different VMSS values so that the data streams they control can share the available bandwidth in different proportions. Due to the fact that, in the TCP decoupling approach, when congestion occurs routers will drop header packets before dropping data packets, each competing control TCP will receive the same bandwidth for their header packets regardless of its VMSS (the number of bytes of data packet associated with a header packet). As depicted in figure 7, suppose that the VMSS values of control TCP sender 1 and control TCP sender 2 are VMSS1 and VMSS2, respectively. Then the ratio of the achieved bandwidth for data stream 1 to that of data stream 2 will be VMSS1/VMSS2. Because of this property, by configuring different VMSS values for different control TCPs, one can allocate available bandwidth to the competing data streams in different proportions.

4.8.2. *Routers using the FIFO or RED packet dropping method*

This section presents experimental results showing that even when routers do not use the special packet dropping method presented in section 4.7.1 (and, as a result, both header and data packets may be dropped during congestion), TCP circuits can still compete fairly with each other and achieve their fair shares of available bandwidth. This property is desirable and important because, as we point out in section 4.7, in lossy wireless networks the "lossless" property is not critical. It would be useful that routers can just use the common FIFO or RED packet dropping methods so that TCP decoupling approach can be widely applied.

In the experiments, we let two TCP connections (TCP 1 and TCP 2) compete for the bandwidth of a shared 10 Mbps link. The router where the traffic of TCP 1 and TCP 2 merge uses the FIFO packet dropping method and has a buffer size of 50 packets. The RTTs of TCP 1 and TCP 2 are about 1 ms. Each experiment lasts 5 min. These TCP connections may be traditional TCP connections or TCP circuits with different VMSS values and different number of control TCPs. We present four experimental cases. The values of (X, Y) presented below are for TCP 1 and TCP 2, respectively.

Case 1. Normal TCP connections. Bandwidths achieved are (645 KB/s, 582 KB/s).

Case 2. TCP circuits. GMBs are (0 KB/s, 0 KB/s). VMSSs are (1500, 1500). Number of control TCPs per TCP circuit are (1, 1). Bandwidths achieved are (621 KB/s, 567 KB/s).

Case 3. TCP circuits. GMBs are (0 KB/s, 0 KB/s). VMSSs are (1500, 1500). Number of control TCPs per TCP circuit are (2, 1). Bandwidths achieved are (773 KB/s, 399 KB/s).

Case 4. TCP circuits. GMBs are (0 KB/s, 0 KB/s). VMSSs are (3000, 1500). Number of control TCPs per TCP circuit are (1, 1). Bandwidths achieved are (750 KB/s, 454 KB/s).

By comparing case 1 to case 2, we see that even under the common FIFO packet dropping method, TCP circuits still fairly compete with each other. By comparing case 2 to case 3 and comparing case 2 to case 4, we see that using different values of VMSS and using different number of control TCPs for a TCP circuit is still effective in allocating bandwidth even under the common FIFO packet dropping method. To save space, similar experimental results under RED packet dropping method are not shown here.

4.8.3. Control packet overhead

Header packets, sent by a control TCP sender, are regarded as bandwidth overhead in the TCP decoupling approach because they do not carry data payloads and their existence is solely for congestion control purposes. The control TCP sender sends one header packet per VMSS-byte worth of data packets. Assume a typical situation where each header packet has HP_Sz $= 52$ bytes (40 bytes for the TCP/IP headers and 12 bytes for the TCP timestamp option) and VMSS is 1500 bytes (Ethernet and WaveLAN's MTU). Then the header packet overhead ratio for data packets sent by the control TCP sender is HP_Sz/VMSS $= 52/1500$, which is about 3.4%. In the reverse direction, acknowledgment packets sent by the control TCP receiver to the control TCP sender is also regarded as bandwidth overhead. Because in the TCP decoupling design (and also in the normal TCP design), a control TCP receiver sends back an acknowledgment packet for every other header packet, the acknowledgment packet overhead ratio is about (3.4%/2), which is 1.7%. In total, the control packet overhead is 5.1%. The ratio can be lowered by increasing VMSS to a larger value.

VMSS can be larger than the path MTU without risking the possibility of packet fragmentation because the VMSS-byte worth of data packet(s) associated with a header packet is not sent out as a single IP packet of VMSS bytes. Instead, the data is sent as a sequence of separate data packets that are already queued in the tunnel queue at the sending side of a TCP circuit (see figures 2 and 3). Traditionally it is unfavorable to use a large MSS (maximum segment size) to transfer a big chunk of packet in a network because, during its lengthy transmission, a packet with a higher priority such as voice cannot be transmitted. Also, in a lossy wireless network, the PER of a large packet is also higher than that of a small packet. In the TCP decoupling approach, using a large VMSS does not have these problems. Since VMSS bytes of data is actually sent as a sequence of separate data packets, not a single VMSS-byte large packet, a high-priority packet can cut in and be transmitted as soon as the ongoing transmission of a data packet is finished. For the same reason, the PER remains the same regardless of the value of VMSS.

5. Reliable decoupling socket approach for improving TCP performance over lossy wireless networks

The reliable decoupling socket approach described in this section is a direct application of the basic TCP decoupling approach. In this approach, TCP congestion control and TCP error control are independently and separately applied to a stream of data packets. As figure 8 depicts, a TCP connection is set up as usual between the sending and the receiving hosts to reliably transport data from the sending host to the receiving host. It is called "data TCP connection", or more briefly, "data TCP". Its sole function is to transport data. A TCP circuit is then set up between the same sending and the receiving hosts. The data packet stream generated by the sender of the data TCP is then sent into the TCP circuit.

The sender and receiver of the data TCP handle only error control. Their TCP congestion control is disabled and the data packets generated by the sender of the data TCP can be sent into the TCP circuit at the maximum speed allowed by the TCP circuit (i.e., as long as the tunnel queue of the TCP circuit is not full). The TCP circuit uses TCP congestion control to probe for available bandwidth in networks via its tiny header packets. The TCP circuit's congestion control is triggered only when its tiny header packets are corrupted and lost. A corrupted and lost data packet will not trigger the TCP circuit's congestion control. Because the PER of these tiny header packets is much smaller than that of full-size packets carrying MTU data payload, the probability of

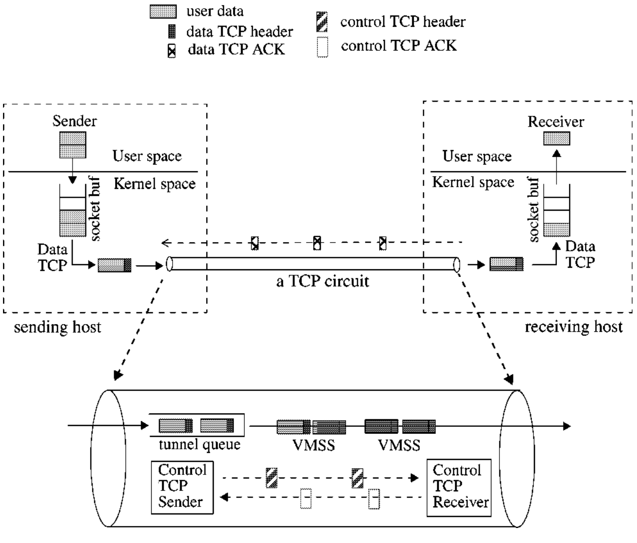


Figure 8. Implementation of the reliable decoupling socket.

mistakenly triggering TCP congestion control to reduce the sending rate upon packet corruption is significantly reduced. The reliable decoupling socket approach thus provides a reliable and high throughput data transfer over lossy wireless network while using TCP congestion control to avoid network congestion.

The reliable decoupling socket on each of the sending and receiving hosts is implemented internally as two TCP sockets – one control and one data sockets. The control socket is associated with the control TCP. The data socket is associated with the data TCP, on which user application's data is transmitted. Following the decoupling principle, data packets will be sent at rates under the control TCP's congestion control. The data socket is provided to the application user for transmitting the user's data whereas the control socket is hidden and invisible to the user.

While the control TCP controls sending rates for data packets, the data TCP is responsible for retransmitting corrupted or lost application data. In our current implementation, the data TCP makes direct use of TCP's existing facilities such as sequence numbers and triggering mechanisms for packet retransmission. (This is not absolutely necessary. Other retransmission schemes can also be used.) The data TCP does not deal with congestion control. Its congestion window size (cwnd) is always set to infinity, except when a lost packet needs to be retransmitted. When retransmitting a lost packet, the data TCP will temporarily set the congestion window size (`cwnd`) to one MSS so only one packet is retransmitted. After retransmitting the lost packet, the `cwnd` is reset to infinity. At the sender of the data TCP, outgoing data packets are redirected and sent to the tunnel queue of the TCP circuit. The sender can send its data packet to the tunnel queue as fast as it can as long as the tunnel queue does not overflow.

6. Discussions on data TCP's error control

Experimental results on testbed networks show that it is important for the data TCP to be aggressive in retransmitting lost data, as long as their sending is allowed by the congestion window of the control TCP. Otherwise, timeouts on the data TCP could happen easily, and performance can degrade drastically. To achieve the high goodputs reported in section 8, the data TCP in the current implementation has the following features:

F1. The receiver uses the SACK option [25] to report to the sender up to three missing packets in an acknowledgment packet.

F2. The sender retransmits the first unacknowledged packet every time when some number X of additional duplicate acknowledgment packets are received [24]. The number X is the current window size of the control TCP. Thus, the method will retransmit again a previously retransmitted packet should it get corrupted or lost. This method can minimize chances of timeout.

F3. The sender uses a fine-grain retransmission timer of 50 ms, rather than the system default of 500 ms. The timer's exponential backoff is disabled.

Features F1 and F2 greatly reduce possible timeouts of the data TCP. Should timeouts still happen, F3 will minimize the negative impacts of timeouts on performance.

It is important to emphasize that the data TCP will send applications data under these aggressive send features, only when the sending is allowed by the congestion window of the control TCP. In the reliable decoupling socket implementation, the control TCP uses TCP reno, a normal congestion control algorithm, with a default coarse-grain retransmit timer of 500 ms, exponential backoff enabled, and the normal 3-duplicate acknowledgment packets trigger of fast retransmission. That is, the control TCP does not employ any aggressive feature such as F1, F2 and F3 at all. Since the control TCP is not aggressive and it controls the sending rate of the data TCP, the use of these aggressive retransmission features by the data TCP causes no harm to other network users.

However, the data TCP should not be unnecessarily aggressive. Otherwise, retransmission may become excessive and will hurt the overall goodput of the data TCP. For feature F2 above, the number X is linked to the current window size of the control TCP to reduce the chance of premature retransmission due to an unnecessarily small X.

7. Why the reliable decoupling socket approach can improve TCP performance

Before presenting experimental results showing performance improvements provided by the reliable decoupling socket approach, in this section, we explain why the reliable decoupling socket approach improves TCP performance over wireless networks.

In the reliable decoupling socket approach, as shown in figure 8, it is the control TCP that controls the sending rate of the data TCP's packets. The data TCP uses only TCP error control, but not TCP congestion control, to retransmit lost data packets or to transmit data packets as fast as the control TCP allows. Note that the header packets, which are sent by the control TCP, are now the only packets whose losses will trigger TCP's congestion control to reduce the data TCP's sending rate. Due to their small packet length of only 52 bytes, the chance of incorrectly triggering TCP congestion control is significantly reduced. As a result, the overall negative impact caused by incorrectly triggering TCP congestion control is also reduced. In [13,14], experimental results demonstrate that packet error rates decrease with packet sizes.

An analysis for the TCP decoupling approach, which is similar to that in section 3 for a normal TCP connection, is presented here. Since it is the corrupted header packets rather than the data packets that will mistakenly cause TCP congestion window size to reduce, the size of header packets, rather than the combined size of both a header and data payload, should be used in computing W. When using a packet size of 52, instead of 576 used earlier, the computed W now becomes 14 instead of 4. Computing MAT using this new value of $W = 14$ and the original packet size of 576 results in a new MAT of 119 kbps rather than its old value of 26 kbps in equation (4) – a speed up of $119/26 = 4.57$!

In this case, the increase of W from 4 to 14 is more significant than just an increased MAT. A window size around 4 packets is hardly sufficient for supporting the fast retransmit and recovery mechanism because the mechanism relies on receiving three duplicate acknowledgment packets to trigger the fast retransmission of a lost packet. If fast retransmit and recovery mechanism is usually not triggered, the TCP connection will experience frequent timeouts. These TCP timeouts will severely impair TCP's performance in throughput, delay and fairness. An increase of the window size to a sufficiently large value such as 14 eliminates this timeout problem.

According to equations (2) and (3), the TCP decoupling approach achieves a performance improvement proportional to $\sqrt{\mathrm{MTU/HP\_Sz}}$ over the normal TCP approach. It is obvious that if HP_Sz can be further reduced, the TCP decoupling approach will achieve an even higher performance improvement. Actually, it is the effective PER of header packets that matters, as the ultimate goal is to reduce the effective PER of header packets to zero so that no congestion control will be wrongly triggered. Section 10 will discuss some schemes that can either physically reduce HP_Sz or reduce the effective PER of header packets.

8. Experimental results

8.1. Descriptions of experiments

On our testbed network described in figure 9, there are two user TCP connections (one from node A to E, and the other from node B to F) contending for the bandwidth of a wireless link between C and D, which is simulated by a wired Ethernet. The experiments use an Ethernet link to simulate a wireless link, rather than directly using a real wireless link such as a WaveLAN network [32]. This is because the experiments need to precisely generate and control the desired BERs, and it is hard to do so using a real wireless link. Besides, WaveLAN implements IEEE 802.11 protocol [11] and thus employs ARQ to automatically retransmit a corrupted packet up to 4 times. Because the experiments want to clearly identify and evaluate the TCP decoupling approach's performance without ARQ's interference, the experiments did not use WaveLAN networks[1].

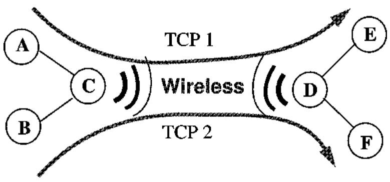


Figure 9. Testbed network.

As described in section 5, each of these two competing connections is internally implemented as a pair of data and control TCPs. Both the control and data TCPs use TCP reno and the data TCPs are enhanced with features F1, F2 and F3 of section 6. Performance numbers on TCP SACK are obtained from hosts running Windows 98, which has a built-in version of TCP SACK. The experiments focus on the aggregate goodputs (measured at the application layer) of these two connections under varying BERs and RTTs on the simulated wireless link. In order to generate a given BER, bit errors were randomly generated on the simulated wireless link according to a given BER [11]. The size of data packets is 1500 bytes (Ethernet and WaveLAN's MTU) and the size of the header packets is 52 bytes (40 bytes TCP/IP header + 12 bytes TCP timestamp option).

8.2. Reliable decoupling socket experiments suite 1

The top curve in figure 10 is a theoretical upper bound on the goodput that any TCP scheme can possibly achieve over a 10 Mbps lossy link. This curve is obtained by using goodput = maximum_link_goodput $\cdot$ (1 − packet_error_rate). Since packet_error_rate, i.e., PER, must increases as BER increases, the theoretical goodput upper bound decreases as BER increases. No TCP scheme can achieve a better goodput than this upper bound because this upper bound is derived on the assumptions that packet retransmissions take no time and packet retransmissions do not consume bandwidth.

We see that when BER increases to $20 \cdot 10^{-7}$, TCP reno's goodput already drops drastically to only about 250 KB/s. The reason is that TCP reno generally cannot recover from multiple packet dropping or corruption in one sending window and, as a result, often has to time out. TCP SACK performs better than TCP reno because generally it can tolerate more packet losses in one sending window than TCP reno [8]. However, as BER keeps increasing, TCP SACK's goodput also rapidly goes down. The low goodput of TCP SACK when BER is high is inevitable. This is because, in TCP SACK, a tiny TCP/IP header is always coupled with a large data payload, and as we explained in section 7, doing so unnecessarily increases the chance of wrongly triggering TCP's congestion control. In contrast, due to the decoupling of a tiny TCP/IP header from a large data payload, we see that TCP decoupling outperforms TCP reno and TCP SACK on all BERs, and the performance improvement is about the what our analysis predicts – 350%.

There is a gap between the curve of the theoretical upper bound and that of TCP decoupling. This is due to some unnecessary retransmissions in the current TCP decoupling scheme, as discussed in the end of section 6. When BER is very small and near $1 \cdot 10^{-7}$, TCP reno's goodput is slightly

[1] As of December 1, 1999, after having contacted many development engineers in industry, the authors are still looking for methods that can disable the ARQ of an IEEE 802.11 WaveLAN card.

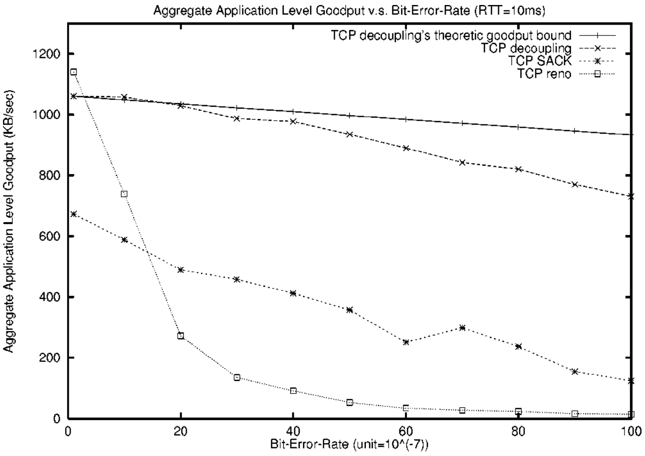


Figure 10. Performance improvements of TCP decoupling compared to TCP reno and TCP SACK for various values of BER. RTT = 10 ms.

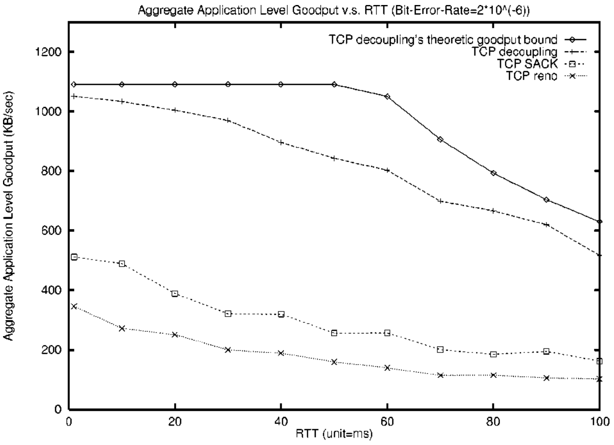


Figure 11. Performance improvements of TCP decoupling compared to TCP reno and TCP SACK for various values of RTT. BER is $2 \cdot 10^{-6}$.

higher than that of TCP decoupling. The difference is due to the approximately 3% header packet overhead in this particular implementation of TCP decoupling.

8.3. Reliable decoupling socket experiments suite 2

This set of experiments is similar to that of section 8.2, but with varying RTTs. A fixed value of $\text{BER} = 2 \cdot 10^{-6}$ is used in these experiments. Figure 11 shows that TCP decoupling always outperforms TCP reno and TCP SACK on all RTTs.

The top curve in figure 11 gives a theoretical upper bound on the goodput that any TCP scheme can possibly achieve for various values of RTT. The declining trend of the upper bound as RTT increases, depicted in figure 11, is an inevitable consequence of $\text{BER} > 0$. Equations (1) and (2) show that W is a function of BER and PS. (In fact, W is inversely proportional to the square root of $\text{BER} \cdot \text{PS}$.) For the experimental suite 2, since BER and PS are fixed, so is W. Equation (3) shows that for a fixed W, MAT must decrease linearly as RTT increases. Figure 11 shows that the achieved goodput of the TCP decoupling scheme approaches the upper bound, although it does not match due to some retransmission redundancy.

Figure 11 shows that the goodput of TCP decoupling at $\text{RTT} = 100$ ms is approximately 560 KB/s. This goodput is close to the best possible performance under the TCP decoupling approach. With $\text{BER} = 2 \cdot 10^{-6}$ and $\text{PS} = 52$ bytes for header packets, equation (1) implies $\text{PER} = 0.0008$. For this value of PER, solving equation (2) for W gives $W = 56$. With $W = 56$, $\text{PS} = 1500$ bytes for data packets, and $\text{RTT} = 100$ ms, equation (3) gives $\text{MAT} = 652{,}500$ Bytes/s. After accounting for the packet error rate of $\text{BER} \cdot 1500 \cdot 8 = 0.024$ for data packets, and the overhead of the 52-byte TCP/IP header associated with each 1500-byte data packet, a theoretical upper bound on the goodput of approximately 615 KB/s is obtained. The achieved goodput of 560 KB/s is 9% lower than the upper bound.

9. Advantages over other approaches

The reliable decoupling socket approach is an end-to-end approach because it does not need any special support from the wireless network. Unlike many other schemes presented in section 2, it does not require a special TCP-aware agent to run on the base station to snoop passing TCP packets; it does not need to split a TCP connection into two connections at the base station, and it does not require the uses of FEC and ARQ on wireless links. Because of this end-to-end property, the reliable decoupling socket approach can be quickly deployed in any kind of wireless network to realize the $\sqrt{\text{MTU/HP\_Sz}}$ performance improvement. On the contrary, the other schemes mentioned in section 2 have special requirements for the wireless network. For example, snooping and splitting schemes are not suitable to a multi-hop all-wireless network because it is impractical to snoop the traffic of a TCP connection or to split a TCP connection multiple times on routers along the TCP connection's path. The advantages offered by using FEC and ARQ cannot be realized unless they are employed on wireless links.

10. Future improvements

Equation (2) shows that if we can reduce the PER for those packets whose droppings will trigger TCP congestion control, a larger W can be resulted and, as a result, a higher TCP goodput can be achieved. The TCP decoupling approach achieves this goal by using tiny TCP/IP header packets to implement TCP congestion control so that the chance of mistakenly treating a packet dropping due to link corruptions as one due to network congestion can be reduced. Currently, the size of a header packet of 52 bytes has reached the minimum for a packet to be a TCP/IP packet and also carry the useful TCP timestamp option, which allows for a more accurate estimate of a TCP connection's RTT.

Using the TCP header compression algorithm proposed in [22] and the *twice* algorithm proposed in [12] on wireless links, one can greatly reduce the size of header packets (and thus, their PERs) without the bad effects on TCP's performance caused by dropping a header-compressed packet [12]. The TCP header compression mechanism can compress the TCP/IP header of a header packet from 40 bytes down to only 3 bytes, resulting in a $3 + 12$ (TCP timestamp option) $= 15$ byte packet. (Note that the TCP header compression algorithm does not attempt to compress TCP options. However, the same method can be used to also compress the TCP timestamp option and result in a packet size even smaller than 15 bytes.)

Twice algorithm works with the TCP header decompresser at the receiving end of a wireless link. If the decompresser detects state inconsistency (by noticing the wrong computed TCP checksum) when decompressing a header-compressed packet, *twice* first assumes that a packet has been dropped and makes a guess of the content of the dropped packet's TCP/IP header based on the past history of TCP header contents. It then advances its decompression state as if the lost packet had been correctly received and decompressed, and then decompresses the newly arrived header-compressed packet again. If the computed TCP checksum is correct, the guess that one packet is dropped is correct and every thing is back to the consistent state. Otherwise, *twice* assumes that two packets are lost and the above procedure repeats.

It is worth noting that TCP header compression and *twice* are particularly well suited to the TCP decoupling approach. First, there is no data payload coupled with a header packet. Second, the difference between consecutive header packets is only in the sending sequence number field and the difference is always VMSS. These two properties enable the TCP header compression to always compress a 40-byte header into a 3-byte header and allow *twice* to easily make a correct guess about the TCP header content of a missing (corrupted) header packet. For *twice*, because the PER of the tiny header packets is further reduced by TCP header compression, the probability that more than one consecutive header packets are dropped will be further significantly reduced. As a result, *twice* would easily succeed in its first guess almost every time.

Another dimension of improvement is to apply FEC and/or ARQ to only header packets to protect them from corruption so that the effective PER of header packets is reduced. Since the size of header packets is small, the added redundancy overhead caused by applying FEC (or ARQ) to only these header packets is also tiny compared to the added redundancy overhead when FEC (or ARQ) is universally applied to both the header and data payload of a 1500-byte TCP/IP packet in the traditional approach.

One limitation with using TCP header compression and *twice* or using FEC and ARQ is that these approaches cannot be performed simply at the two end hosts of a TCP connection. Instead, they must be performed at the link layer (on a per hop basis) and, as a result, require support from the wireless networks.

Another dimension of improvement is to use a larger VMSS at the expense of possibly generating more bursty traffic in networks. (Note that in this case a traffic shaper can be used at the sender of a control TCP to reduce the burstiness.) It is clear that due to a nonzero PER of header packets, there must still be a limit on the bandwidth achieved by header packets. Since the achieved bandwidth of data packets is VMSS times that of header packets (discussed in section 4), if the achieved bandwidth of data packets does not reach the bandwidth of the wireless link, we can increase the VMSS to achieve a higher link utilization.

11. Conclusions

The TCP decoupling approach proposed in this paper is a new, general, and powerful approach. It applies TCP's congestion control alone to a packet stream for which TCP's error control is not desired or should be performed separately from TCP's congestion control. The TCP decoupling approach has several important applications. This paper presents the application of the TCP decoupling approach in improving TCP's performance over lossy wireless networks.

The reliable decoupling socket approach, which is a direct application of the general TCP decoupling approach, improves a TCP connection's goodput over a lossy wireless network without any support from the wireless network. It improves TCP performance by using tiny TCP/IP header packets to implement TCP congestion control for a stream of large data packets. Because the large data payload is decoupled from the tiny TCP/IP header, the chance that a corrupted tiny header packet will wrongly trigger TCP's congestion control on the stream of large data packets is greatly reduced. The resulting performance improvement can be analytically shown to be proportional to $\sqrt{\mathrm{MTU/HP\_Sz}}$, where MTU is the maximum transmission unit of the wireless link and HP_Sz is the size of a packet containing only a TCP/IP header. For example, on a WaveLAN wireless network, where MTU is 1500 bytes and HP_Sz is 40 bytes, the achieved goodput improvement is about 350%. Experimental results comparing TCP decoupling, TCP reno and TCP SACK's performance in various wireless network conditions confirm the analysis.

Acknowledgements

This research was partially supported by Nortel, Sprint, Air Force Office of Scientific Research Multidisciplinary University Research Initiative Grant F49620-97-1-0382, and National Science Foundation Grant CDA-94-01024.

References

[1] E. Ayanoglu, S. Paul, T.F. Laportaa, K.K. Sabani and R.D. Gitlin, AIRMAIL: A link-layer protocol for wireless networks, Wireless Networks 1 (February 1995) 47–60.

[2] M. Allman, D. Glover and L. Sanchez, Enhancing TCP over satellite channels using standard mechanisms, RFC 2488 (January 1999).

[3] D.O. Awduche, J. Malcolm, J. Agogbua, M. O'Dell and J. McManus, Requirements for traffic engineering over MPLS, Internet draft (June 1999) (work in progress).

[4] A.V. Bakre and B.R. Badrinath, Implementation and performance evaluation of indirect TCP, IEEE Transactions on Computers 64(3) (1997) 260–278.

[5] H. Balakrishnan and R.H. Katz, Explicit loss notification and wireless web performance, in: *IEEE Globecom Internet Mini-Conference*, Sydney, Australia (November 1998).

[6] H. Balakrishnan, S. Seshan, E. Amir and R.H. Katz, Improving TCP/IP performance over wireless networks, in: *ACM MOBICOM'95*.

[7] J. Bennett, C. Partridge and N. Shectman, Packet reordering is not pathological network behavior: And it never will be again, IEEE/ACM Transactions on Networking (December 1999) (to appear).

[8] R. Bruyeron, B. Hemon and L. Zhang, Experimentations with TCP selective acknowledgment, ACM Computer Communication Review 28(2) (April 1998).

[9] R. Callon, N. Feldman, A. Fredette, G. Swallow and A. Viswanathan, A framework for multiprotocol label switching, Internet draft (June 1999) (work in progress).

[10] D.M. Chiu and R. Jain, Analysis of the increase and decrease algorihms for congestion avoidance in computer networks, Computer Networks and ISDN Systems 17 (1989) 1–14.

[11] B.P. Crow, I. Widjaja, J.G. Kim and P.T. Sakai, IEEE 802.11, Wireless local area networks, IEEE Communications Magazine 35(9) (September 1997).

[12] M. Degermark, M. Engan, B. Nordgren and S. Pink, Low loss TCP/IP header compression for wireless networks, in: *ACM MOBICOM'96*.

[13] D. Duchamp and N.F. Reynolds, Measured performance of a wireless LAN, in: *17th Conference on Local Computer Networks, IEEE* (1992) pp. 494–499.

[14] D. Eckhardt and P. Steenkiste, Improving wireless LAN performance via adaptive local error control, in: *Sixth IEEE International Conference on Network Protocols (ICNP'98)*, Austin (October 1998).

[15] K. Fall and S. Floyd, Simulation-based comparisons of Tahoe, Reno, and SACK TCP, ACM Computer Communication Review 26(3) (1996) 5–21.

[16] S. Floyd, Connections with multiple congested gateways in packet-switched networks, Part 1: One-way traffic, Computer Communications Review 21(5) (1991).

[17] S. Floyd and V. Jacobson, Random early detection gateways for congestion avoidance, IEEE/ACM Transactions on Networking 1(4) (1993) 397–413.

[18] FreeBSD web site.

[19] G. Held, *Frame Relay Networking* (Wiley, 1999).

[20] C. Huitema, *Routing in the Internet* (Prentice Hall, Englewood Cliffs, NJ, 1995).

[21] V. Jacobson, Congestion avoidance and control, in: *ACM SIGCOMM'88* (1988) pp. 314–329.

[22] V. Jacobson, Compressing TCP/IP headers for low-speed serial links, RFC 1144.

[23] H.T. Kung and S.Y. Wang, TCP trunking: Design, implementation, and performance, in: *IEEE ICNP'99* (November 1999).

[24] D. Lin and H.T. Kung, TCP fast recovery strategies: Analysis and improvements, in: *IEEE INFOCOM'98* (March 1998) pp. 263–271.

[25] M. Mathis, J. Mahdavi, S. Floyd and A. Romanow, TCP selective acknowledgment options, RFC 2018 (October 1996).

[26] D.E. McDysan and D.L. Spohn, *ATM: Theory and Application* (McGraw-Hill, New York, 1995).

[27] R. Perlman, *Interconncetions: Bridges and Routers* (Addison-Wesley, 1992).

[28] J. Postel, Transmission Control Protocol, RFC 793 (September 1981).

[29] N. Samaraweera and G. Fairhurst, Reinforcement of TCP/IP error recovery for wireless communications, ACM Computer Communications Review 28(2) (1998).

[30] J. Touch, S. Ostermann, D. Glover, M. Allman, J. Heidemann, S. Dawkins, J. Semke, K. Scott, J. Griner, D. Tran, T. Henderson and H. Kruse, Ongoing TCP research related to satellites, Internet draft (June 1999).

[31] S.Y. Wang, Decoupling control from data for TCP congestion control, Ph.D. Thesis, Harvard University (September 1999), `http://www.eecs.harvard.edu/networking/decoupling.html`

[32] WaveLAN web site, `http://www.wavelan.com`

[33] G.R. Wright and W.R. Stevens, *TCP/IP Illustrated*, Vol. 2, *The Implementation* (Addison-Wesley, 1995).

S.Y. Wang is currently an Assistant Professor at the Department of Computer Science and Information Engineering, National Chiao Tung University, Hsinchu, Taiwan. He received his Ph.D. degree in computer science from Harvard University in 1999. He is the author of the Harvard TCP/IP network simulator, which is being used by researchers worldwide at more than 200 industrial laboratories, universities, and Internet service providers. His research interests include computer networks, operating systems, and Internet technologies.
E-mail: shieyuan@csie.nctu.edu.tw

H.T. Kung is William H. Gates Professor of Computer Science and Electrical Engineering at Harvard. He received his Ph.D. from Carnegie Mellon and served on their faculty before joining Harvard in 1992. Over the last 25 years, he has pursued a variety of interests: algorithms and complexity, database systems, VLSI architectures, parallel computing, mobile computing, and computer networks.
E-mail: kung@harvard.edu