Wireless Networks 12, 301–319, 2006
© 2006 Springer Science + Business Media, LLC. Manufactured in The Netherlands.
DOI: 10.1007/s11276-005-5276-9

# On Striping Traffic over Multiple IEEE 802.11(b) Wireless Channels

S.Y. WANG, C.C. HWANG and C.L. CHOU
*Department of Computer Science and Information Engineering, National Chiao Tung University, Hsinchu, Taiwan*

**Published online:** 30 December 2005

**Abstract.** Due to interference, path loss, multipath fading, background noise, and many other factors, wireless communication normally cannot provide a wireless link with both a high data rate and a long transmission range. To address this problem, striping network traffic in parallel over multiple lower-data-rate but longer-transmission-range wireless channels may be used. In this paper, we propose a new striping method and evaluate its performances over multiple IEEE 802.11(b) channels under various conditions. Our extensive simulation results show that this method is quite effective for such an application.

**Keywords:** striping network traffic, parallel transmission

## 1. Introduction

Providing a long-range wireless link with a high data rate is difficult. Radio waves arrive at a wireless receiver from different directions with different time delays. They combine via vector addition at the receiver antenna to form a resultant signal. The amplitude of the resultant signal may be large or small depending on whether the incoming waves combine to reinforce each other or cancel each other. The phase relationship between the various incoming waves also changes. Substantial amplitude and phase fluctuations may thus occur and the signal is subjected to fading. When relative motion exists between the wireless transmitter and receiver, there is a Doppler shift in the received signal. All of these factors can cause inter-symbol interference and thus decoding errors at the receiver.

Increasing the transmission time of each bit can reduce the inter-symbol interference effect. However, this means that the data rate of a wireless link must be lowered. As such, given the same transmit power, a wireless link normally is either high-rate-but-short-range or long-range-but-low-rate. Achieving both a high data rate and a long transmission range at the same time normally is impossible. As an example, in [12], the Lucent Technology Inc's IEEE 802.11(b) [4] radio characteristics table shows that the effective transmission range for the 1, 2, 5.5, and 11 Mbps data rates are decreasing. They are 540, 400, 195, and 120 meters in the open space environment, respectively.

To overcome this dilemma, using multiple long-range-but-low-rate wireless channels and sending network traffic over them in parallel may be used. This operation is called "striping" and the used channels are called "stripes" in the literature [2]. Using striping, potentially a long-range-and-high-rate wireless trunk can be created by aggregating the bandwidth of multiple long-range-but-low-rate channels.

In addition to providing higher aggregated bandwidth, striping also provides fault tolerance for interfaces and channels. First, since multiple wireless interfaces are used at the transmitter and receiver, if one of them fails, others can still be used to continuously send and receive network traffic. Second, since multiple wireless channels are used between the transmitter and receiver, if the quality of one of these channels becomes very bad due to severe multipath fading, others can still be used to continuously convey network traffic.

Using striping, the formed wireless trunk can be directly used between two terminals (can be fixed or mobile) to exchange their packets. It can also be used between a mobile terminal and a fixed base station. In addition, it can also be used between two fixed wireless bridges to exchange packets between two buildings. Wireless bridges (e.g., the CISCO Aironet 350 series wireless bridge product [3]) are primarily used to provide high-speed long-range outdoor links between buildings. Striping is very suitable for such an application.

Although striping can provide multiple advantages, several issues remain. First it should be implemented transparently at a layer so that all upper layers will not notice its existence. Second, after packets are striped over multiple channels, their order should be maintained before they are delivered to the upper layer at the receiver. However, on wireless channels, packets can easily get corrupted, dropped, and/or retransmitted. Without implementing a packet resequencing mechanism at the receiver, excessive packet reorderings can occur. Third, the extra delays introduced in striping for maintaining packet delivery order need to be kept small. Fourth, a dynamic fault tolerance mechanism should be implemented so that the available bandwidth of all wireless channels can be efficiently aggregated and utilized.

In this paper, we design and implement a new striping method for striping traffic over multiple IEEE 802.11(b) channels. The issues discussed above are addressed in this method. We evaluated the performances of this method under various network conditions, including ideal channel, congestion, bit errors, hidden-terminal, and mobility. The performance metrics used are the achieved aggregated throughput and the extra delays introduced for maintaining packet delivery order. Simulation results show that this method can generate satisfactory performances under these conditions.

The rest of the paper is organized as follows. In Section 2, we survey related work and present the design of Surplus Round Robin (SRR) striping method in details because later we will compare the performances of our striping method to its performances. In Section 3, we present the design and implementation of our striping method. In Section 4, we evaluate and present the performances of our striping method under various conditions. The performances of SRR and our striping method are also compared. In Section 7, we discuss our future work. Finally, in Section 8, we conclude the paper.

## 2. Related work

Striping has been proposed and studied in the literature. However, most designs assumed physical transport links with constant bit rates and stable link characteristics such as those found in circuit-switched networks . However, this is not the case with wireless channels. Due to congestion, interference, bit errors, multipath fading, and many other factors, the available bandwidth of a wireless channel and the packet transfer latency on a wireless channel can vary drastically over time.

Recently, the authors in [5,6,8] studied striping over wireless networks. However, their focuses are different from those of this paper and they did not study the performance of striping under various conditions. In [8], the author applied striping to four wide-area Cellular Digital Packet Data (CDPD) [13] channels and focused only on the achieved throughput under the typical condition. In [5,6], the authors focused only on the transmission delays caused by detecting and retransmitting lost packets over striping channels. They assumed that sequence numbers are not used in the packets sent over striping channels and the link layer does not automatically retransmit lost packets. Under these assumptions, they proposed a simple delay-time model and presented some analyses.

In contrast, this paper focuses on striping network traffic over multiple IEEE 802.11(b) channels. The medium access control (MAC) protocol of IEEE 802.11(b) is accurately simulated in our study. Using simulations, we extensively studied the performances of our method under various conditions. In addition, we also compared the performances of our method to those of SRR [1].

The striping method proposed in [1] is a reliable and scalable striping protocol. It is named Surplus Round Robin (SRR) and most of the research papers refer to it. In [1], it is shown that SRR can perform well on fixed networks. However, because its performances on IEEE 802.11(b) wireless channels have not been studied, we will compare its performances with those of our striping method on IEEE 802.11(b) wireless channels.

For this reason, in the following we will present SRR in more details. With this background, later we can explain why SRR performs poorly on IEEE 802.11(b) wireless channels when presenting simulation results.

### *2.1. Surplus round robin*

The SRR method is designed for general purpose. Three key ideas of SRR are load sharing, logic reception, and marker packets. In the sender side, SRR uses a load sharing algorithm to stripe packets over multiple links. Logic reception means that the receiver buffers incoming packets and performs the inverse algorithm to predict which stripe to receive packets from. Marker packets are used in SRR to perform synchronization recovery at the receiver.

The load sharing algorithm uses a basic round-robin scheme to switch among stripes for transmission. Starting from the first stripe, after sending some packets on a stripe, the sender switches to the next stripe. This operation is repeated until the sender rolls back to the first stripe. At that time, a round is completed.

The load sharing algorithm allows each stripe to share the traffic load equally. In SRR, a counter is kept for each stripe. The counter represents how many bytes can still be sent on the stripe. Whenever a packet is sent on a stripe, the size of the packet is subtracted from the stripe's counter. When the counter becomes negative, the sender cannot send more packets on the current stripe. As such, it switches to the next stripe. After a round, a stripe will again receive its turn to send packets. At that time, a fresh quantum value will be added to the stripe's counter. This value represents how many bytes can be sent on a stripe in a round. Figure 1 illustrates this operation.

At the receiver, packets coming from each stripe are buffered in a per-stripe queue. The receiver simulates the inverse version of the striping algorithm used in the sender to decide which stripe queue to receive packets from. Like the mechanism used in the sender, a counter is kept for each stripe. Whenever a packet is received from a stripe queue, the size of the packet is subtracted from that stripe's counter. When the counter becomes negative, it is refilled with the

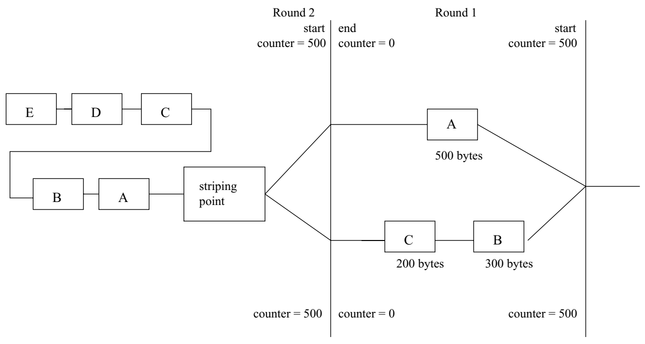

Figure 1. The operation of SRR at the sender.

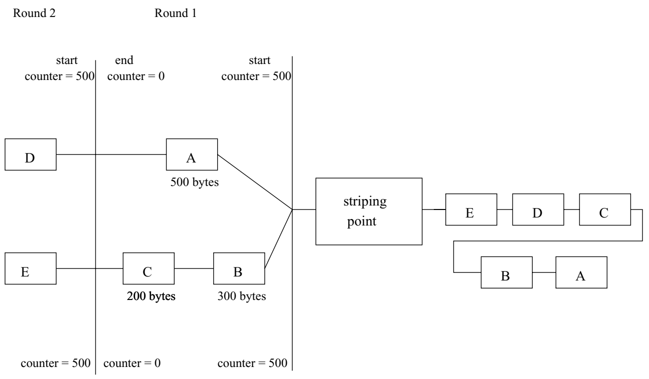

Figure 2. The operation of SRR at the receiver.

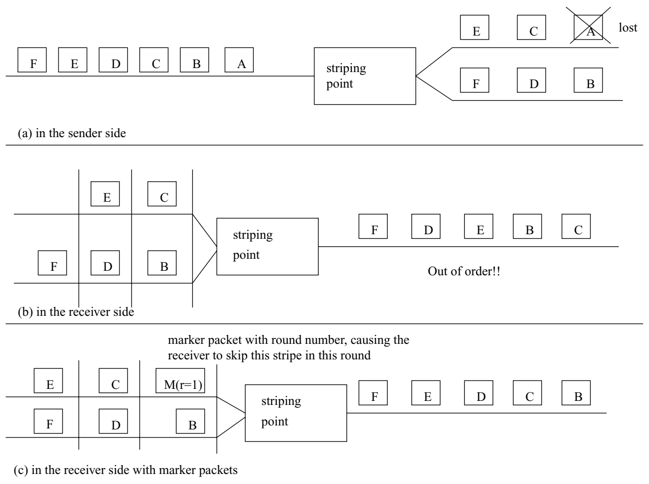

Figure 3. Marker packets are used in SRR for synchronizing the sender and receiver.

same quantum value used in the sender, and the receiver switches to the next stripe. By this design, the receiver can receive packets in their original order without using sequence numbers in these packets. Figure 2 illustrates this operation.

If packets are lost on the stripes, the reception of packets will become out of order. Marker packets are thus introduced to solve this problem. In SRR, the sender periodically sends a special packet named "marker packet" on each stripe. A marker packet is different from normal data packets. A marker packet contains the information of the current round number. When the receiver receives such a packet, the receiver can synchronize its current round number with the sender. With these marker packets, the receiver can recover from out-of-order packets. Figure 3 shows this design. In figure 3(a), the packet labeled 'A' is lost. In figure 3(b), we can see that the reception in the receiver now becomes out of order. If the marker packet is sent, as shown in figure 3(c), the lost packet can be detected via the round number carried in the marker packet and packet reordering can be avoided.

SRR can guarantee FIFO packet delivery order only when no packet is lost. When a data packet is lost, all data packets following it will be received and delivered in wrong order to the upper layer until a marker packet arrives. When a data packet and a marker packet are both lost, this out-of-order problem will become worse because more data packets will be received and delivered in wrong order. It is clear that marker packets are more important than data packets. However, on wireless channels, data and marker packets are subjected to bit errors equally.

As mentioned earlier, striping should be transparent to the upper layer. The guarantee of FIFO delivery order is an important issue. Even though SRR can perform order recovery via marker packets, the synchronization can only be re-established after future marker packets are received. Before that time, some out-of-order packets will have been handed to the upper layer. The re-establishment of synchronization will be too late for these packets.

Synchronization can be re-established very soon by sending marker packets frequently. However, since marker packets are bandwidth overhead, a large fraction of bandwidth will be wasted by these marker packets.

In addition, SRR has another problem. Since the receiver simulates the algorithm used at the sender, it expects to receive some packets from the current stripe. Until the counter of the current stripe becomes negative, it will not switch to the next stripe. This design will cause blocking if no packet is expected to arrive on the current stripe.

For example, if (1) the link of the current stripe is broken and thus no new packet will arrive soon or (2) a packet is lost on the current stripe but no new packet is expected to arrive soon due to TCP's windowing congestion control, the receiving operation will block forever on the current stripe until some new packets arrive. In the first case, the operation resuming time is when the link becomes non-broken while in the second case it is when the TCP sender timeouts and resends the lost packet. In either case, during this period the receiver cannot proceed to receive more packets from other stripes. We can see that in such a situation, the fault tolerance capability inherent in striping is lost. In addition, TCP will perform poorly due to frequent timeouts. In Section 4, the poor simulation results of SRR confirm this reasoning.

## 3. The proposed method

Because TCP traffic is the most dominant traffic in the current Internet, the proposed method should work well with TCP. In the first sub-section, we first discuss the FIFO delivery effect on TCP performance. After reading the first sub-section, readers will understand that maintaining FIFO delivery order, reducing packet losses, and reducing packet resequencing delays are all very important to achieve good TCP performance over a wireless trunk.

### *3.1. FIFO effect on TCP performance*

TCP performs poorly when encountering many out-of-order packets. In this sub-section, we briefly describe TCP congestion control protocol. Readers can refer to [9] for more details.

For providing a reliable service, the TCP receiver constantly sends back acknowledgment packets to the TCP sender to acknowledge the receipt of data packets. When receiving an out-of-order packet, the TCP receiver will immediately generate an acknowledgment packet (called "a duplicate ACK") and send it back to the TCP sender. The purpose of this duplicate ACK is to let the TCP sender know that a packet was received out of order.

When three or more duplicate ACKs are received, it is a strong signal to the TCP sender that a packet was lost. The TCP sender then immediately retransmits the lost packet without waiting for its retransmission timer to expire. Since the lower bound of this retransmission timer is 1 second, this quick response is called the fast retransmit and recovery algorithm.

In addition to retransmitting the lost packet, the TCP sender reduces its current sending rate by a half to mitigate the current congestion level. This is done by cutting its congestion window size by a half.

From the above description, we see that a packet loss will cause all of its following packets to become out-of-order. This will trigger TCP congestion control and lower the TCP sender's throughput by a half. Even if there is no packet loss, if the order of received packets is so wrong that more than three duplicate ACKs are generated, the TCP sender's throughput will also be reduced by a half.

These explanations show the importance of FIFO delivery order to good TCP performance. When striping is used, maintaining packet FIFO delivery order after they are sent over multiple stripes thus is also important. However, we note that strictly maintaining FIFO delivery order should not cause too many packets to be dropped or be delayed by a long time. Otherwise, TCP retransmit timer will be constantly triggered causing severe sending rate reductions.

## *3.2. System architecture*

We implement the striping method at the device driver layer of the FreeBSD 4.8 operating system. The architecture diagram is shown in figure 4. To use striping transparently, we use the tunnel network interface provided in FreeBSD 4.8 to represent the wireless trunk interface. A tunnel interface is a pseudo network interface without any physical network equipment attached to it. However, from the kernel's point of view, this pseudo network interface's functionality is exactly the same as that of a normal network interface (e.g., an Ethernet interface). That is, the kernel can send and receive packets through a tunnel interface and can use it as the outgoing interface for some routing entries.

In the sending direction, to stripe network traffic over the wireless trunk, a routing entry that specifies the trunk interface as the outgoing interface is set up at the sender. From now on, the kernel will direct all trunking packets to the trunk interface for transmission. When the device driver of the trunk interface receives these packets, it will dispatch them to different IEEE 802.11(b) interfaces using a round-robin scheme. An IEEE 802.11(b) interface, when receiving such a packet from the trunk driver, will then transmit the packet over its assigned frequency channel.

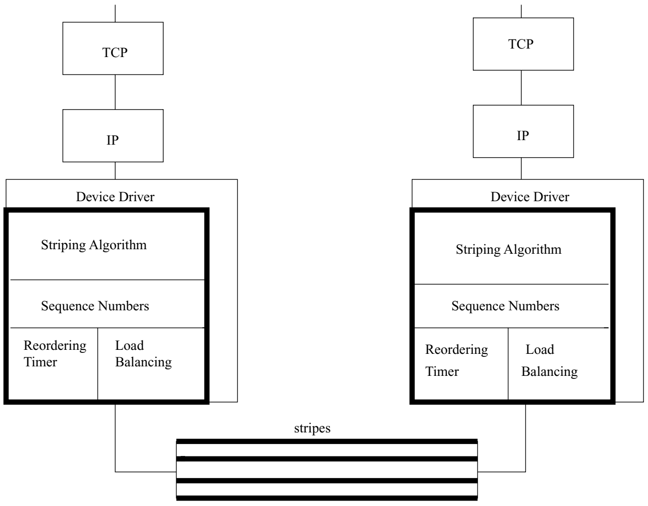

Figure 4. The architecture diagram of the proposed striping method.

In the receiving direction, when a packet arrives at the receiver, it will be received by an IEEE 802.11(b) interface. This interface will then deliver the packet to the trunk driver. In the trunk driver, packets that are received in order will be delivered to the upper layer immediately. Otherwise, out-of-order packets will be temporarily stored in a resequencing queue waiting to become in-order.

Figure 5 shows the protocol stacks used inside a wireless trunk interface. This example assumes that three IEEE 802.11(b) interfaces are used to form the wireless trunk

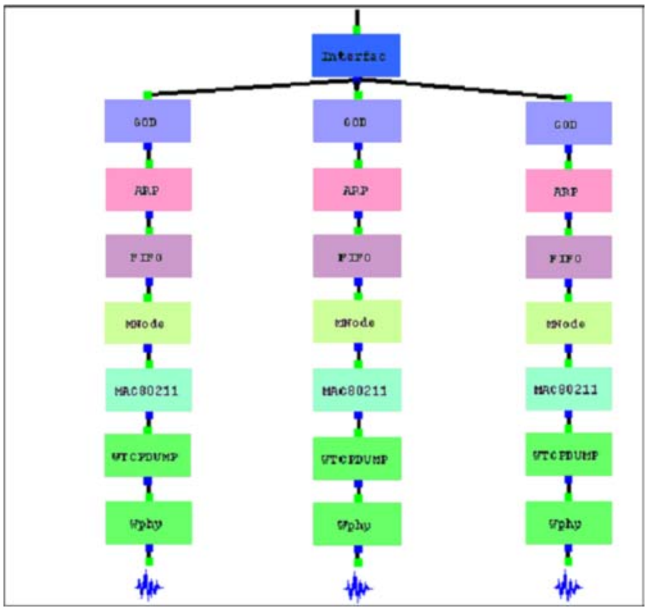

Figure 5. The internal protocol stacks used inside a wireless trunk interface.

interface. In our implementation, each IEEE 802.11(b) interface uses a protocol stack consisting of GOD, ARP, FIFO, MNODE, MAC802.11, WTCPDUMP, and WPHY modules. These modules are for routing, address resolution, FIFO queue, mobility management, IEEE 802.11 MAC protocol, tcpdump packet capture, and wireless physical channel purposes, respectively. To avoid interference, these wireless interfaces are configured to use different frequency channels.

### 3.3. Sequence numbers

Because packets are striped over multiple channels and may arrive at the receiver out of order, our striping method uses network-layer sequence numbers to enable the receiver to know their original order. With this information, the receiver can re-sequence out-of-order packets before delivering them to the upper layer.

At the sender, there is a network-layer counter maintained in the wireless trunk driver. When a packet is to be sent over the wireless trunk, the value of this counter is put into the header of this packet serving as the packet's sequence number. The value of this counter is then incremented by one for the next outgoing packet.

### 3.4. Round-robin scheme

At the sender, the wireless trunk driver uses a simple round-robin scheme to dispatch outgoing packets over multiple wireless channels. We configure each participating IEEE 802.11(b) wireless interface driver to let its output queue can hold at most one packet. When the trunk driver wants to dispatch a packet to a wireless interface and finds that the interface's output queue is empty (i.e., idle), the packet will be dispatched to that interface for immediate transmission. However, if the trunk driver finds that the output queue of that interface is full (i.e., busy transmitting), it will skip the current wireless interface and try the next one. This procedure repeats until one idle wireless interface is found. At that time, the packet is dispatched to that interface for transmission.

In case all wireless interfaces are busy and their output queues are all full, the trunk driver will store the outgoing packet in its own output queue and simply return. (The maximum length of this output queue is the default 50 packets.) It will be invoked again to send out more packets when a wireless interface becomes idle. This automatic invocation is achieved by the assistance of interrupt service routine (ISR). When a wireless interface finishes transmitting a packet, its ISR will be invoked by the interface to notify the kernel of this event. In our striping design, we let the ISR call the trunk driver's packet output function and tell it where to begin the round-robin search (i.e., starting from the interface that just became idle).

With this design, the wireless trunk driver can fully utilize the available bandwidth of all wireless channels and provide fault tolerance. For example, suppose that a wireless channel is experiencing congestion and its available bandwidth suddenly reduces, most packets will be directed and sent over other wireless channels. Setting the maximum length of the output queue of every wireless interface to only one packet also reduces the end-to-end packet delivery delay on the trunk. If the maximum queue length is set to a large number, say the default 50 packets used in FreeBSD, a packet may experience long queuing delay in the queue of an interface that is experiencing bad channel conditions.

### 3.5. Packet reordering mechanism

At the receiver, when an IEEE 802.11(b) interface receives a packet, it will hand the packet to the trunk driver immediately. The trunk driver will check whether the packet is received in order. If yes, it will hand the packet to the upper layer immediately. In case the packet is received out of order, the trunk driver will not hand it to the upper layer immediately. Instead, it will set up a reordering timer and store the packet in a resequencing queue. The packet will then wait there for the first "hole" in the sequence number space to be filled. Here, a "hole" means a gap in the received sequence numbers. A packet loss always causes a hole. However, a hole can also be caused when there is no packet loss but packet reorderings occur.

If all packets belonging to the first hole have arrived (i.e., the first hole has been filled) before the timer expires, the timer will be reset and the packets in the queue will be delivered to the upper layer one by one until the queue becomes empty or the next hole is encountered. On the other hand, if the timer expires before the first hole is fully filled, all packets in the queue will be flushed and delivered to the upper layer regardless whether there may be some other holes in the queue. In both cases, the variable N will be updated to the sequence number of the last packet delivered to the upper layer. Figure 6 shows an example diagram demonstrating the above packet holes processing.

When a packet arrives at the receiver, its sequence number $S$ is compared with $N + 1$. If $S = N + 1$, the trunk driver immediately delivers the packet to the upper layer because it is in-order. If $S > N+1$, the packet is placed into the resequencing queue because it is out-of-order. If $S = N$, the trunk driver drops this redundant packet because it has been successfully delivered to the upper layer. If $S < N$, the trunk driver still delivers this packet to the upper layer. Figure 7 shows an example diagram demonstrating the delivery/queuing/dropping processing of a packet that has just arrived.

The above design regarding the $S < N$ situation is to improve TCP throughput. In the following, we call a packet received in such a condition a "late" packet because it is too late to be delivered in order at the trunk layer. When $S < N$, the packet may be a redundant packet that has been delivered to the upper layer. In such a case, it should be dropped. However, because all packets in the resequencing queue are flushed to the upper layer when the reordering timer expires and at that time they may contain holes, a late packet actually

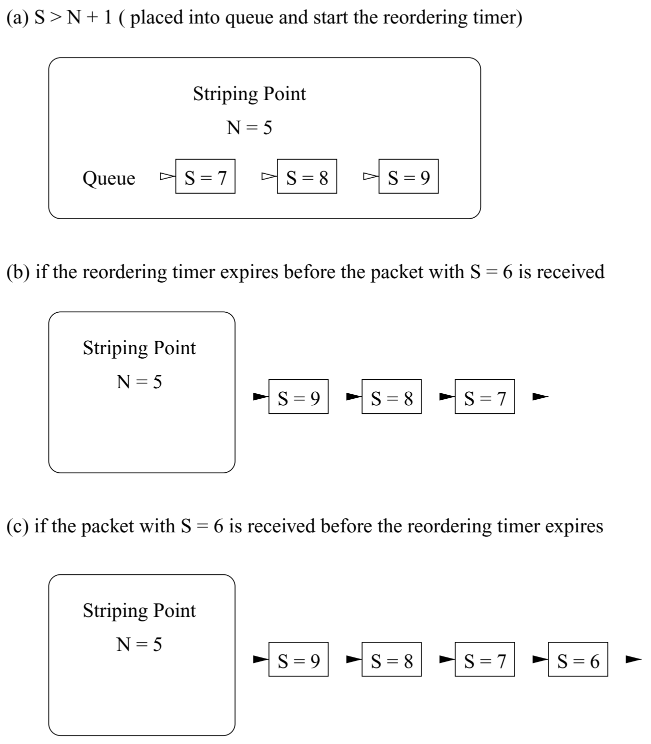
(a) S > N + 1 ( placed into queue and start the reordering timer)
Striping Point
N = 5
Queue
S = 7
S = 8
S = 9
(b) if the reordering timer expires before the packet with S = 6 is received
Striping Point
N = 5
S = 9
S = 8
S = 7
(c) if the packet with S = 6 is received before the reordering timer expires
Striping Point
N = 5
S = 9
S = 8
S = 7
S = 6

Figure 6. An example diagram demonstrating the packet holes processing.

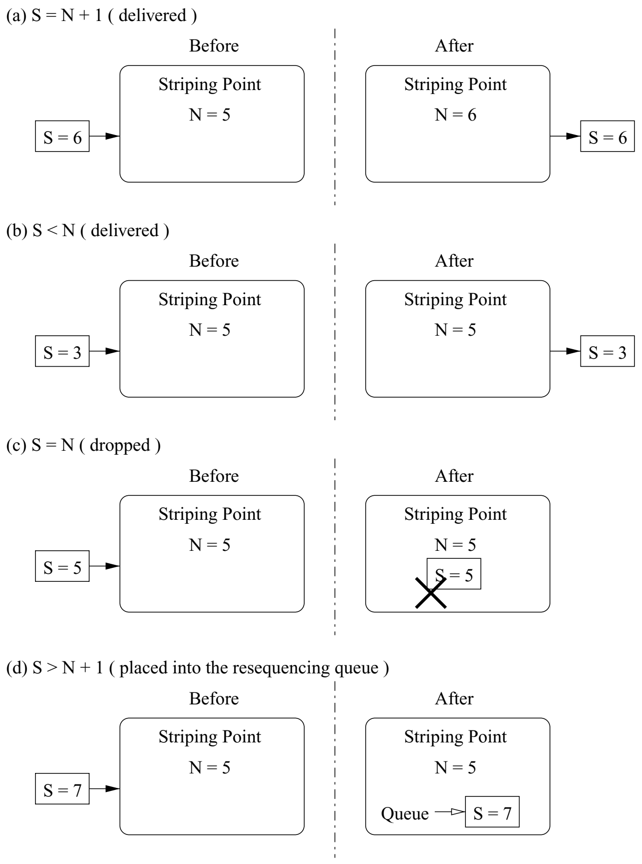

Figure 7. An example diagram demonstrating the delivery/queuing/dropping processing of a packet that has just arrived.

may be a new packet and able to fill a hole in the packets already flushed to the upper layer.

For TCP traffic, since the TCP receiver implements a reassembly queue at the TCP layer, a late packet may be un-late for the TCP reassembly queue. For example, if a late packet can fill the second hole in the TCP reassembly queue, even though it is late for the trunk layer it is still un-late for the TCP layer. Because (1) a TCP connection's throughput will be halved due to any packet loss, (2) redundant packets occur infrequently, and (3) the upper layer can discard redundant packets without any performance loss, we let the trunk driver still deliver late packets to the upper layer.

Note that the above design decision may result in non-FIFO delivery order to the upper layer. As explained before, this is due to the goal to improve TCP's achieved throughput. Maintaining a strict FIFO delivery order actually can be easily done by dropping all late packets. However, this may result in many holes in the packets delivered to the TCP layer. Since TCP can resequence out-of-order packets before delivering their data to the application program, maintaining a strict FIFO delivery order at the trunk layer at the cost of many packet losses is not worth it. It would only significantly lower TCP throughput without other advantages.

On the other hand, because UDP does not resequence out-of-order packets by itself and packet losses do not cause UDP to lower a UDP connection's throughput, a UDP-based application program may want to request a strict FIFO delivery order from the trunk layer. This property may be important for some real-time multimedia UDP-based application programs. In such a case, the trunk driver can be configured to drop all late packets.

### *3.6. Reordering timer*

From the above discussion, we see that when using striping to provide a transport service to the upper layer there is trade-off between (1) maintaining a strict FIFO delivery order, (2) reducing unnecessary late packet dropping, and (3) reducing unnecessary packet resequencing delay. It is clear that if the trunk driver insists on providing a strict FIFO delivery order, the unnecessary packet dropping number and packet resequencing delay will be high. On the other hand, if the trunk driver allows some degree of packet reordering, these problems can be mitigated.

Given this dilemma, the trunk driver tries to reach a balancing point where good TCP and UDP throughput can be achieved while packet resequencing delay can still be kept low. A solution is to allow a small degree of packet reordering to occur in the packet stream delivered to the upper layer. As long as no more than three duplicate ACKs are generated by the TCP receiver, the TCP sender's congestion control will not be triggered.

To achieve this goal, the reordering timer's timeout value should be carefully designed. If this value is set too high, although less packet reorderings will occur in the packets delivered to the upper layer, higher packet resequencing delay will result in the trunk driver. On the other hand, if this value is set too low, the situation will become reversed.

Clearly, the reordering timer's timeout value should be set to an appropriate value to achieve good performances. In addition, it should be set to different values under different network conditions. In IEEE 802.11(b), the MAC protocol uses acknowledgment frames to detect whether a transmitted frame has successfully reached the receiver. A frame may be retransmitted up to 4 or 7 times depending on whether it is a long or short frame [4]. With this MAC protocol, if the wireless channels are not busy and in good condition, the reordering timer can be set to a smaller value as retransmission of lost packets can be finished quickly. In contrast, if the channels are busy or in bad conditions, the timer should be set to a larger value to give retransmissions more time to finish.

Because wireless channel conditions change dynamically, there is no single timeout value that is suitable for all conditions. Clearly, it should be adjusted dynamically according to the current condition. That is, it should be adaptive. In our design, if the trunk driver receives a late packet (i.e., the $S < N$ condition), it increases the timeout value of the reordering timer by one millisecond. Figure 8 shows the achieved TCP throughput on a 3-channel wireless trunk under different values of this increment parameter. We can see that the system throughput is not sensitive to this parameter. Using 1, 2, or 3 milliseconds for this parameter yields very close results.

The reason for the above design is that the current timeout value is too small for the current network condition. Without increasing the current timeout value, many packets will not be retransmitted and arrive at the receiver in time. To let the design responsive, when the timeout value is increased, its new value is used by the current reordering timer immediately.

We need another policy to decrease the timeout value of the reordering timer. Otherwise, this value may go up to a high value without any chance to go down. In our design, we adopt the typical TCP saw-tooth window size adjusting pattern to dynamically adjust the timeout value. If the trunk driver does not receive any out-of-order packet for a period of time, it will reduce the current timeout value by a half. An adjusting timer is used to perform this work and its timeout value is set to one second. When the adjusting timer expires, the reordering timer's timeout value is halved.

How often to expire the adjusting timer should also be carefully designed. If it expires too frequently without good reasons, the reordering timer's timeout value will vary frequently when the network conditions are stable. This will result in sub-optimal performances. The ideal design should be that when the network conditions are stable, the timeout value can be maintained at a stable level matching the current network condition. In contrast, when the network conditions vary quickly, the timeout value can also vary quickly to match the current network conditions.

In our design, if a late packet is received, the adjusting timer is set up. If another late packet is received before the adjusting timer expires, the adjusting timer is reset. In this way if the trunk driver receives many late packets in a short period of time, the expiration time of the adjusting timer will continuously be postponed and its final value be set to the arrival time of the last late packet plus one second. Using this design, when the timeout value of the reordering timer is still growing and less than what it should be, the timeout value will not be unnecessarily halved by the expiration of the adjusting timer.

Figure 9 shows an example of the reordering timer's timeout value diagram. We can divide the diagram into several rounds each ending with a reduction in the timeout value. In each round we see that when late packets arrive, the timeout value of the reordering timer keeps increasing and the adjusting timer is rescheduled again and again. When the last setup of the adjusting timer in a round expires, which is one second after the arrival time of the last late packet in a round, the timeout value of the reordering timer is halved.

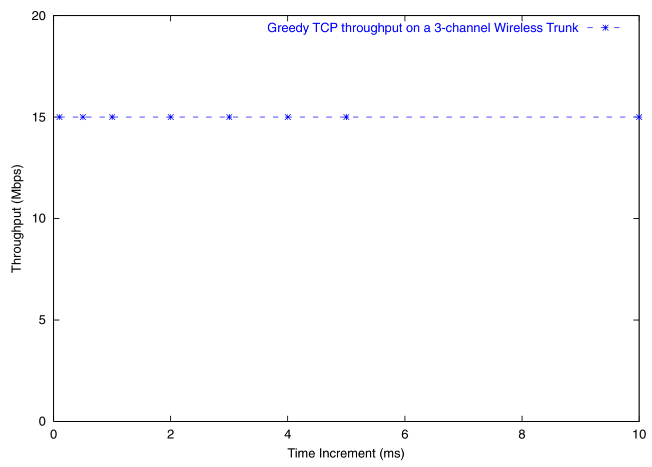

Figure 8. The achieved TCP throughput on a 3-channel wireless trunk under different values of the increment parameter.

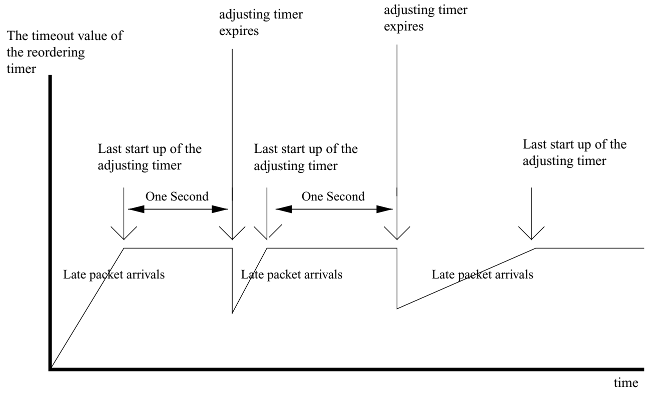
The timeout value of the reordering timer
adjusting timer expires
adjusting timer expires
Last start up of the adjusting timer
Last start up of the adjusting timer
Last start up of the adjusting timer
One Second
One Second
Late packet arrivals
Late packet arrivals
Late packet arrivals
time

Figure 9. The diagram showing the changes of the timeout value of the reordering timer.

## 4. Performance evaluations

In this section, we present the performances of our striping method under various conditions. Because most research papers refer to SRR, we also compare the performances of SRR to those of our method in some cases.

### 4.1. Simulation environment

We developed our striping code in the FreeBSD 4.8 operating system. With this code, we can perform real experiments using notebook computers equipped with IEEE 802.11(b) network interface cards (NIC). However, for several reasons, we decided to use the NCTUns 1.0 network simulator [11] to evaluate the performances of our striping method.

First, the NCTUns 1.0 is high-fidelity, extensible, and open-source. This makes it suitable for conducting networking research. More specifically, it can accurately simulate the 802.11(b) MAC-layer acknowledgment and automatic retransmission mechanism. Second, because it directly uses the network subsystem of the kernel of FreeBSD to generate accurate simulation results, it can directly use our in-kernel striping code to generate striping simulation results. Third, conducting real experiments using notebook computers each equipped with several IEEE 802.11(b) NICs is infeasible. Right now every notebook computer has only two PCMCIA slots. This limits the maximum number of IEEE 802.11(b) NICs that can simultaneously be installed and used by a notebook computer to only two. Furthermore, because the two PCMCIA slots are placed immediately next to each other and the antenna of an IEEE 802.11(b) NIC is much thicker than the body of the NIC, it is infeasible to insert two IEEE 802.11(b) NICs into the two slots at the same time.

For these reasons, we used the NCTUns 1.0 to conduct simulation study. Using the simulation approach, we can easily scale the number of IEEE 802.11(b) NICs used by the trunk sender and receiver without any problem. In addition, we can easily specify the BER of the wireless channels, the moving speed of the trunk sender and receiver, and other parameters. Using the simulation approach also avoids us to synchronize the clocks of the trunk sender and receiver machines, which is required to measure the end-to-end packet delivery delay over the wireless trunk.

### 4.2. Simulation results

The metrics used to evaluate the performances of our striping method include (1) achieved throughput and (2) packet delivery latency. When evaluating achieved throughput, we used both TCP and UDP protocols. However, because UDP is insensitive to packet reordering and losses, generally its achieved throughput is quite good regardless of the used striping method. Thus, in most cases, we report only achieved TCP throughput on a wireless trunk.

Figure 10 shows the basic simulation environment. Each node is equipped with three IEEE 802.11(b) NICs. The three NICs on node 1 use three different frequency channels (e.g., f1, f2, and f3). The three NICs on node 2 also use the same frequency channels (i.e., f1, f2, and f3) to form three separate wireless channels between node 1 and node 2. Node 2 is the sending node and node 1 is the receiving node. The distance between the sending node and the receiving node is 100 meters. During a simulation, unless stated otherwise, the sending node will greedily send its data to the receiving node. Every simulation lasts 250 seconds of simulated time.

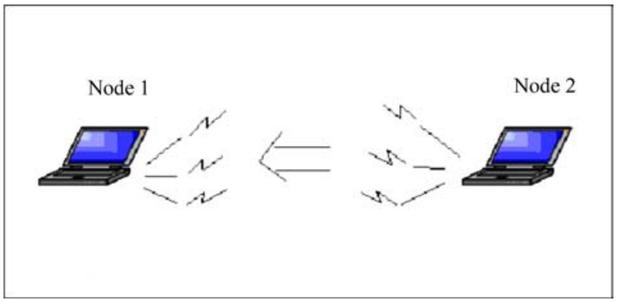

Figure 10. The basic simulation environment. Three wireless channels are set up between node 1 and node 2.

At first, we assume that the wireless channels are error-free (i.e., BER = 0) and there are no other active nodes around (i.e., no congestion). In this clean environment, we evaluate the performances of our striping method to show its capability in good channel conditions. We then gradually introduce bad conditions such as BER, congestion, mobility, hidden-terminal to show how it will perform under such conditions.

#### 4.2.1. Good channel condition

To show the benefit provided by striping, we first run a non-striping case in which each node is equipped with only one wireless NIC. It is assumed that in this case BER = 0 and there is no congestion. To establish a TCP connection and greedily send TCP packets, we used the stcp/rtcp traffic generator programs, which are freely available on the Internet. The stcp program is run on node 2 and the rtcp program is run on node 1. During the simulation, the stcp program greedily sends its data to the rtcp program and the rtcp program periodically measures and reports its received throughput. Figure 11 shows the achieved application-layer throughput reported by the rtcp program. (The average throughput is reported every 1 second.) We see that in good channel conditions the achieved TCP throughput on a single wireless channel is about 5 Mbps.

Next we use the environment depicted in figure 15 to evaluate our striping method. Again, we assume that BER = 0 and there is no congestion in this case. The stcp program is still run on node 2 and the rtcp program is still run on node 1 to generate greedy TCP traffic. Figure 12 shows the achieved application-layer throughput over a three-channel wireless trunk. As can be seen in this figure, the achieved throughput is about 15 Mbps. This result shows that the throughput gain provided by a three-channel wireless trunk using our striping method is almost three (15/5).

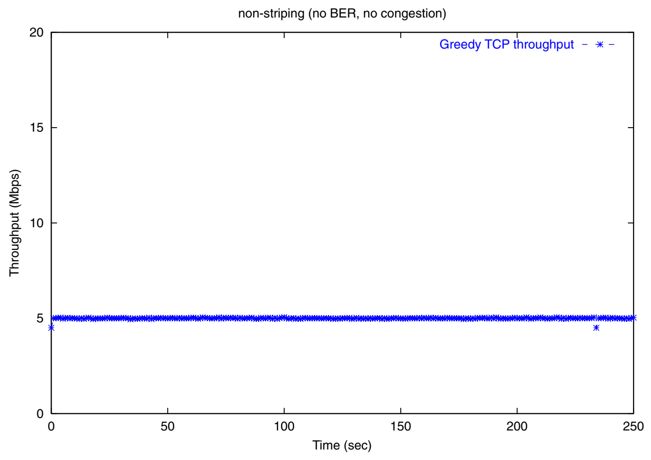

Figure 11. The achieved TCP throughput on a single IEEE 802.11(b) channel.

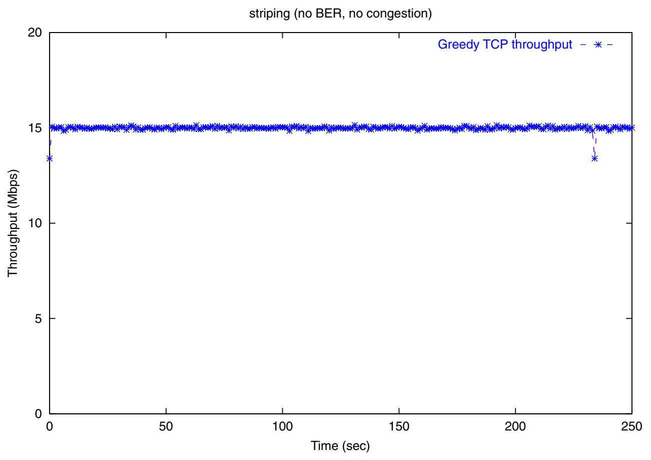

Figure 12. The achieved TCP throughput on a three-channel wireless trunk using our striping method.

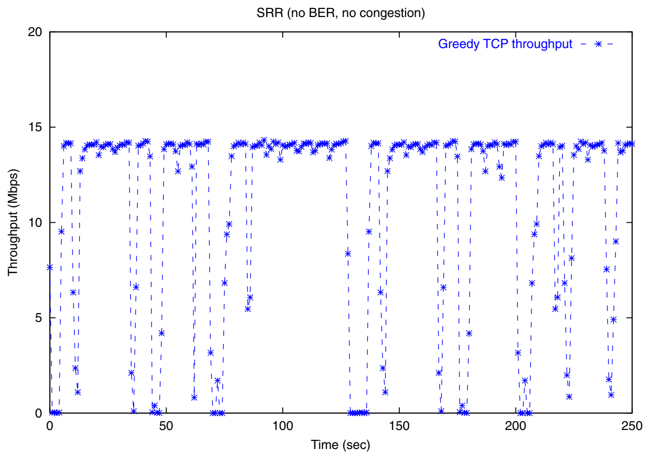

Figure 13. The achieved TCP throughput on a three-channel wireless trunk using SRR.

To make performance comparisons with SRR, we implemented SRR and evaluated its performance in the same environment. In our SRR implementation, a new quantum of 3,000 bytes is added to each striping channel when a round is over. When a channel's quantum is exhausted, a marker packet is sent on the channel.

Figure 13 shows the achieved throughput of SRR. We see that SRR does not perform as well as our striping method. First, the maximum achieved throughput is only 14 Mbps. Second, its throughput fluctuates greatly and sometimes drops to 0 Mbps.

For the lower throughput, it is due to the use of marker packets. As explained in Section 2.1, SRR needs to spend some bandwidth for its marker packets, which are bandwidth overhead solely used to synchronize the trunk sender and receiver. This explains why its maximum throughput cannot be as high as that provided by our striping method.

For the frequent throughput droppings, it is due to TCP sender's frequent timeouts. As explained in Section 2.1, the trunk receiver in SRR may get blocked when a TCP packet is lost on a striping channel but no new packet is expected to arrive soon due to TCP's windowing congestion control. This is exactly the case here. Note that although in this case the wireless channels are in good condition, TCP packets can still be dropped on these channels due to collisions between the trunk sender and receiver. This is because TCP traffic is two-way traffic. A TCP connection's data packets need to compete with its ACK packets on wireless channels, and some of them may thus be dropped due to collisions.

Using our striping method, we performed a simulation to evaluate the performance of UDP traffic. The environment is the same as that for testing TCP performance. The only exception is that the ttcp program to used to generate greedy UDP traffic. In figure 14, we see that the UDP throughput gain achieved on a three-channel wireless trunk using our striping method is also three. For SRR, because the greedy UDP traffic is not controlled by TCP windowing congestion

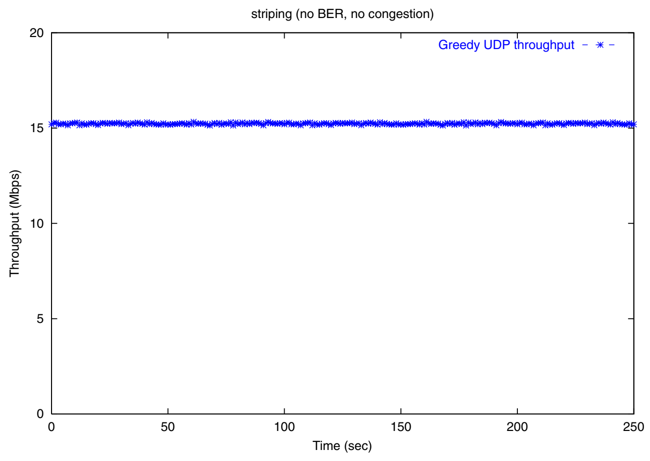

Figure 14. The achieved UDP throughput on a three-channel wireless trunk using our striping method.

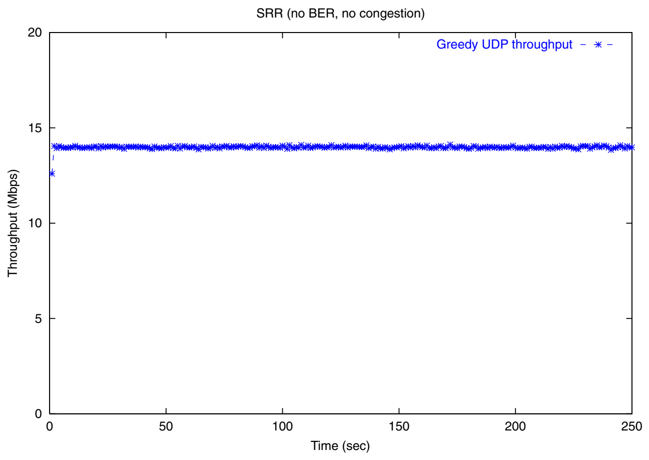

Figure 15. The achieved UDP throughput on a three-channel wireless trunk using SRR.

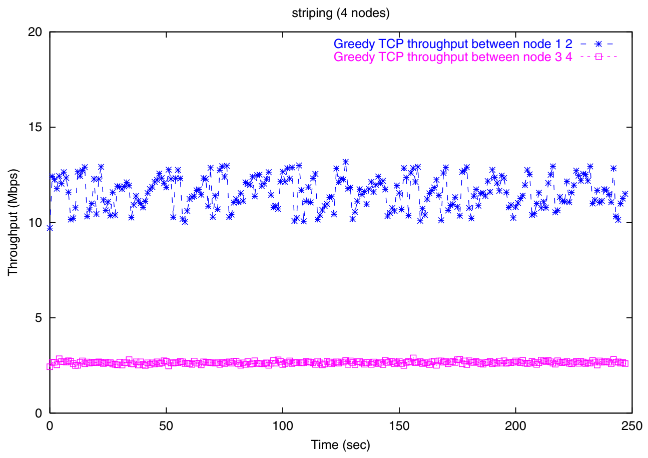

Figure 17. The achieved TCP throughput on a three-channel wireless trunk using our striping method, one channel is experiencing congestion.

control, the UDP sender does not timeout or reduce its sending rate even though some of its packets are dropped on the wireless channels. As such, its achieved throughput is also stable around 14 Mbps. Figure 15 shows the UDP throughput on a three-channel wireless trunk using SRR.

### 4.2.2. *Congestion condition*

Here we introduce congestion condition to the simulation environment. In figure 16, we introduce two new nodes, node 3 and node 4. They are placed in the transmission range of node 1 and node 2. The wireless NICs of node 3 and node 4 are configured to use the first channel of the wireless trunk, which is channel 1. Since these four nodes share the bandwidth of channel 1, in the ideal case, we can expect that node 3 and node 4 will receive one half of the achievable throughput on channel 1, which is about 2.5 Mbps (5/2). For the same reason, we can expect that node 1 and node 2 will receive about 2.5 Mbps throughput on channel 1.

Figure 17 shows the achieved TCP throughput of the wireless trunk and the achieved TCP throughput between node 3 and node 4. The achieved throughput between node 3 and node 4 is 2.5 Mbps, which is expected. The achieved throughput of the wireless trunk is about 12.5 Mbps, which is the ideal maximum achievable throughput and can be explained. Because of bandwidth sharing on channel 1, node 1 and node 2 are expected to receive 2.5 Mbps throughput on channel 1. On each of the other two channels, because there is no bandwidth sharing, these two nodes can receive the full available throughput of a single channel. For this reason, a total throughput of 12.5 Mbps (2.5 + 5 + 5) is the ideal maximum achievable throughput for the wireless trunk under this congestion condition.

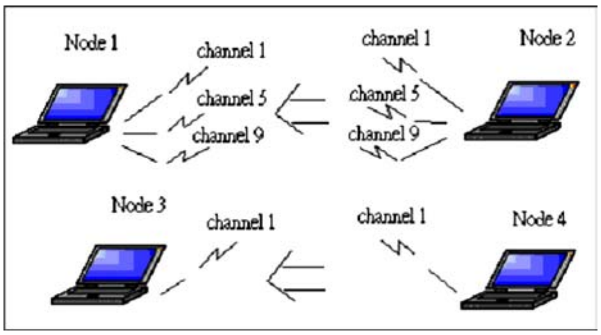

Figure 16. The simulation environment where one channel used by a three-channel wireless trunk is experiencing congestion.

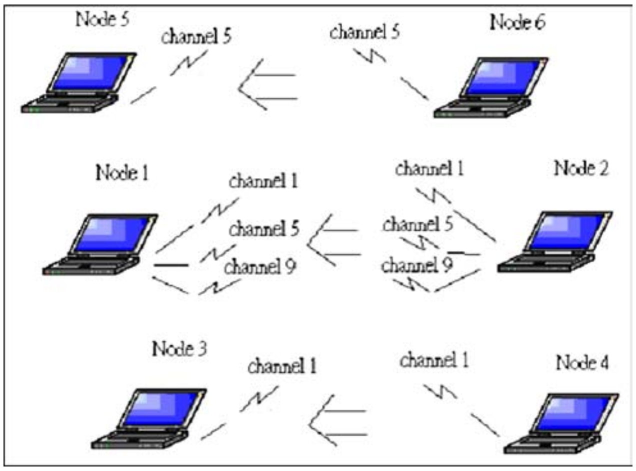

Figure 18. The simulation environment where two channels used by a three-channel wireless trunk are experiencing congestion.

In figure 18 we add two more nodes (node 5 and node 6) to increase the congestion level. The wireless NICs of these two nodes are configured to use channel 5, which is the second channel used by the wireless trunk. Figure 19 shows the results. As expected, node 3 and node 4 receive about 2.5 Mbps on channel 1, node 5 and node 6 receive about 2.5 Mbps on channel 5, and the wireless trunk receives a total of about 10 Mbps (2.5 + 2.5 + 5) over these three channels.

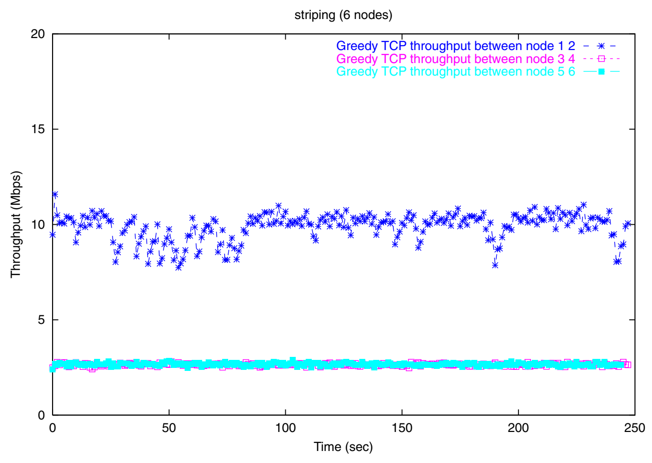

Figure 19. The achieved TCP throughput on a three-channel wireless trunk using our striping method, two channels are experiencing congestion.

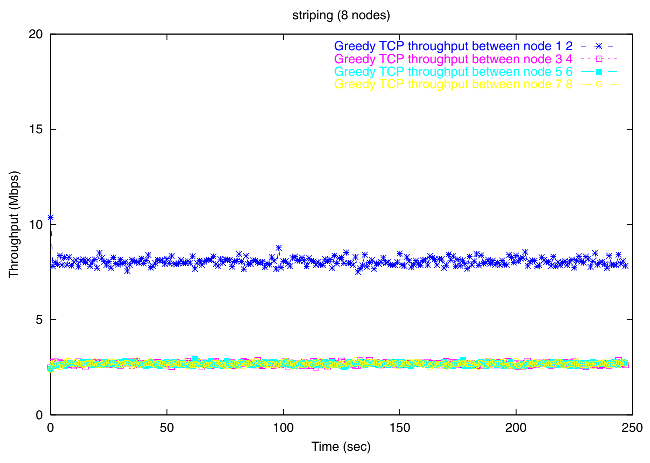

Figure 21. The achieved TCP throughput on a three-channel wireless trunk using our striping method, three channels are experiencing congestion.

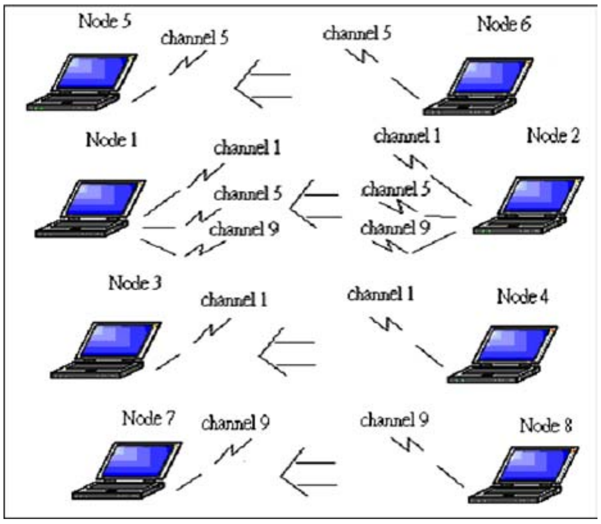

Figure 20. The simulation environment where three channels used by a three-channel wireless trunk are experiencing congestion.

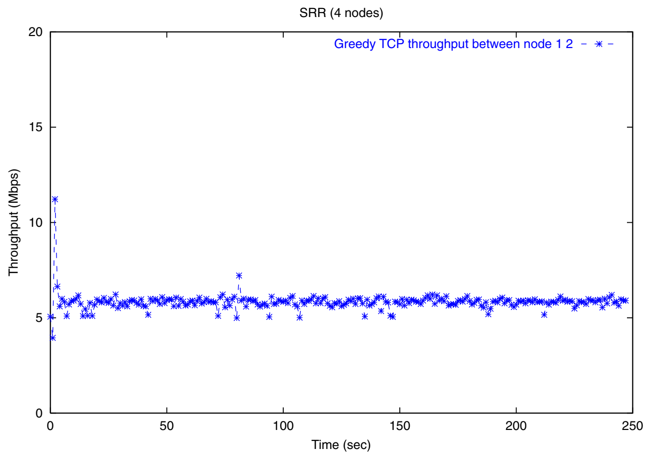

Figure 22. The achieved TCP throughput on a three-channel wireless trunk using SRR, one channel is experiencing congestion.

To further increase congestion, in figure 20 we add node 7 and node 8. Their wireless NICs are configured to use channel 9, which is the third channel used by the wireless trunk. Figure 21 shows the results. As expected, the wireless trunk receives the ideal maximum achievable throughput of 7.5 Mbps (2.5 + 2.5 + 2.5) under the created congestion condition.

The results of these congestion cases show that our striping method can enable a wireless trunk to receive its ideal maximum throughput. To compare the performances of SRR and our striping method under congestion condition, we used the environment depicted in figure 16 to run the simulation. Figure 22 shows the results. We see that the achieved throughput of SRR is only 6 Mbps, which is far less than the 12.5 Mbps achieved by our striping method. The poor SRR throughput is caused by frequent triggerings of TCP congestion control, which in turn are caused by excessive out-of-order packets caused by congestion.

### *4.2.3. Bit-error condition*

In all above cases, we assumed that there is no bit error on wireless channels. As such, no packet will be dropped due to corruption. In the real world, however, bit errors can occur due to multipath fading, noise, Doppler shift, and many other factors. Here we introduce bit-error condition to the simulation environment to see how our striping method will perform.

For comparison, we first ran non-striping cases in which only one channel is used and its BER is set to different non-zero values. For a specified BER and a received packet, we compute the corresponding PER (packet error rate) for this packet. We then draw a random number between 0 and 1 to determine whether this packet should contain error bits. If so, this packet is dropped.

Figure 23 shows the results. We see that as BER increases the achieved TCP throughput decreases. The achieved throughput is about 4.9 Mbps, 4.5 Mbps, and 4.1 Mbps when BER is 0.000001, 0.000005, and 0.00001, respectively.

Then we evaluated our striping method under bit-error condition. The used environment is the same as that depicted in figure 10 except that at this time the BER is set to a non-zero value and applied to all channels. Figure 24 shows the results. We see that the achieved throughput of the three-channel wireless trunk is about 14.7 Mbps, 13.5 Mbps, and 12 Mbps when BER is 0.000001, 0.000005, and 0.00001, respectively. Compared with figure 23, clearly our striping method enables a wireless trunk to achieve the ideal maximum throughput (3 times of the achievable throughput on a single channel) under bit-error condition.

To see how SRR will perform under bit-error condition, we used SRR to perform the same simulation test. Figure 25 shows the results. Here we see that SRR performs very badly due to many packet droppings.

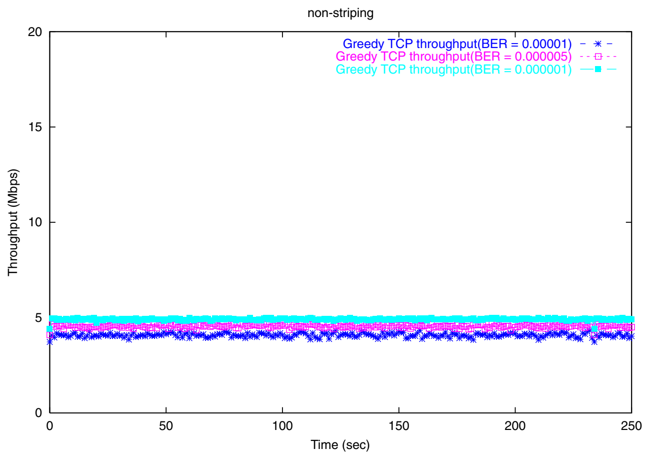

Figure 23. The achieved TCP throughput on a single channel with different BERs.

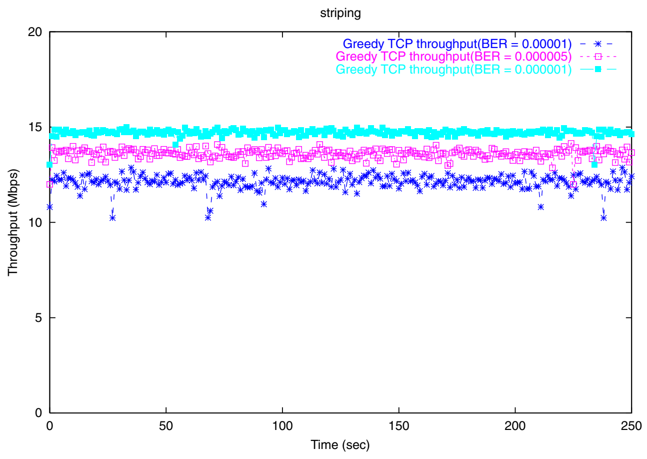

Figure 24. The achieved TCP throughput on a three-channel wireless trunk using our striping method, with different BERs.

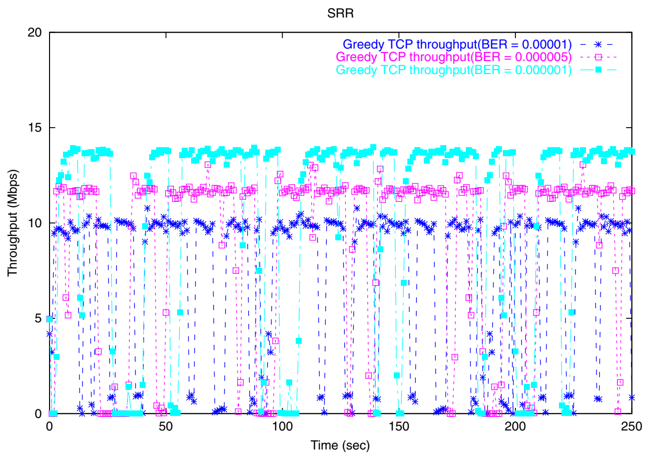

Figure 25. The achieved TCP throughput on a three-channel wireless trunk using SRR, with different BERs.

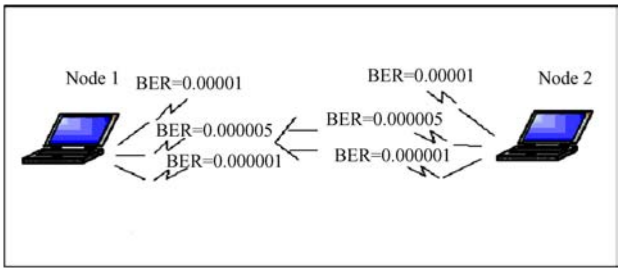

Figure 26. The simulation environment where the channels used by a three-channel wireless trunk are experiencing different BERs.

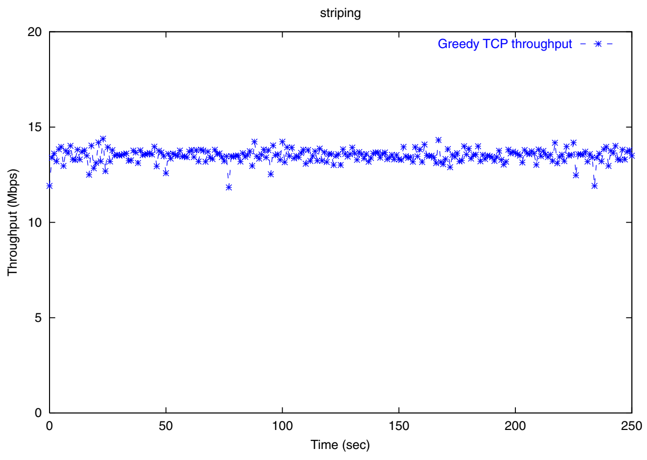

Figure 27. The achieved TCP throughput on a three-channel wireless trunk using our striping method, each channel experiencing a different BER.

In each of the above cases, the same specified BER is applied to all channels. Here we tested another case in which different BERs are applied to different channels. The simulation environment for this case is shown in figure 26. Figure 27 shows the results. As can be seen, the achieved throughput is about 13.5 Mbps, which is the ideal maximum achievable throughput (4.1 + 4.5 + 4.9).

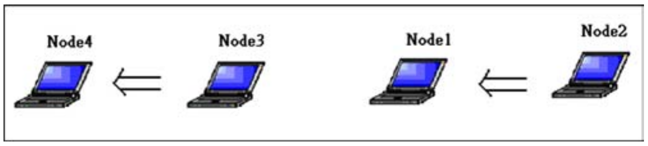

Figure 28. A simulation environment that has the hidden-terminal problem.

### *4.2.4. Hidden-terminal condition*

In wireless communication, the hidden-terminal problem is a common and serious issue. Here we use the simulation environment depicted in figure 28 to briefly explain it. Our simulation cases are also performed using this environment.

In this environment, all nodes use the same frequency channel. Node 1 is in the transmission range of node 3 but node 2 is not. That is, node 1 can receive the signal of node 3 but node 2 cannot. Also, node 3 is in the transmission range of node 1 but node 4 is not. During a simulation, node 2 and node 3 greedily send their data to node 1 and node 4, respectively.

Although IEEE 802.11(b) MAC protocol implements a CSMA/CA (carrier sense multiple access with collision avoidance) mechanism to avoid packet collisions, in a hidden-terminal environment packets can still collide with each other and get dropped. For example, suppose that node 3 is sending a packet to node 4 when node 2 wants to send a packet to node 1. Even though node 2 performs carrier-sense trying to detect an ongoing packet transmission, it cannot detect the signal of the packet that is being sent from node 3 to node 4. As such, node 2 decides to send out its packet to node 1. However, this packet cannot be successfully received by node 1 because its signal will get collided with the signal being sent out by node 3.

Due to these collisions, the packets sent from node 2 to node 1 will constantly get corrupted at node 1. However, the packets sent from node 3 to node 4 will not be affected at node 4. As such, when these packets are greedy TCP packets, our simulation results show that the TCP connection from node 3 to node 4 can achieve 5 Mbps throughput. However, the TCP connection from node 2 to node 1 cannot achieve any throughput. (Its throughput is almost 0 Mbps due to excessive TCP timeouts.) Since the maximum throughput that can be achieved by a wireless trunk is the sum of the achievable throughput on each channel, we can expect that the achievable TCP throughput on a wireless trunk that suffers from the same hidden-terminal condition would also be close to 0 Mbps. Indeed, our simulation results confirm this reasoning.

Since TCP results are not interesting, we do not present them in this paper. Instead, in the following we present the achievable UDP throughput on a wireless trunk. First, we ran a basic case in which there is a greedy UDP flow from node 2 to node 1 and from node 3 to node 4, respectively. Figure 29 shows the result. We see that because UDP is not sensitive to packet losses, unlike a TCP flow, the UDP flow from node 2 to node 1 can still receive some bandwidth (0.3 Mbps). The UDP flow from node 3 to node 4 receives about

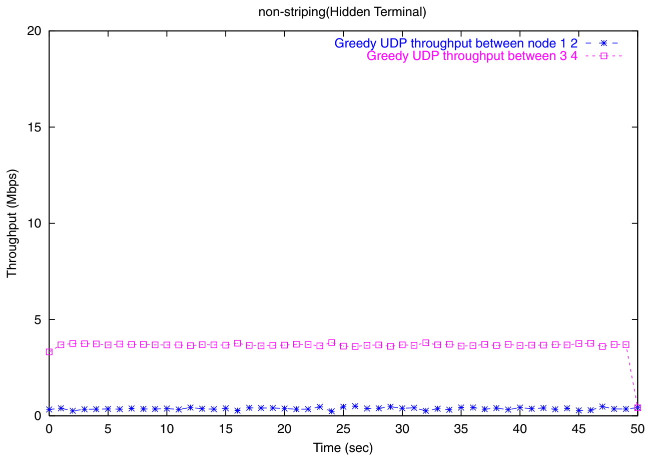

Figure 29. The achieved throughput of two greedy UDP flows in the hidden-terminal condition.

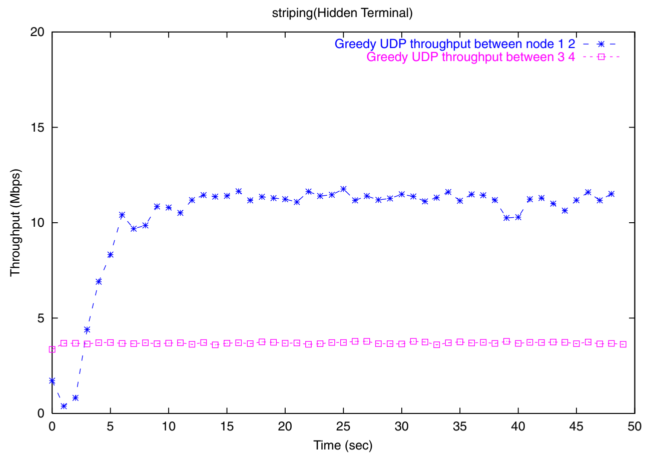

Figure 30. The achieved UDP throughput of a three-channel wireless trunk using our striping method, one channel is in the hidden-terminal condition.

4 Mbps, which is not as high as that (5 Mbps) in the previous TCP case. The reason is that when node 1 sends back IEEE 802.11(b) ACK frames to node 2 for received UDP packets, its signal will reach node 3. Because node 3 performs carrier-sense, it will refrain itself from sending out its packets for sometime. Since the UDP traffic volume is not zero, node 3 cannot receive the full 5 Mbps channel throughput in this UDP case.

Then we set up a three-channel wireless trunk between node 1 and node 2 and let one of these channels use the same frequency channel as that used between node 3 and node 4. Figure 30 shows the UDP throughput of the wireless trunk using our striping method. We see that the trunk can stably receive a maximum aggregated throughput of about 10.3 Mbps, which is the sum of 0.3, 5, and 5. This result shows that the wireless trunk using our striping method can achieve the highest throughput that it is possibly to achieve. On the other hand, Figure 31 shows the results of using SRR. We see that the achieved UDP throughput of SRR is less

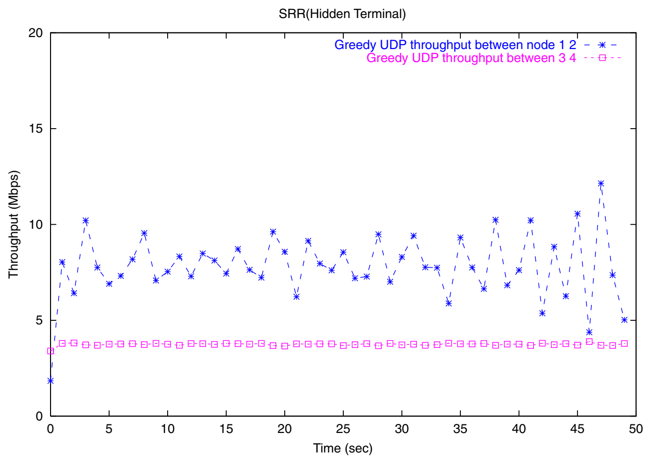

Figure 31. The achieved UDP throughput of a three-channel wireless trunk using SRR, one channel is in the hidden-terminal condition.

than that of our striping method and its achieved throughput fluctuates greatly.

### *4.2.5. Mobility condition*

When either the trunk sender or the receiver moves, the power of the signal received at the trunk receiver will vary. Because the BER of the received packets depends on the signal-to-noise ratio, it will vary as well. Here we use the free space equation (i.e., Friis equation [7]) to model the power of a received signal as a function of distance and use the modulation scheme used by IEEE 802.11(b) (i.e., CCK) to derive the BER v.s. power curves.

The simulation environment used in this case is depicted in figure 32. Three wireless channels are used between node 1 and node 2 and a greedy TCP connection is set up from node 2 to node 1. During the simulation, we let node 1 and node 2 move away from each other at a speed of 0.7 m/s.

First we ran a non-striping case. Figure 33 shows the result. We see that as the distance between the trunk sender and receiver increases, the achieved TCP throughput decreases due to increased BER. When the distance exceeds the effective transmission range, the achieved throughput almost drops to 0 Mbps due to huge BERs.

Then we evaluated the throughput of the wireless trunk using our striping method. Figure 34 shows the result. We see that although the throughput is also getting down when the distance increases, it is almost three times of the throughput achievable on a single channel. This shows that our striping

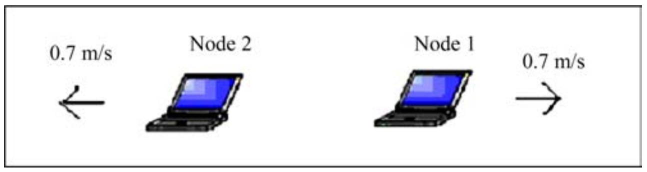

Figure 32. The simulation environment used to test the mobility case.

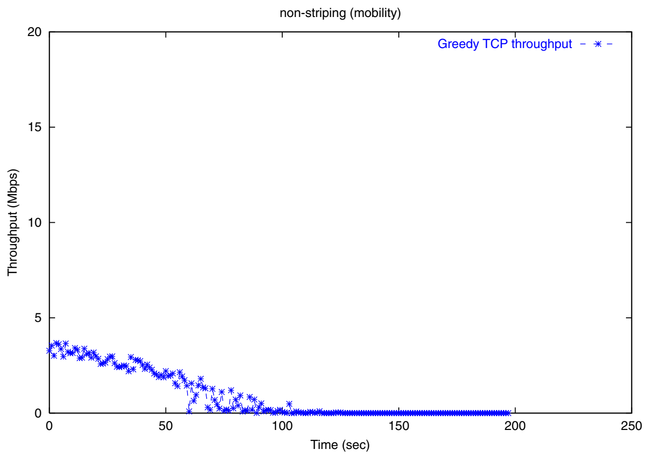

Figure 33 The achieved TCP throughput on a wireless channel, with the sender and receiver moving away from each other at a speed of 0.7 m/s.

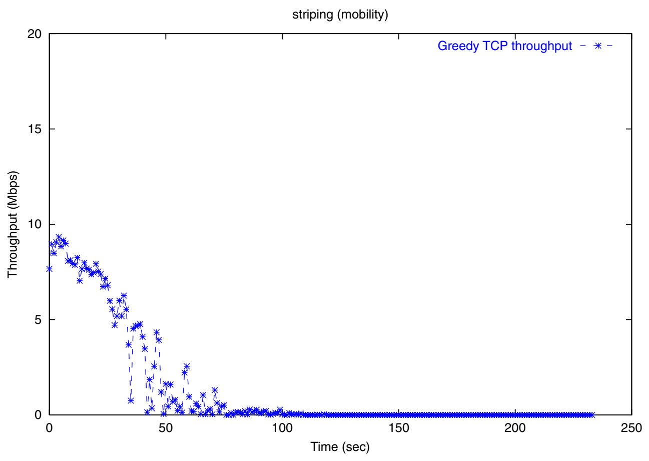

Figure 34. The achieved TCP throughput on a wireless trunk using our striping method, with the sender and receiver moving away from each other at a speed of 0.7 m/s.

method can perform well in mobility condition. Figure 35 shows the result of SRR. We see that when the BER goes up to a certain level, the achieved TCP throughput quickly drops to 0 Mbps without any further progress. This is due to excessive packet losses, which causes the TCP retransmit timer to back off exponentially.

### *4.2.6. Low traffic load condition*

When the traffic load offered to a multi-channel wireless trunk is less than the available bandwidth of a single channel, it is interesting to see whether spreading such a low traffic among the multiple channels of a trunk would result in poorer performance than transmitting the traffic on a single channel.

Figure 36 shows the achieved TCP throughput on a 3-channel wireless trunk and on a single channel, respectively. We vary the TCP traffic load by varying the data payload generation time interval. The maximum TCP traffic load generated in this test (650 KB/sec) is still less than the available

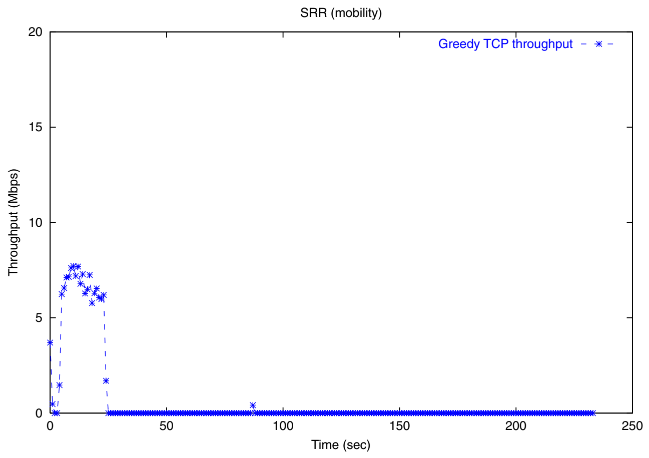

Figure 35. The achieved TCP throughput on a wireless trunk using SRR, with the sender and receiver moving away from each other at a speed of 0.7 m/s.

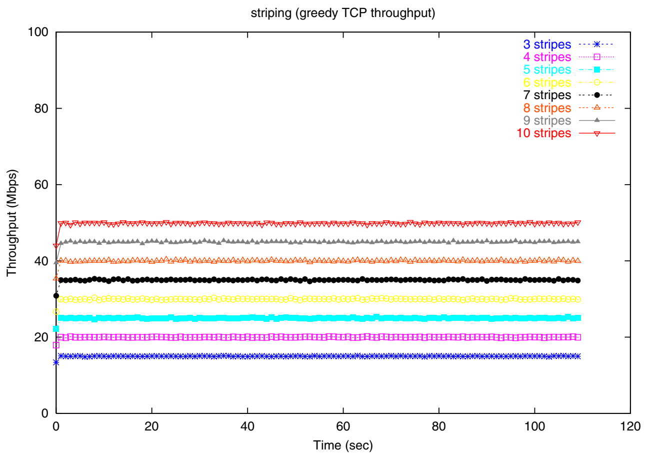

Figure 37. The achieved throughput scales linearly with the number of used channels, using our striping method.

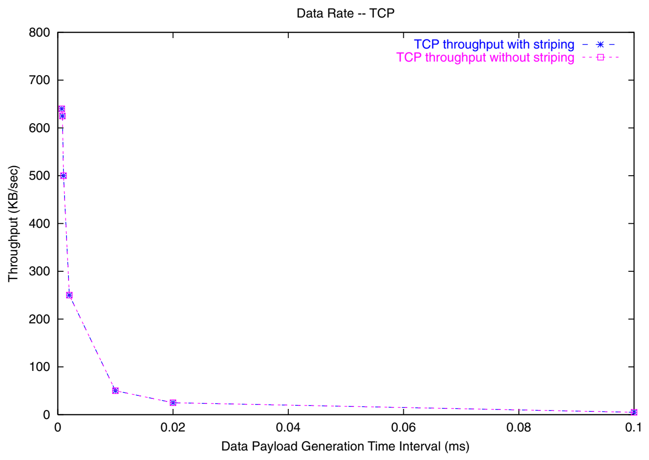

Figure 36. The achieved TCP throughput on a 3-channel wireless trunk and on a single channel, respectively. The maximum TCP traffic load is still less than the available bandwidth of a single channel.

bandwidth of a single channel. From this figure, we can see that the achieved TCP throughput on a 3-channel wireless trunk is very close to that on a single channel. This shows that even when the offered load is less than the available bandwidth of a single channel, using a multi-channel wireless trunk will not generate worse performances than using a single channel.

### *4.2.7. Channel number scalability*

Here we study the scalability of our striping method with respect to the number of channels. In all above simulations, we used only three channels for striping. Although results show that our striping method performs well when applied to a three-channel trunk, we would like to see whether it can still perform well with a large number of channels.

We used the basic environment depicted in figure 10 and varied the number of channels from three to ten. These channels are configured to use different frequency channels. In the real world, using successive IEEE 802.11(b) frequency channels may cause some interference. However, since this issue is not the focus of this paper, we assumed that no such interference exists in simulations. Figure 37 shows the result. We see that the achieved throughput scales linearly with the number of used channels. That is, when N channels are used, the achieved throughput is about $N * 5$ Mbps. This result shows the scalability of our striping method.

### *4.2.8. Multi-hop condition*

Here we study how our striping method would perform when applied to a multi-hop multi-channel wireless chain network. The packet forwarding throughput of a multi-hop single-channel wireless chain network has been studied in [10]. In that paper, the author shows that due to wireless signal interference among neighboring stations, the maximum achievable forwarding throughput on a N-hop wireless chain network is about 1/N and stabilizes at around 1/7 of the throughput achieved on a single-hop network. Figure 38 shows the TCP throughput achieved on a single-channel wireless chain network with different number of hops.

We tested our striping method on a three-channel multi-hop wireless chain network. Figure 39 shows the TCP throughput achieved on such a network using our striping method. Comparing figure 39 to figure 38, we see that our striping method also performs well on a multi-hop wireless network. This is because the achieved throughput improvement is almost 3, which is the number of striping channels used per hop.

### *4.2.9. Packet delivery latency*

Packet delivery latency is an important metric used to evaluate a striping method. In our evaluation, such latency is defined to be the elapsed time between two events. The first event is when the upper layer at the sending node asks the trunk sender to send out a packet. The second event is when the trunk receiver delivers the packet to the upper layer at the receiving

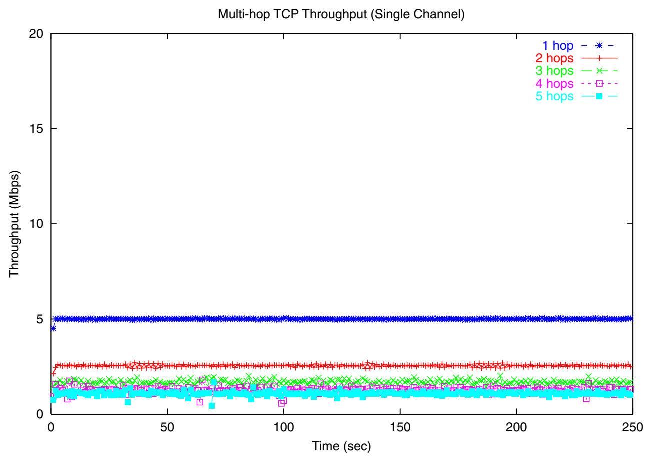

Figure 38. The achieved TCP throughput on a N-hop single-channel wireless chain network.

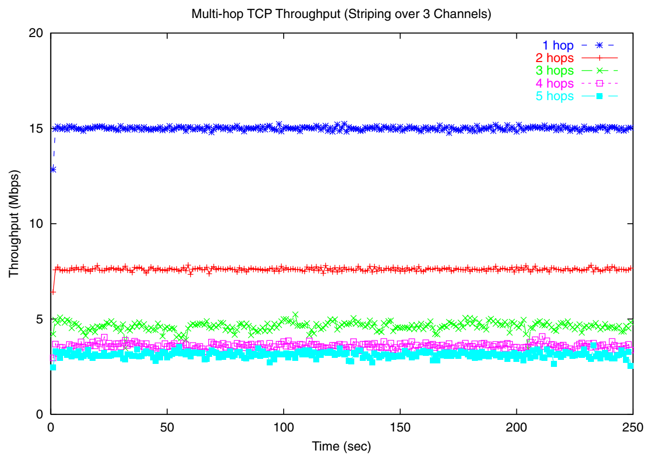

Figure 39. The achieved TCP throughput on a N-hop three-channel wireless chain network, using our striping method.

node. For real-time multimedia application programs such as video conferencing, small packet delivery latency is needed.

When a trunk is used, the main component of packet delivery latency is the time spent waiting in the resequencing queue in the trunk receiver. As the number of channels increases, it can be expected that packet delivery latency will increase as well due to a higher chance of packet reordering among these channels. It is interesting to see how packet delivery latency would change with respect to the number of channels.

We again used the basic simulation environment depicted in figure 10 and varied the number of channels from three to ten. The minimum, maximum, average, and standard deviation of the measured packet delivery latency are reported in Table 1. From the average latency results, we see that the latency roughly scales linearly with the number of channels. On average, an extra 30 ms is added to the latency when one more channel is added to the wireless trunk.

Table 1
The measured packet delivery latency under ideal channel condition

| | Latency (sec) | | | |
|---|---|---|---|---|
| Number of channels | min | max | avg | stdev |
| 3 channels | 0.001025 | 0.096252 | 0.083308 | 0.002576 |
| 4 channels | 0.001025 | 0.203725 | 0.088244 | 0.011838 |
| 5 channels | 0.001025 | 0.442918 | 0.144322 | 0.075733 |
| 6 channels | 0.001025 | 0.784718 | 0.244777 | 0.162877 |
| 7 channels | 0.001025 | 0.635399 | 0.234509 | 0.16618 |
| 8 channels | 0.001025 | 0.482005 | 0.214224 | 0.108651 |
| 9 channels | 0.001025 | 1.075736 | 0.322252 | 0.194462 |
| 10 channels | 0.001025 | 0.947715 | 0.347297 | 0.207922 |

Table 2
The measured packet delivery latency under bit-error condition, three channels are used.

| | Latency (sec) | | | |
|---|---|---|---|---|
| BER | min | max | avg | stdev |
| 0.0 | 0.001025 | 0.096252 | 0.083308 | 0.002549 |
| 0.000001 | 0.001025 | 0.133014 | 0.086926 | 0.004745 |
| 0.000005 | 0.001025 | 0.256145 | 0.110644 | 0.026783 |
| 0.00001 | 0.001025 | 0.669147 | 0.177871 | 0.102995 |

Table 2 shows the effect of BER on packet delivery latency. The simulation environment used is the same as that depicted in figure 10. Three channels are used for the wireless trunk and the specified BER is applied to all of them. From this table, we see that higher BERs do increase the measured packet delivery latency due to more packet reorderings.

From these results, we see that maintaining FIFO packet delivery order comes at a cost. Although, maintaining FIFO packet delivery order is critical to achieving good TCP throughput, it will also unavoidably increase packet delivery latency for application programs.

### *4.2.10. Reordering Timer's behavior*

The reordering timer plays an important role in our striping method. To see how its timeout value adapts to the current network condition, we ran the following simulation. The tested environment is depicted in figure 40. Node 2 and node 1 form a pair of nodes where a three-channel wireless trunk is set up between them. This trunk uses frequency channels 1, 5, and 9. Node 6 and node 5 form a pair of nodes that use channel 1 to exchange their data. Node 4 and node 3 form another pair of nodes that also use channel 1 to exchange their data. On each pair of nodes, a greedy TCP connection is set up.

At the beginning, the three greedy TCP connections are set active. That is, the TCP connection from node 6 to node 5 and the TCP connection from node 4 to node 3 are competing the bandwidth of channel 1 with the TCP connection on the wireless trunk. At 50'th second, the TCP connection from node 6 to node 5 is killed, resulting in only two active TCP connections. At 100'th second, the TCP connection from node

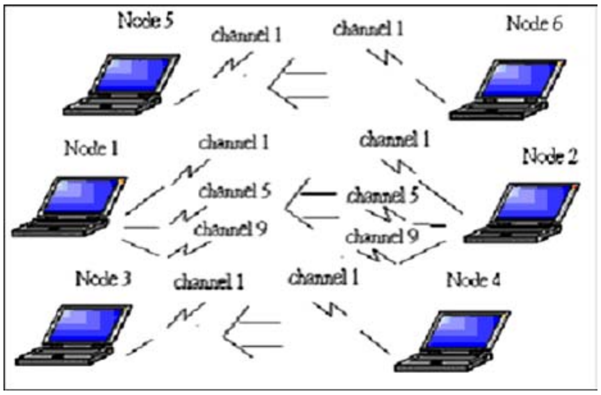

Figure 40. The environment used to observe the dynamic behavior of the reordering timer used in node 1.

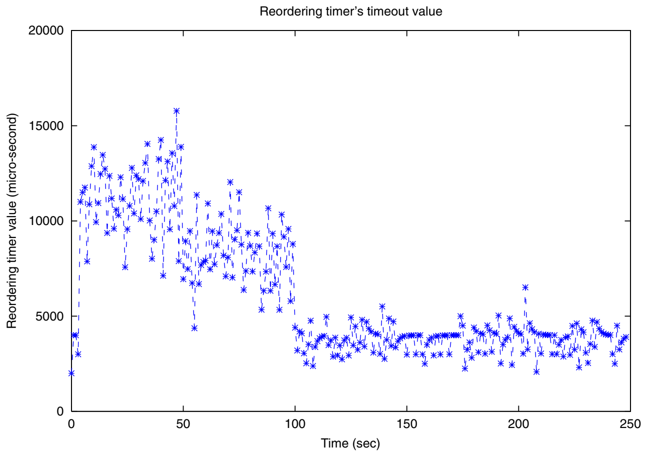

Figure 41. The dynamic behavior of the timeout value of the reordering timer used in node 1.

4 to node 3 is also killed, leaving no other TCP connection to compete with the TCP connection on the wireless trunk. With this arrangement, the level of congestion on channel 1 decreases as time proceeds. This allows us to observe how the timeout value adapts to the current congestion level.

Figure 41 shows the behavior of the timeout value of the reordering timer used in node 1. At 50'th and 100'th second, we see that it can adapt to the new congestion level quickly. In the first 50-second period where the congestion level is the highest, it ramps up to a high value (about 13 ms) and stays there. During the second period, it switches to a medium value (about 9 ms) and stays there. Finally, when there is no congestion, it switches to a low value (about 4 ms) and stays there. Although the timeout value fluctuates during the first and second periods, it is an unavoidable cost for letting the reordering timer quickly adapt to a new network condition.

## 5. Comparison with another scheme

SRR is proposed for wired networks with large bandwidth and infrequent bit errors. Here we compare the performances of our scheme with those of the scheme proposed in, which is proposed for wireless networks. For brevity, in the following discussion, we will call this scheme the JKM scheme.

The main differences between our scheme and the JKM scheme are as follows. First, the JKM scheme implements a selective reject ACK scheme with positive and negative ACK packets at the trunk level. However, our scheme does not implement an ACK scheme at the trunk level. Second, the JKM scheme will continuously issue retransmission requests to the trunk sender to ask it to resend a lost packet until it is eventually received. If there is a hole in received packets and the hole has not been filled, the trunk receiver will not deliver those received packets after the hole to the upper layer. In contrast, our scheme does not issue retransmission requests at the trunk level. It sometimes allows holes to be passed up to the upper layer to avoid delaying these arrived packets by too much time.

We have compared the performances of our scheme with those of the JKM scheme on a 3-channel wireless trunk. Our simulation results show that in all of the cases tested in this paper, our scheme performs better than the JKM scheme. However, to save space, here we only report the comparison results for three difference cases.

Figure 42 shows the achieved TCP throughput under the JKM scheme when there is no BER and congestion in the wireless channels. Comparing this figure with figure 12, which shows the achieved TCP throughput under our striping scheme, we see that the achieved TCP throughput in the JKM scheme is lower than that in our scheme. The throughput difference is due to the use of many ACK packets in the JKM scheme.

Figure 43 shows the achieved TCP throughput under the JKM scheme under varying BER conditions. Comparing this

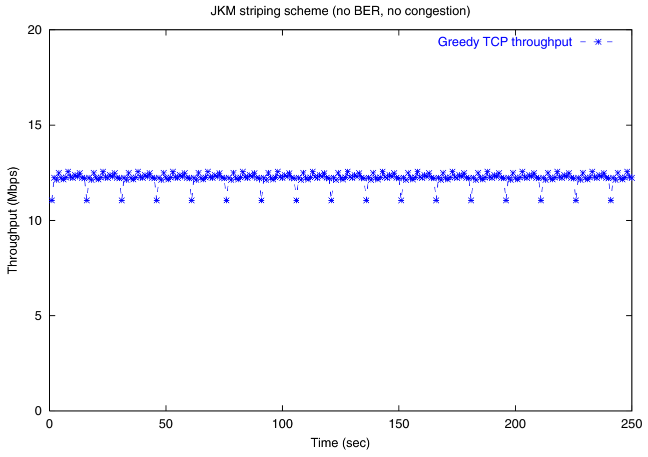

Figure 42. The achieved TCP throughput under the JKM scheme. No BER and no congestion.

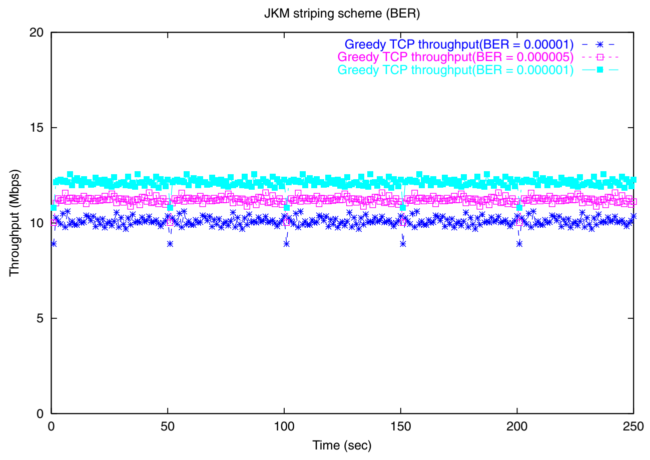

Figure 43. The achieved TCP throughput under the JKM scheme under varying BER conditions. No congestion.

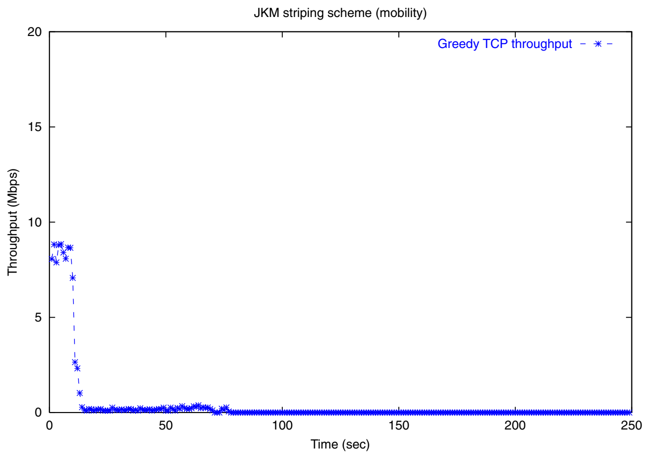

Figure 44. The achieved TCP throughput under the JKM scheme in mobility conditions. No BER and no congestion.

figure with figure 24, which shows the achieved TCP throughput under our striping scheme, we see that the achieved TCP throughput in the JKM scheme is lower than that in our scheme. Again, the throughput difference is due to the use of many ACK packets in the JKM scheme.

Figure 44 shows the achieved TCP throughput under the JKM scheme in the same mobility condition as described in figure 32. Comparing this figure with figure 34, which shows the achieved TCP throughput under our striping scheme in the same mobility condition, we see that the achieved TCP throughput in the JKM scheme is much lower than that in our scheme. We found that this significant throughput difference is primarily due to excessive TCP timeouts in the JKM scheme, which are caused because the JKM scheme delays arrived packets indefinitely for filling up holes at the trunk level.

## 6. Deployment discussion

Inter-cell interference (or called co-channel interference) means the signal interference that a mobile node may receive when it is in the overlapping area between two adjacent cells. To avoid this problem, generally one can assign different frequency channels to adjacent cells. Because in IEEE 802.11(b) there are three non-overlapping channels, normally one can use these channels to avoid inter-cell interference. In the proposed wireless trunk scheme, however, because a wireless trunk may use multiple channels, inter-cell interference now may exist because the channels used by a cell may be partially overlapped with those used by an adjacent cell.

In IEEE 802.11(b) there are eleven channels, among which three channels are totally non-overlapping while the other eight channels are partially overlapped. This means that these eight channels can still be used for site planning, although some minor co-channel interference may result among them. The impact of the resulting co-channel interference can be analyzed by treating the signal power of an adjacent channel as noise and use the SNR v.s. BER curves to derive the corresponding BER for a specific modulation scheme. Since the wireless trunk scheme proposed in this paper can be applied to IEEE 802.11(a) equally well, which has eight non-overlapping channels, we expect that this co-channel interference problem will be mitigated in IEEE 802.11(a) and future technologies that have more non-overlapping channels.

## 7. Future work

In the future, we plan to test the performances of our proposed striping method in the real-world environment. Currently, the form factor of notebook computers does not allow us to easily use multiple 802.11(b) wireless interface cards on a single notebook computer to conduct real experiments. We hope that future technology advancements can help remove this limitation.

## 8. Conclusions

Due to interference, path loss, multipath fading, background noise, and many other factors, wireless communication normally cannot provide a wireless link with a high data rate and a long transmission range at the same time. To address this problem, striping network traffic in parallel over multiple lower-data-rate but longer-transmission-range wireless channels can be a solution.

In this paper, we propose a new method for striping traffic over multiple IEEE 802.11(b) channels, and use the NCTUns 1.0 network simulator to evaluate its performances under various conditions. Tested conditions include ideal channel, congestion, bit errors, hidden-terminal, mobility, and channel number scalability. Extensive simulation results show that, under all of these conditions, the aggregated throughput achieved via striping traffic over N wireless channels is almost N times of the throughput achievable on a single channel. In addition, the extra resequencing delay introduced by this method is small. On average, only an extra 30 ms delay is introduced for each added wireless channel.

From these simulation results, we see that the proposed striping method can create a wireless trunk with both a high data rate and a long transmission range. It effectively overcomes the high-rate-but-short-range or long-range-but-low-rate problem in single-channel wireless communication.

## Acknowledgments

We would like to thank the anonymous reviewers for their valuable comments. This research was supported in part by MOE Program for promoting Academic Excellence of Universities under the grant number 91-E-FA06-4-4, NSC under the grant number NSC-92-2213-E-009-063, and the Chung-Shan Institute of Science and Technology under the grant number BC93B13P.

## References

[1] Hari Adiseshu, Guru Parulkar and George Varghese, A reliable and scalable striping protocol, ACM SIGCOMM'96 (8) (1996) 131–141.
[2] C. Brendan, S. Traw and J.M. Smith, Striping within the network subsystem, IEEE Network (July/August 1995) pp. 22–29.
[3] CISCO Aironet 350 Series Wireless Bridge Data Sheet, available at http://www.cisco.com
[4] IEEE Computer Society LAN MAN Standards Committee, Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, IEEE Std 802.11-1999. The Institute of Electrical and Electronics Engineering, New York, 1999.
[5] F. Jacquet, Philippe Kauffmann, Michel Misson, Striping schemes for wireless communication system, The Fifth IEEE Symposium on Computer and Communications, Antibes, France, 4-6 July 2000
[6] F. Jacquet and Michel Misson, Striping over Wireless Links: Effects on Transmission Delays, The 11th IEEE International Symposium on Personal Indoor and Mobile Radio Communications (PIMRC'00), 18-21 Sep. 2000.
[7] J.D. Parsons, *The Mobile Radio Propagation Channel* (John Wiley & Sons, LTD., 2000).
[8] Alex C. Snoeren Adaptive Inverse Multiplexing for Wide-Area Wireless Networks, IEEE GLOBECOM'99, December 1999.
[9] W. Richard Stevens, *TCP/IP Illustrated*, Volume 1 (Addison Wesley, 1995).
[10] S.Y. Wang, Optimizing the packet forwarding performance of wireless chain networks, Computer Communications 26(14) (2003) pp. 1515–1432.
[11] S.Y. Wang, C.L. Chou, C.H. Huang, C.C. Hwang, Z.M. Yang, C.C. Chiou and C.C. Lin, The Design and Implementation of the NCTUns 1.0 Network Simulator, Computer Network 42 (2) (June 2003) pp. 175–197. (Available for download at http://NSL.csie.nctu.edu.tw/nctuns.html)
[12] WaveLAN IEEE User's Guide, Lucent Technology, 1998.
[13] Wireless Data Forum. Cellular digital packet data system specification, Release 1.1 January 1995.

**S.Y. Wang** is an Associate Professor of the Department of Computer Science and Information Engineering at National Chiao Tung University, Taiwan. He received his Master and Ph.D. degree in computer science from Harvard University in 1997 and 1999, respectively. His research interests include wireless networks, Internet technologies, network simulations, and operating systems. He is the author of the NCTUns 2.0 network simulator and emulator, which is being widely used by network and communication researchers. More information about the tool is available at http://NSL.csie.nctu.edu.tw/nctuns.html.
E-mail: shieyuan@csie.nctu.edu.tw

**C.H. Hwang** received his master degree in computer science from NCTU in 2002 and currently is working for a network company.

**C.L. Chou** currently is a third-year Ph.D. student at the Department of Computer Science and Information Engineering, National Chiao Tung University (NCTU), Taiwan. He received his master degree in computer science from NCTU in 2002.