2010-02-09

Studying the Performance of Wireless Mesh Networks Using the HxH Transport Control Protocol

Timothy Scott Larsen
Brigham Young University - Provo

Studying the Performance of Wireless Mesh Networks using the HxH Transport Control Protocol

Tim Larsen

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Daniel Zappala, Chair
Kent E. Seamons
Robert P. Burton

Department of Computer Science

Brigham Young University

April 2010

ABSTRACT

Studying the Performance of Wireless Mesh Networks using the HxH Transport Control Protocol

Tim Larsen

Department of Computer Science

Master of Science

As the need to remain connected increases, more and more people are turning to wireless mesh networks because they reduce the need for network infrastructure. Unfortunately, TCP does not perform well in such networks. HxH, an alternate protocol, has shown great promise in simulations, but since it relies on exploiting passive feedback, real measurements are needed to determine how effective the protocol really is. This thesis uses a measurement study on a wireless mesh network to characterize the performance of the HxH protocol in real-world networks. Several aspects of the HxH protocol do in fact perform well on real networks, but the high rate of packet loss renders other aspects of the protocol ineffective.

ACKNOWLEDGMENTS

I extend my thanks to all those who helped me in this endeavor. To Sandi, my wife, for putting up with altogether too many late nights and for always helping me with my wording regardless of whether the content made any sense to her. To Rob and to Grant for letting me use them as sounding boards for technical and data presentation issues. To my adviser for all of his time, and to everyone else who helped make this thesis a reality.

Contents

Chapter 1

Introduction

As technology advances, it is becoming more and more important for many people to be connected to a network wherever they may be. The popularity of broadband Internet access has created a large demand for network infrastructure such as fiber optic cables. While fiber can provide a fast connection to the Internet, it cannot provide users with ubiquitous access to the network. Wireless technology allows the reach of the existing infrastructure to be increased dramatically and also provides for more flexible access. When wireless networks are extended to multiple hops, the reach of the current network infrastructure can be greatly increased. Multi-hop wireless mesh networks are beginning to be deployed in several places such as the San Francisco Bay area (Bay Area Wireless User Group) [1], (SFLan) [15], Cambridge, Massachusetts [9], Champaign-Urbana, Illinois [3], Seattle, Washington [14], and Southampton, UK [16].

While these deployments have great promise, their success will be limited if they continue to use TCP as a transport protocol. It is well known that TCP has significant problems in multi-hop wireless networks [17], causing applications to receive very low throughput. A number of solutions have been proposed [17] [6] [13], but none have received extensive testing or been deployed yet.

The HxH transport protocol attempts to improve performance in wireless networks by using hop-by-hop mechanisms for reliability and congestion control. This approach can work well for multi-hop wireless networks because it can take advantage of information available at each pair of hops that cannot be easily accessed at the endpoints of a traditional end-to-end

protocol. HxH has two major improvements over end-to-end protocols: passive ACKs and hop-by-hop rate control. A passive ACK occurs when a wireless node listens to its neighbor's transmission and determines which packets the neighbor successfully sends downstream. This eliminates the need to explicitly send a separate ACK packet, saving considerable bandwidth. Hop-by-hop rate control allows a node to react more quickly to changing conditions and to avoid loss by controlling how fast it sends to its neighbor. The significantly lower round trip time on a single hop as opposed to a full round trip allows for much quicker adjustments to the sending rate at each node. In simulations, the HxH protocol can double the throughput achieved, as compared to TCP, depending on the network configuration and the workload.

While excellent results in a simulation are exciting, there are aspects of any simulator that are based on simplifications of real-world conditions. Wireless signals are generally assumed to propagate and degrade uniformly. It is also typical to assume that there is no background interference or other signals that are not part of the simulator. It is important to know whether these simplifying assumptions affect the effectiveness of the HxH protocol, specifically whether the presence of background interference and obstacles makes it less likely that passive acknowledgments will be received, thus requiring the use of explicit acknowledgments. Likewise, the effectiveness of hop-by-hop rate control depends on passive feedback and may suffer due to interference.

This thesis conducts a measurement study of a version of the HxH transport protocol run on a real multi-hop wireless network, which allows various aspects of the performance of the protocol to be measured. The tests are run on chains of various lengths, with and without the RTS/CTS exchange, and with varying buffer sizes at each hop. The measurements are analyzed to get a more complete picture of how the HxH protocol behaves in a real wireless network.

In general the HxH protocol implementation does not have better throughput than TCP. With the right tuning it is nearly as efficient, but there is a lot of variability in all of the measurements. The real wireless network, with its loss and interference, presents

significant challenges for this implementation of HxH, but much of that is due to design and implementation choices. The data gathered from these experiments suggests some ways we could improve the HxH implementation in the future, including using a different mechanism to listen promiscuously to the interface and inject packets on the interface.

Chapter 2

Related Work

Many people are interested in harnessing the benefits of a mesh network, and there a number of real-world deployments of various scales and different purposes. Some of them are designed to test new protocols, and some of them are intended to fill a need for free network access in the community. After discussing some of these mesh networks this chapter focuses on the differences between other leading protocols for mesh networks and the HxH protocol.

2.1 Real Mesh Networks

Multi-hop wireless networks have been simulated extensively for academic purposes, but these simulations have limitations. It is difficult to model wireless signal propagation under idealized conditions, but simulating the real world with its obstacles and interference is prohibitively complex. Because of these limitations academic research is focusing on the measurement of real mesh networks. There are a number of real mesh networks around the world, with some dedicated to academic research and some for the benefit of the communities around them.

Roofnet [9] is an experimental mesh network set up by researchers at the MIT Computer Science and Artificial Intelligence Laboratory. It currently consists of about 20 nodes and provides broadband Internet access to users in Cambridge. Roofnet is primarily used as a testbed for new routing and transport protocols that take advantage of the unique properties of radio.

SeattleWireless [14] is a grassroots effort to provide wireless Internet access to the community without monthly charges and that does not depend on a commercialtelecommunications provider. It is not a network designed for research, but a real network designed and maintained by volunteers.

SFLAN [15] is a similar project in the San Francisco Bay area. The network itself is still considered experimental, but its aim is to provide residents with free wireless Internet access.

The Champaign-Urbana Community Wireless Network [3] is a coalition of wireless developers and volunteers whose mission is to develop decentralized community-owned wireless mesh networks. They develop software and consult with planners to build networks. They have built community networks in Chicago and Washingon, D.C.

The Southampton Open Wireless Network [16] is a mesh network being deployed by the University of Southampton. It aims to be a free-to-use wireless network for students and residents of Southampton and a research tool for the university.

In addition to these academic and community-oriented networks, there are companies such as MeshDynamics [8] and Chaska.net [2] that provide commercial wireless mesh networks. MeshDynamics provides single-radio solutions as well as more robust multiple radio solutions.

2.2 HxH

HxH [13] improves throughput in multi-hop wireless networks through the use of credit-based congestion control at each hop and the use of passive acknowledgments for end-to-end reliability. The congestion control at each hop avoids packet loss by sending only when the downstream neighbor has buffer space. Passive acknowledgments allow for reliability but improve throughput as they do not consume extra bandwidth. Because the protocol is designed to respond to changes in the network conditions at each pair of nodes it is able to respond much more quickly to such changes than protocols that make decisions based on

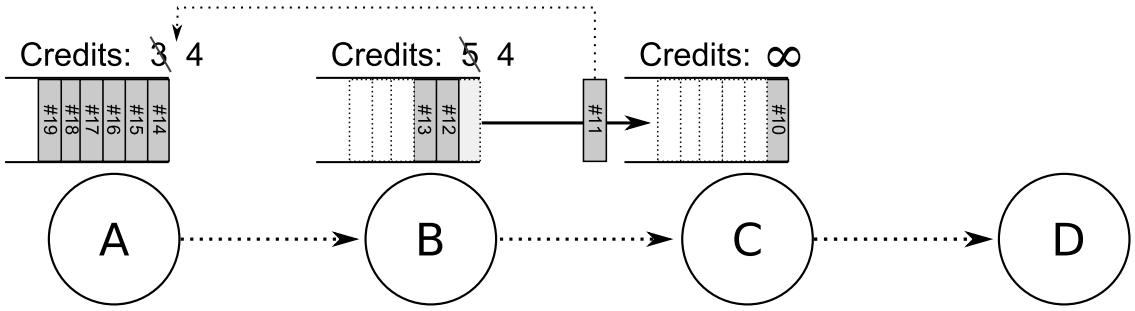

Figure 2.1: Credit-based flow control in HxH. When node B transmits a packet it decrements its pool of credits. Node A overhears the transmission and increments its pool of credits. The penultimate node in the chain is always considered to have infinite credits.

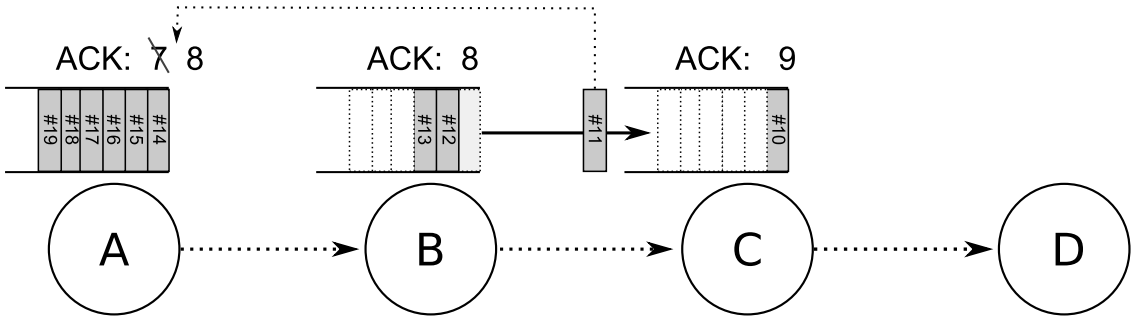

Figure 2.2: Passive acknowledgments in HxH. When node B transmits a packet it includes its current ACK number. Node A overhears the transmission and updates its ACK number accordingly.

end-to-end network conditions or which collate information and make the decisions at the endpoints.

In HxH, each node maintains a separate queue for each flow that is passing through the node. HxH determines which flow a packet belongs to by checking the header for source and destination addresses and ports. If a flow has not been seen before a new queue is created. HxH also maintains state for each flow, including the amount of queue space currently available downstream and an ACK value for the most recently overheard passive acknowledgment. The queue and state for a flow is deleted after a timeout period in which no packets for the flow are received. These queues are serviced using fair queuing, which ensures that flows traversing the same path are treated fairly. If the network layer reports a problem sending a packet it is put back into the appropriate queue and rescheduled for

later. If HxH detects a route failure, flows on the broken route are suspended until a new route has been established.

For congestion control the HxH protocol uses a credit-based algorithm, meaning that a node does not transmit unless it has a credit. In a traditional credit-based algorithm, these credits are transferred from the destination to the source. Because a node can overhear transmissions from its downstream neighbors HxH is able to transfer the credits implicitly, reducing bandwidth. In HxH, the credits are exchanged as in Figure 2.1. Each node maintains a pool of credits which represent downstream buffer space. Each transmission to a downstream neighbor requires that the pool of credits be non-empty and decrements it by one. The credit pool is increased each time a downstream neighbor's transmission is overheard. Because such packets cannot always be overheard, HxH includes in its header the amount of buffer space available at the node so that upstream neighbors can correctly adjust their credit pools. Each node maintains an estimate of the time it takes a packet to travel to the next hop and back. If a node has no credits it will wait for five times this estimate and then transmit a packet anyway. This prevents the protocol from entering deadlock when several packets in a row from downstream neighbors are not overheard. To prevent buffer overrun in this case HxH advertises only half of its available buffer space.

For reliability between hops HxH relies on the link layer reliability built into the 802.11 protocol. This does not guarantee end-to-end reliability, however. In order to guarantee end-to-end reliability, HxH maintains a passive ACK value for each flow at each node. This value represents the cumulative sequence number that the destination node has acknowledged. The last hop can use link-layer reliability to know that the destination received the packet, and it updates its passive ACK value accordingly. All other nodes propagate this value upstream as shown in Figure 2.2. Each HxH packet contains a passive ACK value which can be overheard by the upstream node. As long as there is a steady stream of packets the passive ACK will eventually reach the source and the source can behave as if it had received an explicit ACK. HxH also incorporates end-to-end negative acknowledgments to help it recover from loss. If

the destination detects gaps in its received sequence numbers it will send a NACK to the source. At the end of a transmission or when no packets are expected for a while, passive ACKs will not be relayed back to the source. HxH addresses this problem by having the last packet to leave the source node's transport layer buffer set a flag in its header requesting an explicit ACK. The destination will send an explicit ACK when it sees this flag.

HxH has been implemented for the ns-2 simulator, where it has performed quite well [13]. It has been tested in a variety of network topologies with and without mobility. It has outperformed other transport protocols such as ATP [17] and TCP variants including TCP with adaptive pacing [6].

Chapter 3

Implementation Details

The HxH protocol described in the previous chapter was implemented for the ns-2 network simulator, which meant that the code itself is not easily ported to a real network stack. This chapter describes the implementation of the HxH protocol created for this thesis. We first discuss the general approach taken and some of the limitations of that approach, then discuss some compromises that we made to implement the HxH protocol using this approach. We end this chapter with a discussion of modifications to the HxH protocol that we use to increase the performance of this implementation.

3.1 General Approach

There are several general approaches that can be taken when implementing a network protocol, each with its own limitations. One approach is to write a true kernel implementation of the protocol that has access to the hardware directly. The advantage to this approach is that the implementation can be very efficient and can take advantage of all of the information the hardware has. The disadvantages include slower development due to difficulty debugging and troubleshooting and hardware-dependence. Because of these disadvantages, experimental implementations are rarely done this way.

A second approach is to use a framework that allows access to the various protocol layers from user mode. This approach allows for easier debugging and still provides for efficient implementations. There are several frameworks for implementing transport protocols for wired networks including MINET [4], Daytona [12], and Alpine [5]. MINET is a

framework that was created for use by college students in networking courses. It is designed such that they can implement their own versions of different portions of the network stack in user space, including variations of TCP and UDP. Daytona is a similar framework that was designed to make academic research into network protocols simpler and quicker. Both of these frameworks require application code to link with a special socket library. Alpine is a similar framework but allows for unmodified application binaries to use the user-level TCP stack. We were not able to get a version of any of these frameworks that worked for wireless networks.

A third approach, and the approach taken in this case, is to simply use a packet capture and injection tool and bypass the network stack altogether. In order to listen promiscuously to the wireless interface and send packets we used libpcap [11]. We used Kismet [7] during debugging but did not integrate it into the HxH protocol. libpcap is a packet capture utility that allows an application to read packets directly from the interface regardless of their intended destinations. Kismet is a packet capture utility that specializes in 802.11 networks and can detect wireless-specific information. This approach has the advantage of allowing the protocol implementation full control over what is written to and read from the network interface, but it has the disadvantage of not being able to take advantage of the existing network stack above the network layer. There are a number of other disadvantages to this approach that became apparent during development and testing.

We implement the HxH protocol on top of libpcap, which provides an ability to manipulate packets at the network layer, but does not provide any ability to modify packets at lower layers. A routing socket is used to query the kernel for the routes, and the MAC addresses are acquired by pinging the next hop and reading the MAC address from the ARP cache in /proc/net/arp. This architecture provides a simple basis for implementing the protocol as it avoided the complications of integrating tightly in the network stack, but it has disadvantages as well. Because libpcap provides no way to read the RTS/CTS/DATA/ACK exchange, there is no way to know for sure that a packet has been delivered to the next

hop. libpcap claims successful delivery when the packet is added to the driver's queue, not when the packet is sent on the interface or received at the destination. This window between the time the packet is reported sent and actually received proves problematic for the HxH protocol implementation, and requires some modifications as explained in the next sections. Additionally, the lack of any way to be sure the packet got through undermines some of the basic assumptions of the HxH protocol and requires some protocol modifications in order to work reliably.

3.2 Protocol Modifications

The HxH protocol as simulated uses a callback from the MAC layer to know when a packet has successfully been transmitted to the next node. Because libpcap provides no such mechanism, our implementation cannot use this knowledge to know when to remove a packet from its local buffer, nor can it use it to know for sure that a packet has reached the destination. This limitation means that without consuming more bandwidth to send explicit acknowledgments at each hop we have no way to provide local retransmission of lost packets. Without a local recovery method the HxH implementation falls back on end-to-end reliability. Observations of our network show that the packet loss is significant, and that using end-to-end reliability does negatively affect the performance of HxH.

In order to more closely approximate the way HxH works in the simulator we added a last hop ACK as shown in Figure 3.1. The destination explicitly acknowledges each packet it receives by sending an ACK with the sequence number of the packet it received, which allows the second-to-last node to know that its packet has reached the destination and confidently increase its cumulative passive acknowledgment number. Each packet is acknowledged individually, mimicking the way the MAC layer provides reliability. Because last-hop data loss can be handled by this mechanism the number of end-to-end NACKs is reduced. Unfortunately, the overall time needed to transfer data increases, especially when the last link has a high loss rate.

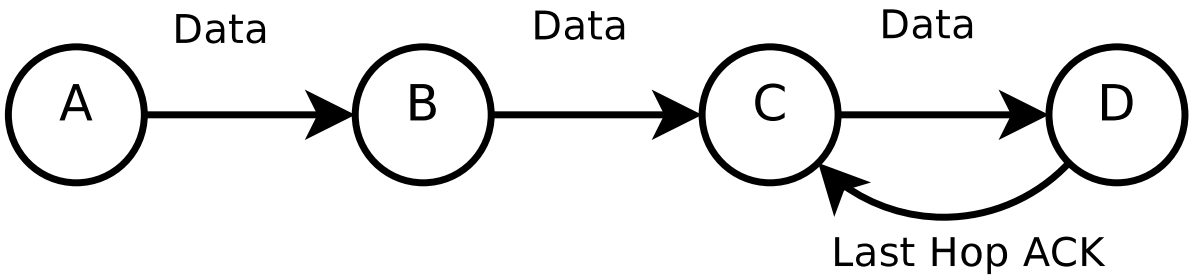

Figure 3.1: The destination node responds to each data packet by sending a last hop ACK with the same sequence number. The second-to-last node periodically resends the data packet until it receives a corresponding last hop ACK packet. The last hop ACKs are not forwarded, and this mechanism is used only on the last hop.

Because the performance sometimes degrades significantly with the last hop ACK, we also tested a version of the protocol without this modification. In general the version without the last hop ACK requires many more end-to-end retransmissions than the other version, but even with these extra retransmissions it has higher data throughput.

In addition to complicating the reliability of the HxH protocol, the choice to use libpcap also complicates the credits mechanism. In simulations the HxH protocol includes as part of every packet the number of credits available at the node that is transmitting. In our implementation this number is calculated right before the packet is given to libpcap. Unfortunately there is a small window of time between the queuing of these packets and the sending of them, and so by the time the packet is sent the credits number is frequently out of date. This can cause the upstream node to overflow the buffer of its downstream neighbor and require many more retransmissions. Figure 3.2 shows such a situation. Each node has a buffer capable of holding 6 packets, meaning that each node will initially have 3 credits for its downstream neighbor. Node A uses its three credits and sends three packets to node B. These packets are received by node B and sent (given to libpcap) in order. Because packet 2 has not yet arrived at node B when packet 1 is given to libpcap, the credits in packet 1 is still set to 3. Packets 2 and 3 also report 3 credits because there are no packets left in HxH's queue at node B. The timing of the next few packets is critical to showing the error. libpcap

on node B sends packet 1, which lets node A know that it has 3 credits. Node A immediately sends packets 4, 5, and 6 to libpcap, which sends them on the wireless interface. Node B receives these packets and stores them in its buffer. libpcap at node B then sends packet 2, which lets node A know that it has 3 credits. Node A immediately sends packets 7, 8, and 9 to libpcap, which sends them on the wireless interface. Node B receives these packets and stores them in its buffer. libpcap at node B then sends packet 3, which lets node A know that it has 3 credits. Node A sends packets 10, 11, and 12 to libpcap, which sends them on the wireless interface. Node B receives these packets but must discard them because its buffer is already full.

Our implementation of HxH mitigates this problem by taking into account not just the number of credits the downstream node reports, but how many packets have been sent to the downstream node since the one that is being overheard. Each node keeps a list of recently transmitted sequence numbers and when it overhears a transmission from its downstream neighbor it uses that list to compute how many packets it has sent to its downstream neighbor since it sent the packet that is now being transmitted by its downstream neighbor. It subtracts this number from the number of available credits advertised in the packet it overheard and uses the result as its number of credits. Figure 3.3 shows this adjusted mechanism at work. The scenario begins in the same way as the scenario in Figure 3.2, but it diverges as soon as node B sends packet 1. Node A overhears the transmission and adjusts its credits, but since two packets have been sent from node A since it sent packet 1, the number of credits A uses is 3-2=1. The adjustment in the credits keeps node A from sending too many packets at once to node B. The adjusted mechanism does occasionally report fewer credits than are actually available, but the credits numbers are quickly corrected by subsequent transmissions and the buffer overflows are avoided.

The end-to-end reliability scheme built into the simulated version of HxH is a fallback mechanism and is not intended to be the primary mechanism to ensure reliability. For each packet that is lost somewhere along the way the destination sends a NACK all the way back

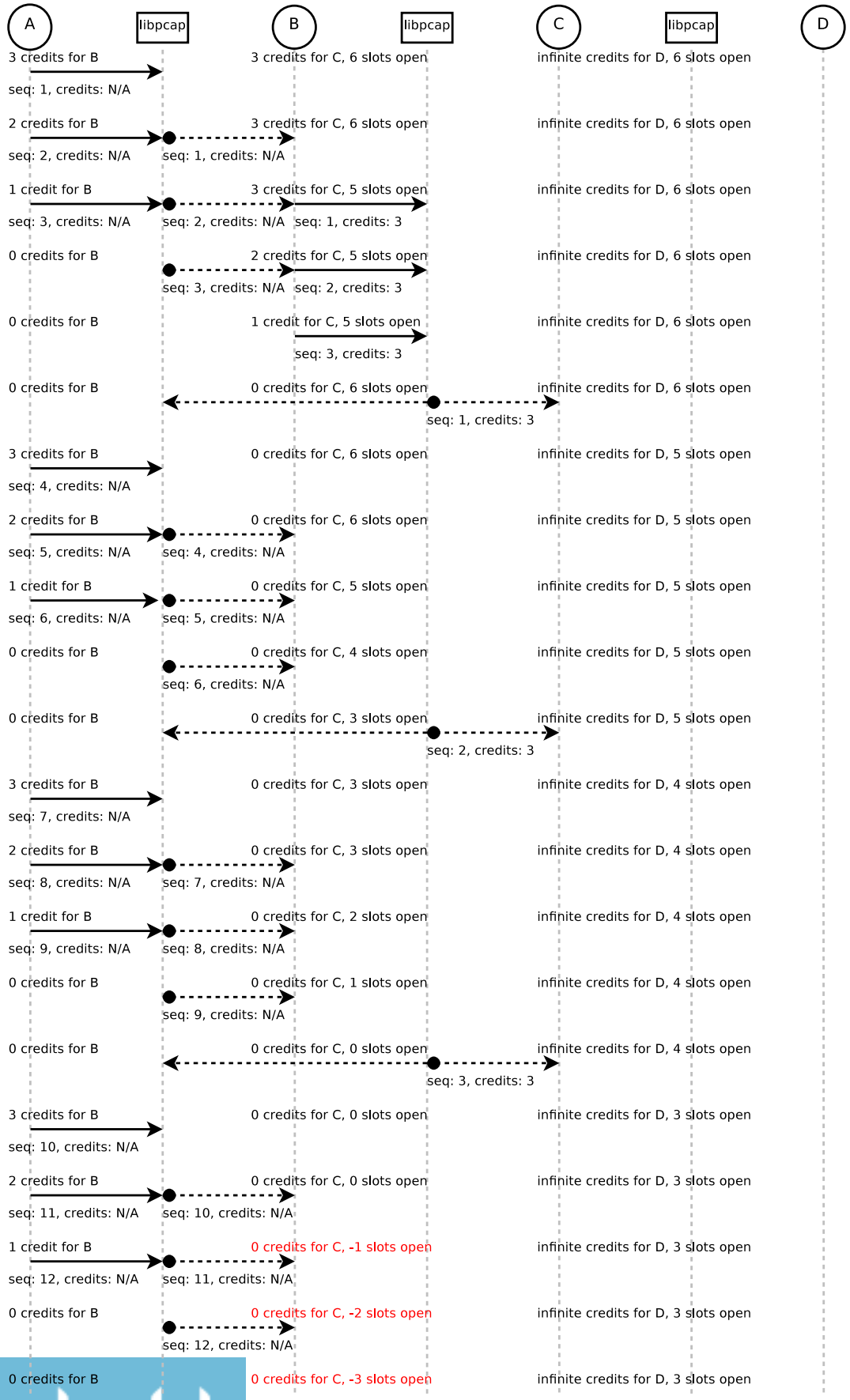

Figure 3.2: An example of how libpcap can cause HxH to send too many packets.

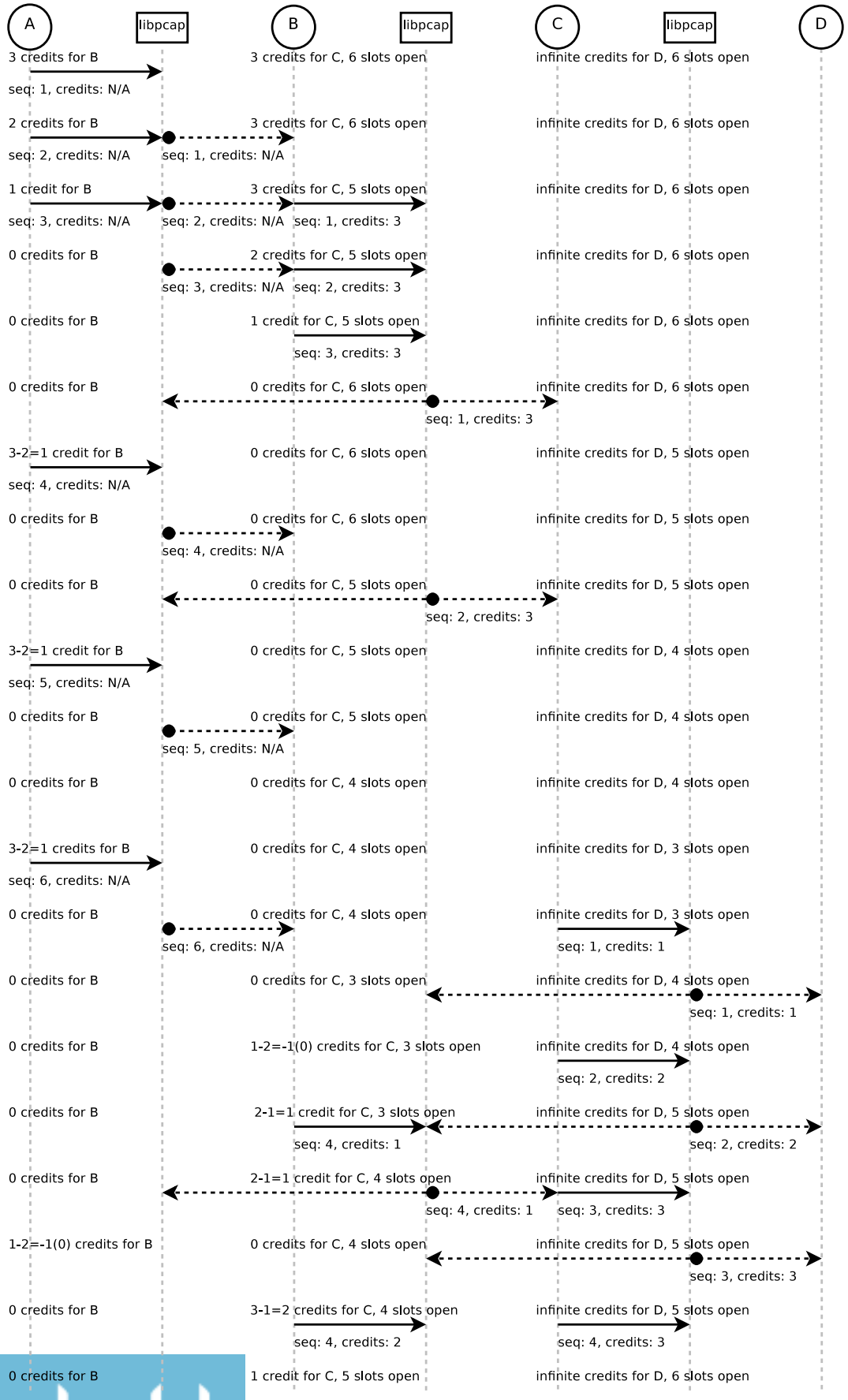

Figure 3.3: An example of how the adjusted credits mechanism can keep libpcap from causing HxH to send too many packets.

to the source. The source sends the missing packet all the way back to the destination, after which the destination then sends a NACK for the next lost packet. This mechanism is fine if the number of lost packets is expected to be very low, but because our observations showed the number of lost packets to be significant, this mechanism quickly becomes problematic. For our implementation we modify the NACK packet to include not just the sequence number of a single missing packet, but a list of the sequence numbers of all of the packets known to be missing. The source then parses this information and resends all of the missing packets, which reduces the number of packets that have to travel back and forth between the source and destination and the overall latency of the system.

The HxH protocol used in the simulations used a buffer size of 8 packets at each node. In simulations this small buffer size helps the packets flow in a fairly steady stream from one node to the next. As long as each node overhears the transmissions of its downstream neighbor this scheme works quite well to prevent bursty transmissions. In our implementation of HxH, however, the small buffer size means that when passive acknowledgments are lost the node must wait until a timeout occurs before transmitting the next packet. When we increase this buffer size the number of passive acknowledgments that must be lost before we wait on a timeout increases as well. Increasing the buffer size to 256 packets improves the performance of the protocol significantly.

Chapter 4

Methodology

In order to measure the performance of the HxH protocol implementation we create a wireless mesh network in the Talmage Math and Computer Science Building on the campus of Brigham Young University and run a variety of trials. The HxH protocol is instrumented such that the logs tell us a great deal about the performance of the protocol. This chapter discusses the setup of the mesh network, the experimental methodology, and the measurements that we take on the HxH protocol implementation.

4.1 The Mesh Network

The mesh network is assembled on the second floor of the Talmage Building. Figure 4.1 shows a map of the second floor and the positions of the wireless nodes. The machines are running Ubuntu 8.04 Server Edition and have 3COM 3CRDAGG75B 802.11a/b/g wireless cards. They use olsrd[10] for routing, but in order to get less variability in some of the larger chains we switch the routing to a predetermined set of static routes.

4.2 Experimental Methodology

Early tests of the wireless network show that the throughput varies significantly over time. For example, a link that might be strong one day might be weak the next. In order to reduce the effect of this variability on the results of our experiments we follow some general rules. The first is that tests where the throughput is to be compared directly are always run at the same general time. The variability in the network makes comparisons between

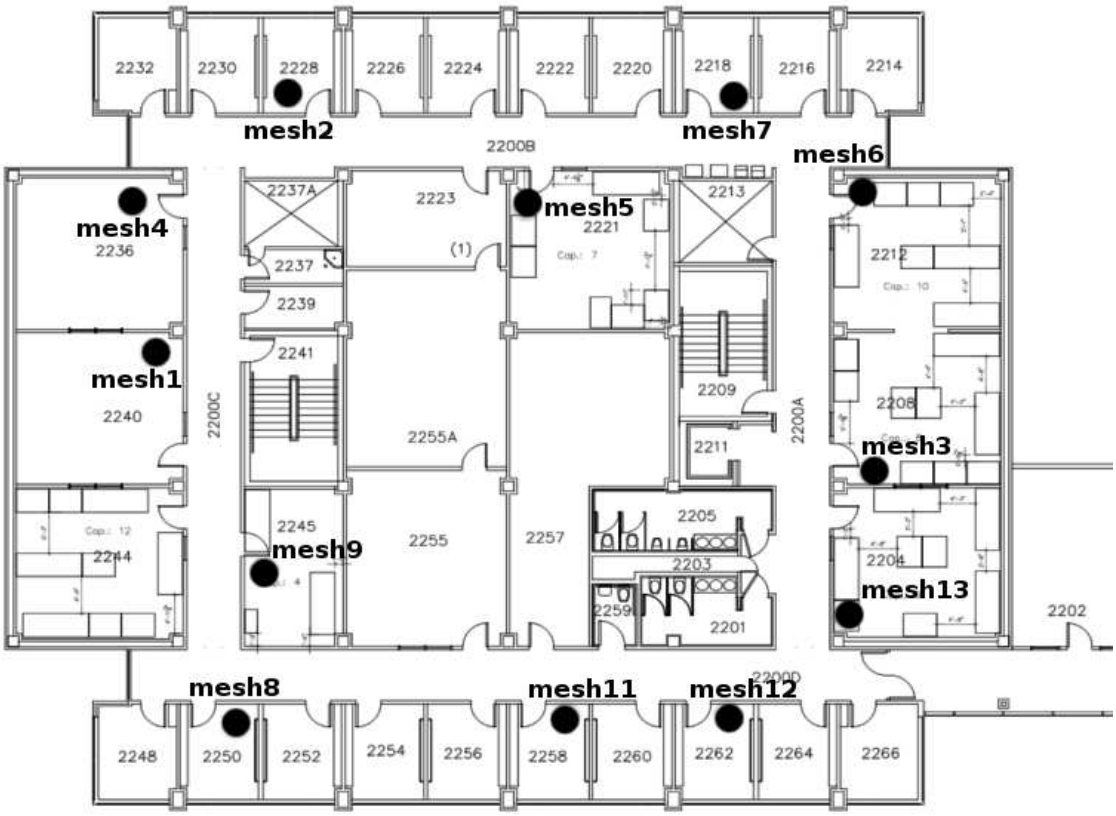

Figure 4.1: *The Wireless Network:* This map shows the locations of the nodes used for the experiments described in this thesis. They are all on the second floor of the Talmage Building.

trials run a few hours apart very difficult. The second is that we always run 30 trials of each protocol in order to mitigate the effect that outliers have on the trials. The last rule we use is to take turns running each of the protocols we measure (TCP, HxH, and HxH without the last hop ACK) to avoid having temporary variations in network conditions be mistaken for differences in protocol performance. This rule should cause the variations in the network to affect all of the protocols consistently and should allow for more meaningful comparisons.

Each trial consists of transferring a one megabyte file from the source node to the destination node. Nodes participating in the transfer use 802.11a, while nodes that are not participating in the transfer are changed to use 802.11b in order to eliminate interference. The trial is considered complete when the source node knows that the destination node has the data. For TCP, the only statistic we gather is the elapsed time. For HxH we gather much more data, including the times of transmission and reception at each node and what state the protocol is in at each node. Using this data we are able to reconstruct many aspects of the transfer and understand how much certain factors have to do with the performance of HxH.

We take several measurements to try to characterize the performance of the HxH protocol. The simplest measurement is the time it takes the protocol to transfer one megabyte of data. Connections of different numbers of hops are measured to try to characterize how the number of hops affects the throughput. In order to characterize how the throughput of the protocol is affected by the RTS/CTS exchange we measure three scenarios: (1) with the RTS/CTS exchange off, (2) with it on for packets over 500 bytes, and (3) with it on for all packets.

Other measurements that are taken include the round-trip time for a packet, that is, the time required between transmission of a packet and the source learning that the destination has received it. We also measure the percentage of passive acknowledgments at each hop that are not overheard by the upstream node. These measurements are used to

decide whether the passive acknowledgments are effective in relaying reliability information back to the source node.

The transmission rate over time at each node is also measured to try to understand whether the passive acknowledgments are relaying enough information upstream to keep the send rate nearly constant. When passive acknowledgments are lost, HxH sends packets at a suboptimal rate, and can cause the transmission to stall. Our measurements suggest that enough passive acknowledgments are lost to create significant problems for the HxH protocol.

The number of retransmissions is also measured in order to understand how often data packets are lost. The measurements are significantly different from those obtained with the ns-2 simulator, suggesting that environmental factors that contribute to packet loss are not modeled in ns-2.

Chapter 5

Results

This chapter presents the results of trials of an implementation of the HxH protocol in a real multi-hop wireless network. Unless otherwise stated, the network uses 802.11a with the RTS/CTS exchange turned off and olsrd [10] for routing. Throughput graphs are created using a one-second sliding window, plotted every 100 milliseconds. The cumulative distribution function graphs each represent the results of 30 trials. Loss rate graphs show the average loss for 30 trials, with error bars representing the 25th and 75th percentiles. Any exceptions to these standards are described in the text.

5.1 Throughput

In simulations the HxH protocol outperforms all of the TCP variants it is compared against. We expected a real-world implementation of HxH to similarly outperform TCP, but, in general, the throughput of HxH is not as good as that of TCP. There are several reasons this may be the case, including the fact that HxH is implemented on top of libpcap and not at the kernel level or using some more efficient mechanism, and the fact that a real multi-hop wireless network has significantly more packet loss than the simulator modeled.

The ns-2 model does a reasonable job of simulating wireless networks, but there are complexities in real-world networks that are extremely difficult to model in a simulator. The radios on each node in our network have different strengths in different directions, and certain radio links are consistent while others are much more variable. These inconsistencies in the strengths of the signals and the consistency of the links leads to much higher packet

loss than we initially expected to encounter and adversely affects the throughput of the HxH protocol. This packet loss will be discussed more fully as it pertains to retransmission rates and the effectiveness of passive acknowledgments later in this chapter.

5.1.1 Throughput on a Single Hop

In order to understand the characteristics of the network itself we measured the throughput between each pair of adjacent nodes. We found that some pairs of nodes have fairly consistent traffic patterns and some do not, and that the direction of the transfer is often an important factor. When mesh11 is transmitting to its neighbor mesh12, for instance, the TCP transfers complete in 0.70 to 0.91 seconds, whereas when mesh12 is transmitting to mesh11 the transfers complete in 1.23 to 1.34 seconds. Figures 5.1 and 5.2 show cumulative distribution functions of the times taken to transfer one megabyte using TCP, HxH, and HxH without lasthop acknowledgments for mesh11 to mesh12 and mesh12 to mesh11 respectively. In addition to the connection not being symmetrical the relative performance of the different protocols varies as well. The times for HxH remain fairly similar, which makes sense since each transfer requires a send from the source and a last hop acknowledgment sent from the destination. These trials were run with a buffer size of 8 packets for HxH, which is what was used in the simulator. This value does not appear to consistently work well with our implementation of HxH, and is responsible for the group of HxH trials that took almost 5 seconds to complete. HxH without the last hop acknowledgments clearly outperforms TCP when transmitting from mesh11 to mesh12, but is slightly slower overall when transmitting from mesh12 to mesh11.

The ns-2 simulator does not model this asymmetry in the transmission rates between two nodes, and we did not expect it to be so pronounced. As the HxH protocol uses passive feedback for reliability and congestion control it is particularly sensitive to connections that are highly asymmetrical. A node might be able to send consistently to its downstream

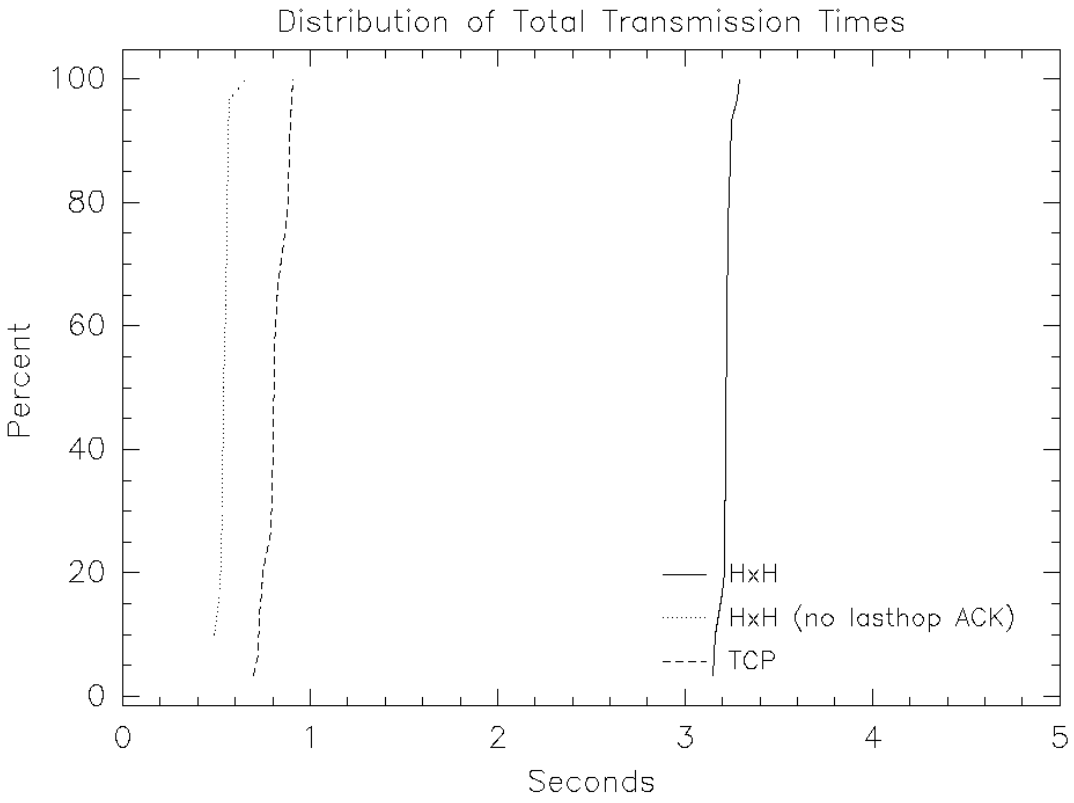

Figure 5.1: *Throughput from mesh11 to mesh12:* This graph shows the cumulative distribution function of the times taken to transfer one megabyte from mesh11 to mesh12 using TCP, HxH, and HxH without last hop acknowledgments.

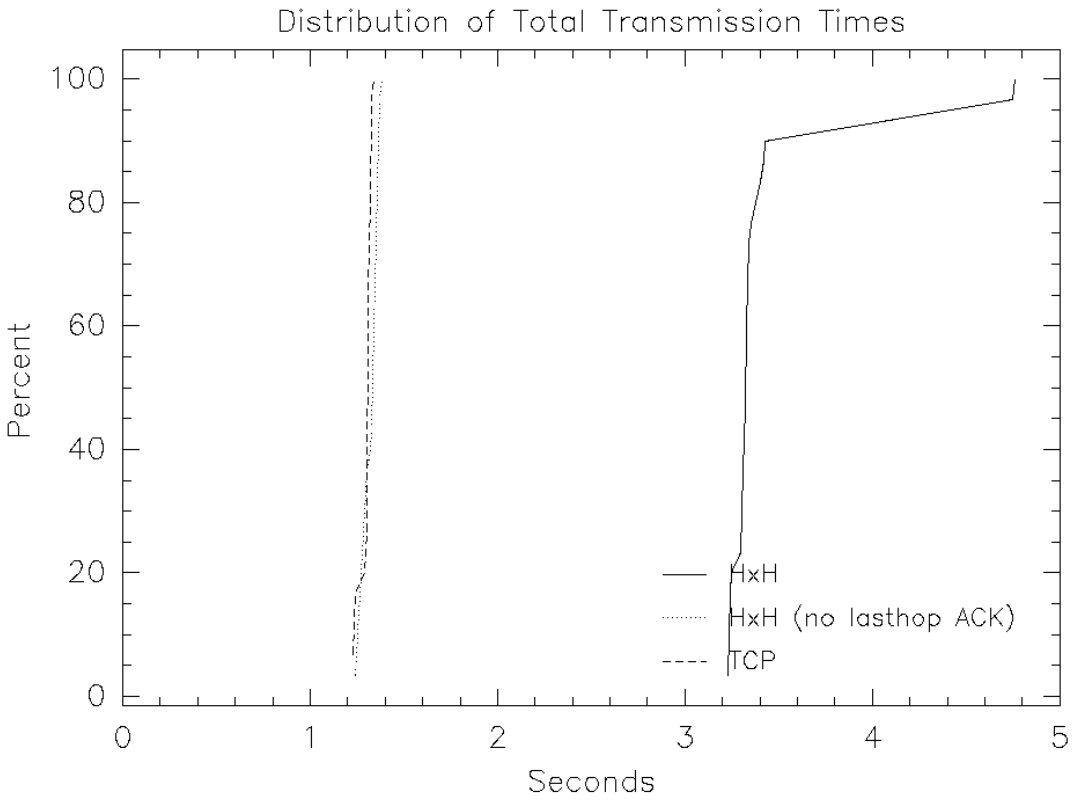

Figure 5.2: *Throughput from mesh12 to mesh11:* This graph shows the cumulative distribution function of the times taken to transfer one megabyte from mesh12 to mesh11 using TCP, HxH, and HxH without last hop acknowledgments.

neighbor but very rarely be able to overhear its downstream neighbor's transmissions, causing the HxH protocol to send at a very suboptimal rate.

The times for transferring between mesh11 and mesh12 are fairly consistent with each other as evidenced by the almost vertical lines for TCP in Figures 5.1 and 5.2. Some pairs of nodes, however, show more variability. When transferring from mesh6 to mesh3, for instance, all but one of the transmissions complete in under a second, but one of them takes almost four seconds. In most cases the variation is much smaller, but even small variations can build up over multiple hops.

5.1.2 Throughput on Multiple Hops

As the path length increases, the throughput of all of the protocols tested deteriorates quickly. In order to get paths longer than four hops using our mesh network we are required to disable the olsrd routing protocol in favor of predetermined static routes. In general, the performance of HxH deteriorates faster than the performance of HxH without the last hop acknowledgments, which deteriorates more quickly than the performance of TCP.

Figure 5.3 shows a cumulative distribution of the times taken for a transfer from mesh12 through mesh3 to mesh6. This relative performance is typical of the two-hop transfers tested in that TCP is clearly the fastest, with the HxH protocol without last hop acknowledgments coming in at roughly double the required time, and HxH being very slow. The slow performance of HxH can be explained by higher-than-expected packet loss which causes the protocol to resend many data packets at the last hop while waiting for them to be acknowledged.

At four hops the variability in the connections between each pair of nodes begins to be more pronounced. Figure 5.4 shows the times for a typical four-hop transfer. Most of the time TCP clearly outperforms HxH, but about 20 percent of the TCP transfers take longer to complete than any of the HxH transfers. It is also interesting to note that the difference between HxH and HxH without the last hop acknowledgments is significantly smaller than

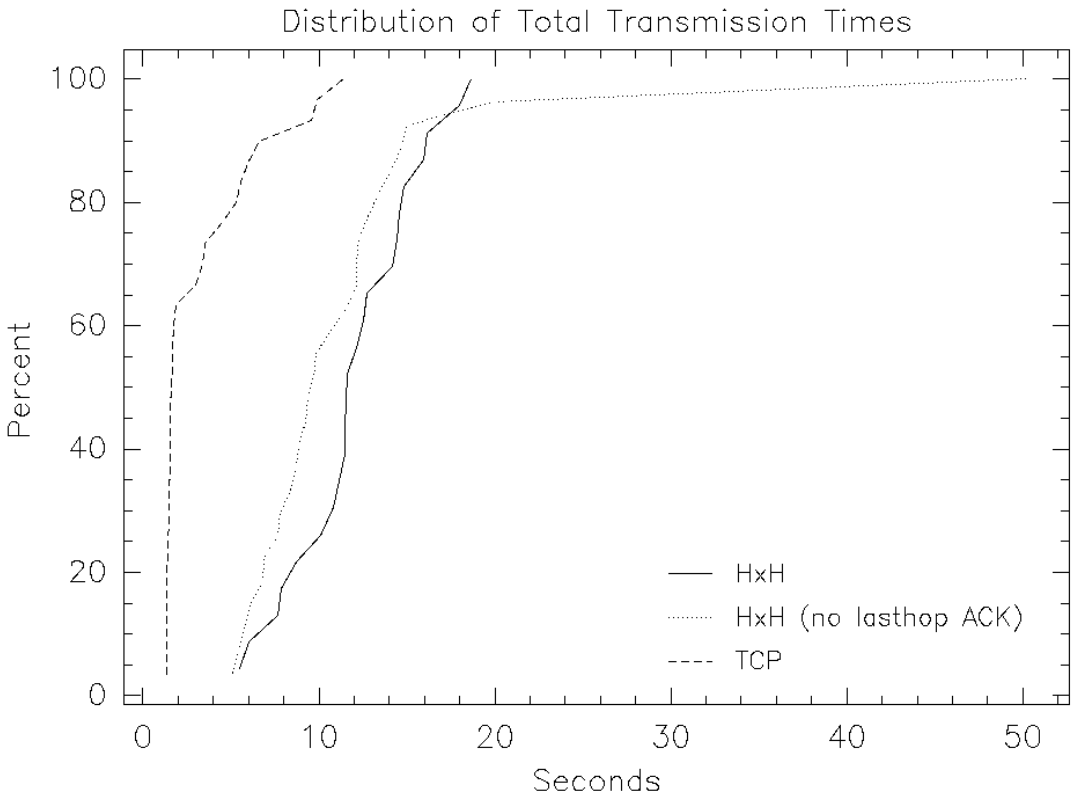

Figure 5.3: *Throughput from mesh12 through mesh3 to mesh6:* This graph shows the cumulative distribution function of the times taken to transfer one megabyte from mesh12 to mesh6 using TCP, HxH, and HxH without last hop acknowledgments.

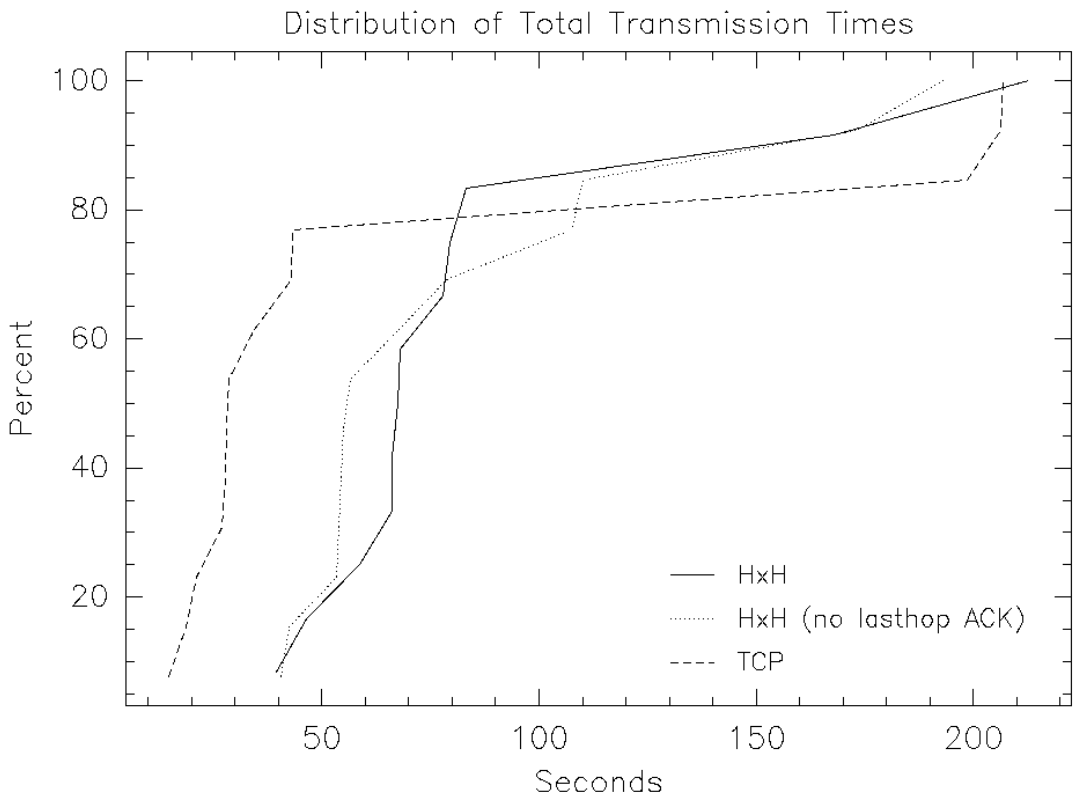

Figure 5.4: *Throughput from mesh6 through mesh5, mesh2, and mesh4 to mesh1:* This graph shows the cumulative distribution function of the times taken to transfer one megabyte from mesh5 to mesh1 using TCP, HxH, and HxH without last hop acknowledgments.

it was on shorter paths. This makes sense as the overhead of the last hop acknowledgment is a smaller percentage of the transfers required to get the data across more hops.

With even longer paths the same trends continue. Figure 5.5 shows the distribution of times for a nine-hop transfer. The difference between HxH and HxH without the lasthop acknowledgments continues to grow more slowly, and TCP still clearly outperforms both HxH protocols most of the time. We expected the throughput of HxH to degrade more slowly than that of TCP as the number of hops grew, but this is not what we observed.

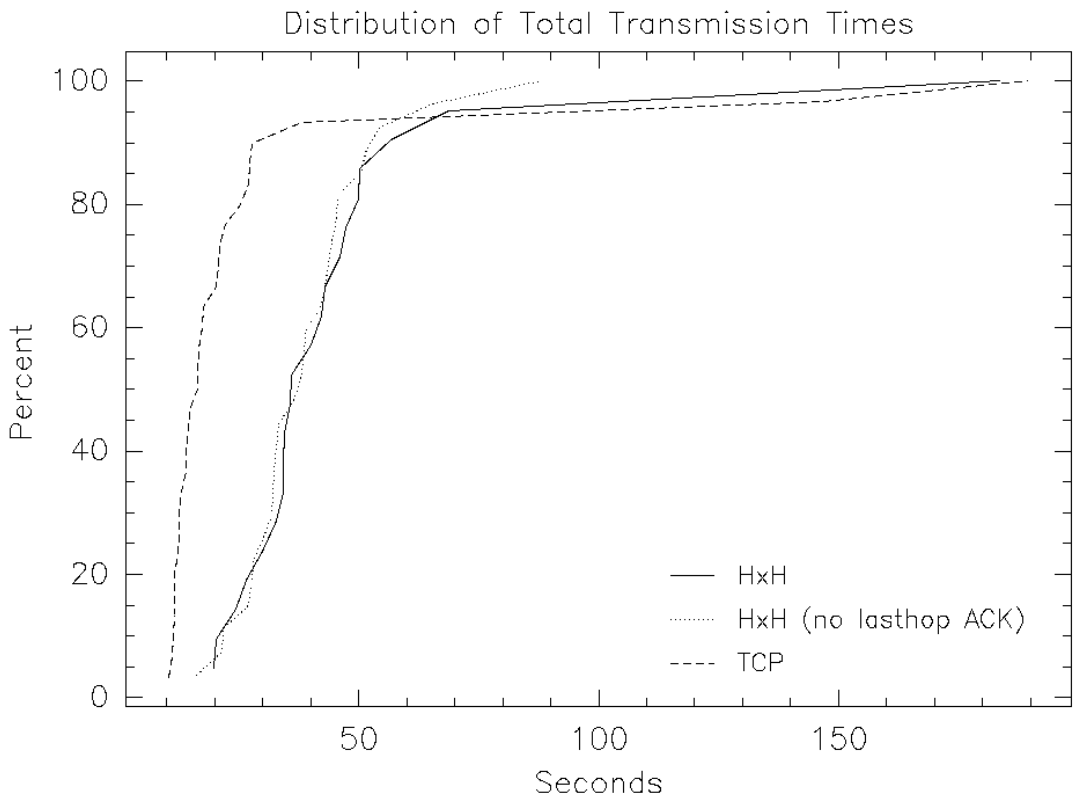

Figure 5.5: *Throughput from mesh8 through mesh9, mesh1, mesh4, mesh2, mesh5, mesh7, mesh6, and mesh3 to mesh13:* This graph shows the cumulative distribution function of the times taken to transfer one megabyte from mesh8 to mesh13 using TCP, HxH, and HxH without last hop acknowledgments.

5.1.3 Buffer Size

In simulations the HxH protocol uses a relatively small buffer size of 8 packets. This buffer size is used to control the transmission rate as a node will not send until it believes its downstream neighbor has buffer space for the packet it is about to send. Under the idealized radio transmission model the small buffer size allows for a more consistent transmission rate across all of the nodes in a long path. All of the results reported in the previous sections also use a buffer size of 8 packets, but in an effort to understand how the buffer size affects the throughput we rerun some of the longer paths with different buffer sizes. Figure 5.6, Figure 5.7, Figure 5.8, and Figure 5.9 show the distributions of times for a transfer over 8 hops with buffer sizes of 8, 128, 256, and 512 packets respectively. As the buffer size increases from 8 to 256 packets the performance of HxH improves significantly with respect to the performance of TCP. With a buffer size of 512 packets the performance degrades slightly again, but remains better than with a buffer size of 8 packets. The larger buffer size allows HxH to queue more packets at a time to libpcap, which in turn reduces the number of packets lost with each transmission. The transmissions become very bursty, but the overall throughput is increased.

5.1.4 RTS

The HxH protocol was primarily tested in the ns-2 simulator with the RTS/CTS exchange enabled. All of the trials reported on in the previous section are run without the RTS/CTS exchange, meaning that all packets are sent as broadcast packets. The HxH protocol is designed to work in an environment where delivery from one hop to the next is guaranteed by the RTS/CTS exchange, so we test this aspect of the protocol in our mesh network by testing the same transfer with RTS/CTS off, on for all packets, and on for packets larger than 500 bytes. Figure 5.8 shows the results with RTS/CTS disabled, Figure 5.10 shows the results when RTS is enabled for all packets, and Figure 5.11 shows the results with RTS

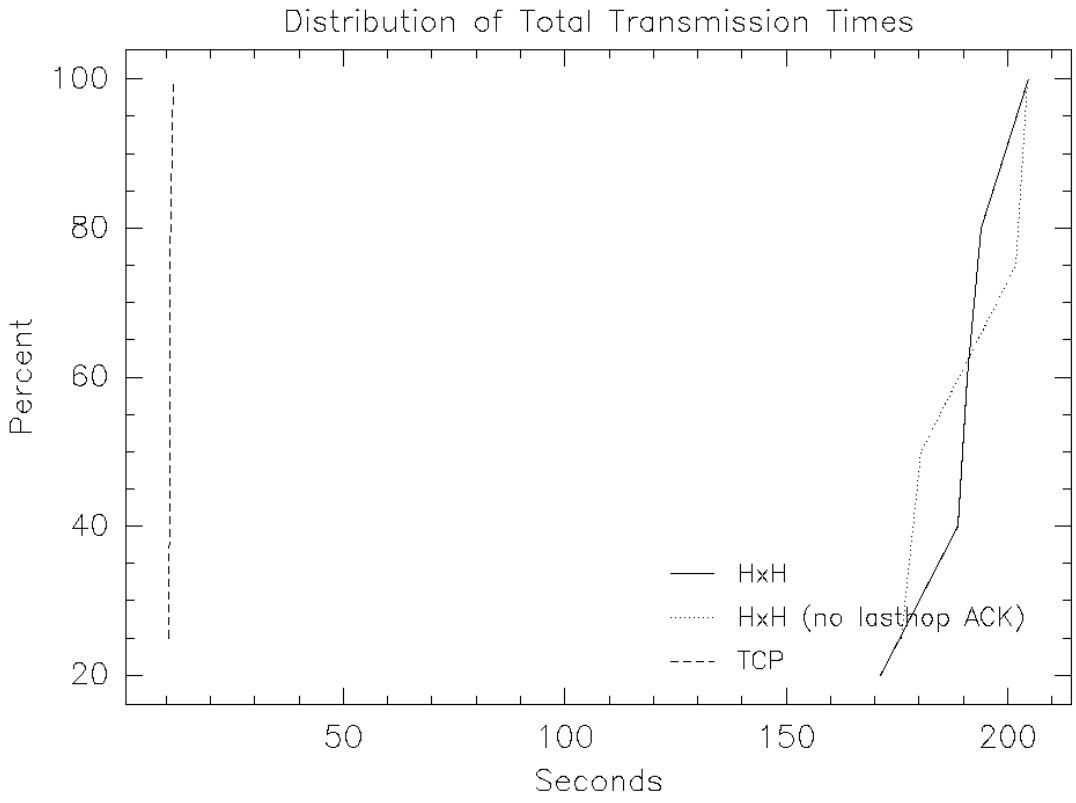

Figure 5.6: *Throughput from mesh8 through mesh9, mesh1, mesh4, mesh2, mesh5, mesh7, and mesh6 to mesh3:* This graph shows the cumulative distribution function of the times taken to transfer one megabyte from mesh8 to mesh13 using TCP, HxH, and HxH without last hop acknowledgments using a buffer size of 8 packets.

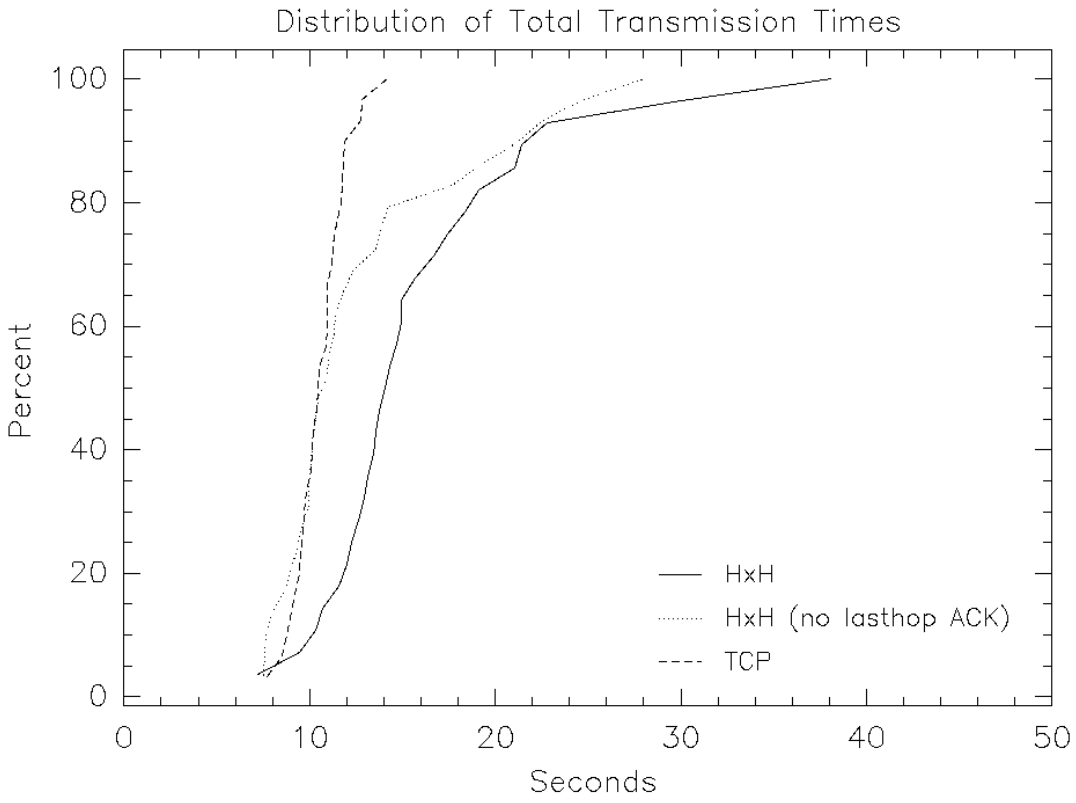

Figure 5.7: *Throughput from mesh8 through mesh9, mesh1, mesh4, mesh2, mesh5, mesh7, and mesh6 to mesh3:* This graph shows the cumulative distribution function of the times taken to transfer one megabyte from mesh8 to mesh13 using TCP, HxH, and HxH without last hop acknowledgments using a buffer size of 128 packets.

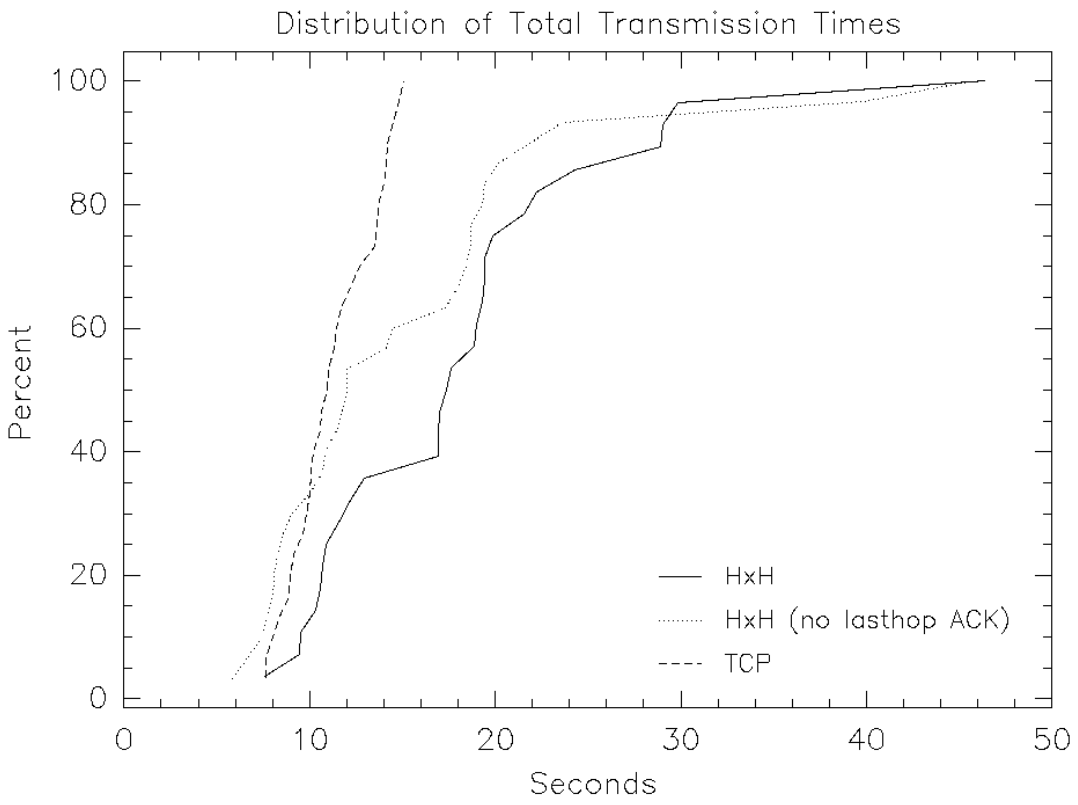

Figure 5.8: *Throughput from mesh8 through mesh9, mesh1, mesh4, mesh2, mesh5, mesh7, and mesh6 to mesh3:* This graph shows the cumulative distribution function of the times taken to transfer one megabyte from mesh8 to mesh13 using TCP, HxH, and HxH without last hop acknowledgments using a buffer size of 256 packets.

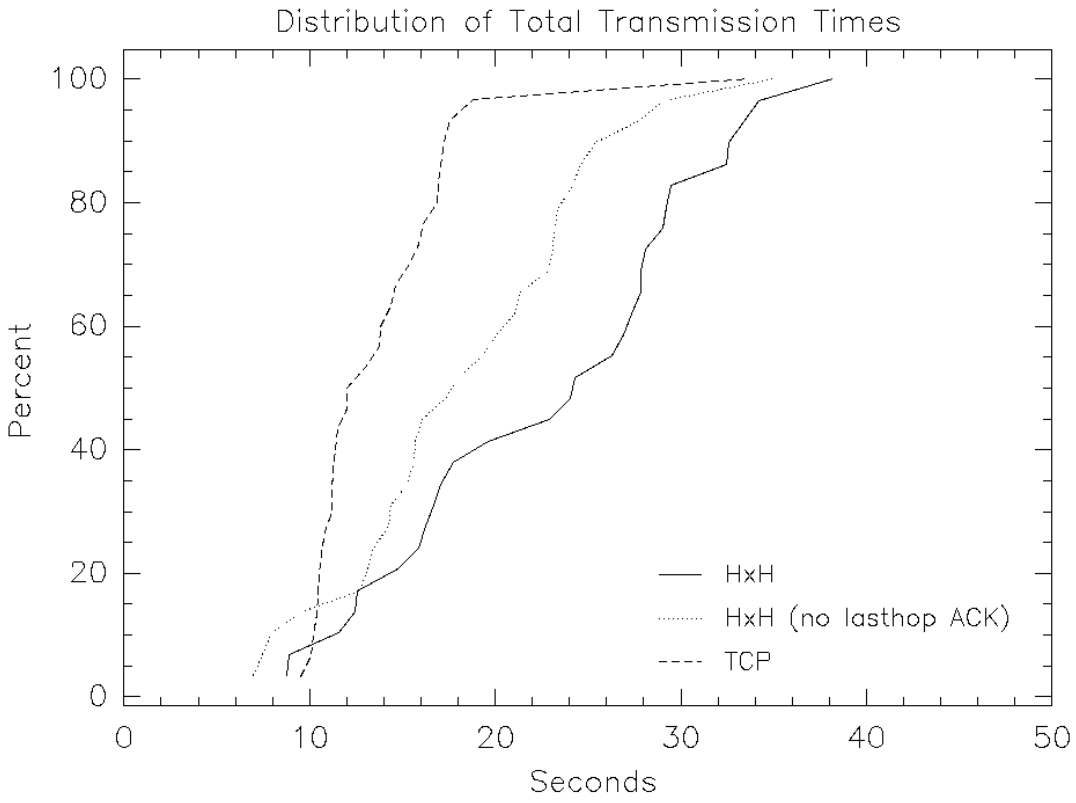

Figure 5.9: *Throughput from mesh8 through mesh9, mesh1, mesh4, mesh2, mesh5, mesh7, and mesh6 to mesh3:* This graph shows the cumulative distribution function of the times taken to transfer one megabyte from mesh8 to mesh13 using TCP, HxH, and HxH without last hop acknowledgments using a buffer size of 512 packets.

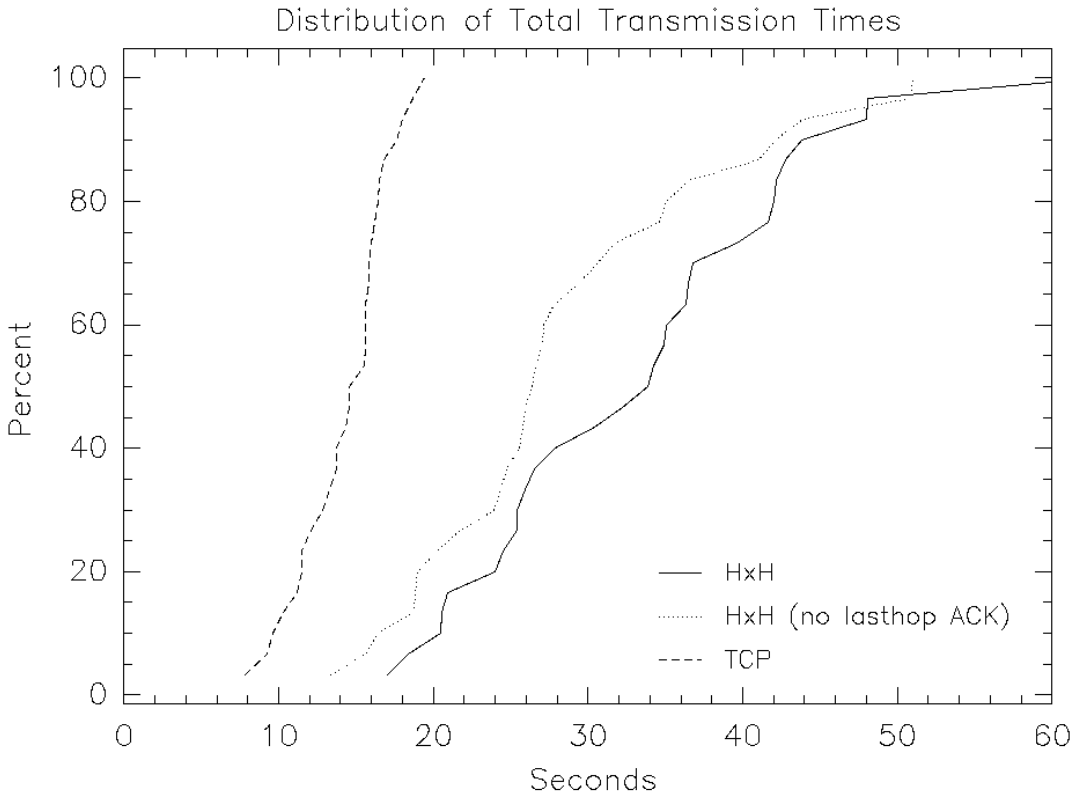

Figure 5.10: *Throughput from mesh8 through mesh9, mesh1, mesh4, mesh2, mesh5, mesh7, and mesh6 to mesh3:* This graph shows the cumulative distribution function of the times taken to transfer one megabyte from mesh8 to mesh13 using TCP, HxH, and HxH without last hop acknowledgments using a buffer size of 256 packets and the RTS/CTS exchange turned on.

enabled for packets larger than 500 bytes. The performance degrades for all protocols when RTS is turned on, but the degradation is more pronounced for both HxH variations.

5.2 Round Trip Times

In the HxH protocol reliability is based primarily on passive acknowledgments. When the passive mechanism fails, the protocol falls back on explicit acknowledgments. In order to understand how effectively this passive feedback is propagating back to the source from the destination, we measure the round trip time, which we define as the interval between the

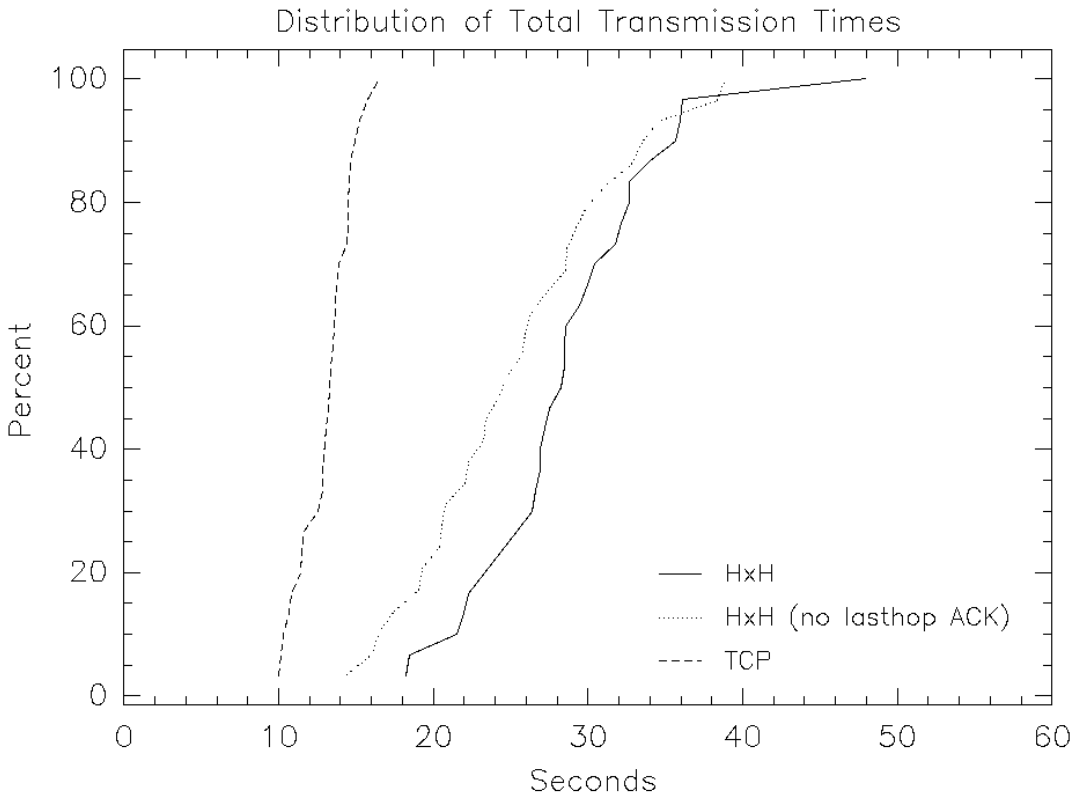

Figure 5.11: *Throughput from mesh8 through mesh9, mesh1, mesh4, mesh2, mesh5, mesh7, and mesh6 to mesh3:* This graph shows the cumulative distribution function of the times taken to transfer one megabyte from mesh8 to mesh13 using TCP, HxH, and HxH without last hop acknowledgments using a buffer size of 256 packets and the RTS/CTS exchange turned on for packets larger than 500 bytes.

transmission of a packet at the source node and the receipt of a passive acknowledgment with equal or greater sequence number. This measurement effectively quantifies how long the sender must keep a packet available for potential retransmissions. The more frequently the passive acknowledgments are heard by the upstream nodes the more consistent the round trip times are.

Figure 5.12 shows a cumulative distribution function for the round trip times as defined above. It is clear that the version without the last hop acknowledgments receives feedback at the source node in a much more timely way than the original version. This makes sense because the version without the last hop acknowledgment is reporting that a packet got through to the destination when the second-to-last node transmits the packet, not when the second-to-last node receives the ACK from the destination. Consequently the round trip times of the version without last hop acknowledgments is more an approximation of the kind of performance we would expect from an implementation of HxH that does not have to resort to the last hop acknowledgment than an exact measurement of the network.

As with the throughput the performance degrades as the length of the path increases. In order to understand the effect of buffer sizes on round trip times we run trials using buffer sizes of 8, 128, 256, and 512 packets and measure the round trip times. Figures 5.13, 5.14, 5.15, and 5.16 show the results of these trials. Increasing the buffer size beyond 8 packets clearly decreases the round trip times. The round trip times decrease significantly at 128 and again at 256 packets, then increase again for 512 packets. Again a buffer size of 256 packets seems to be the best choice. It is not surprising that the round trip times decrease under the same conditions that cause throughput to increase.

Enabling the RTS/CTS exchange negatively affects throughput, but with the right settings appears to have a very positive effect on the round trip times. This makes sense as it means that fewer passive acknowledgments get lost which in turn increases the responsiveness of the HxH protocol. Figure 5.17 shows the round trip times when the RTS/CTS exchange is enabled for all packets, and figure 5.18 shows the times when the RTS/CTS exchange is

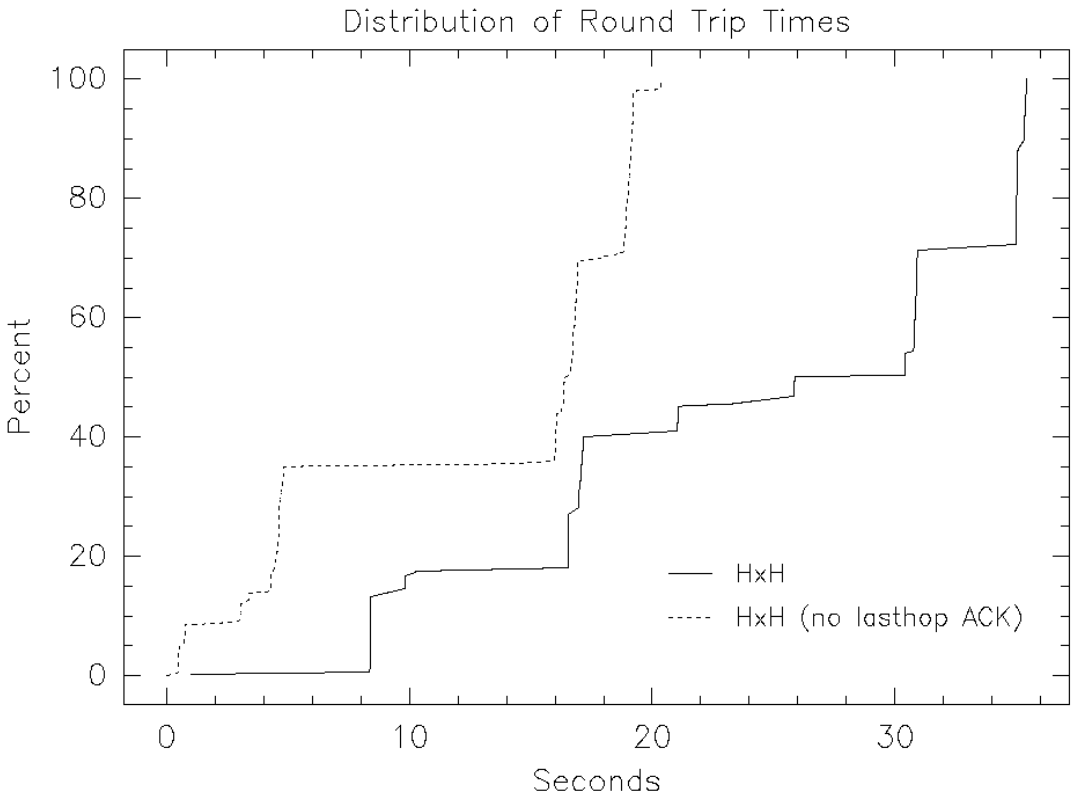

Figure 5.12: *Round trip times from mesh6 through mesh5, mesh2, and mesh4 to mesh1:* This graph shows the cumulative distribution function of the intervals between the transmission of a packet from mesh6 to mesh1 and the receipt of a corresponding passive acknowledgment using HxH and HxH without last hop acknowledgments.

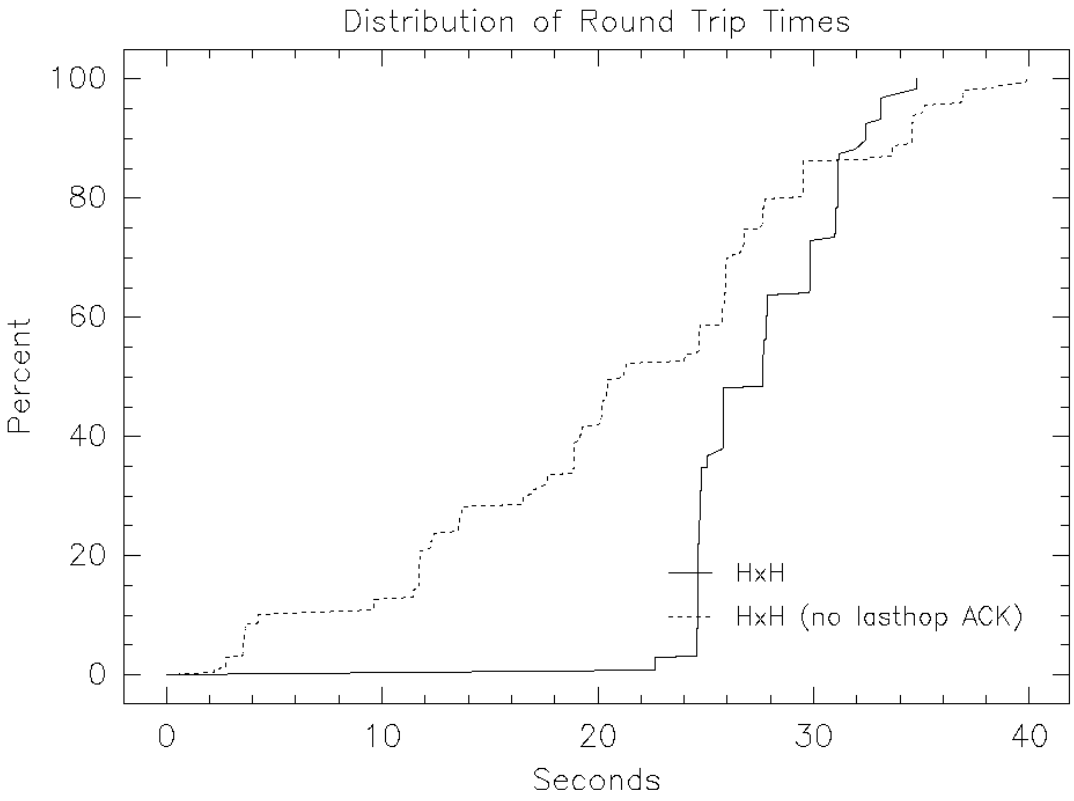

Figure 5.13: *Round trip times from mesh8 through mesh9, mesh1, mesh4, mesh2, mesh5, mesh7, and mesh6 to mesh3:* This graph shows the cumulative distribution function of the intervals between the transmission of a packet from mesh8 to mesh3 and the receipt of a corresponding passive acknowledgment using HxH and HxH without last hop acknowledgments with a buffer size of 8 packets.

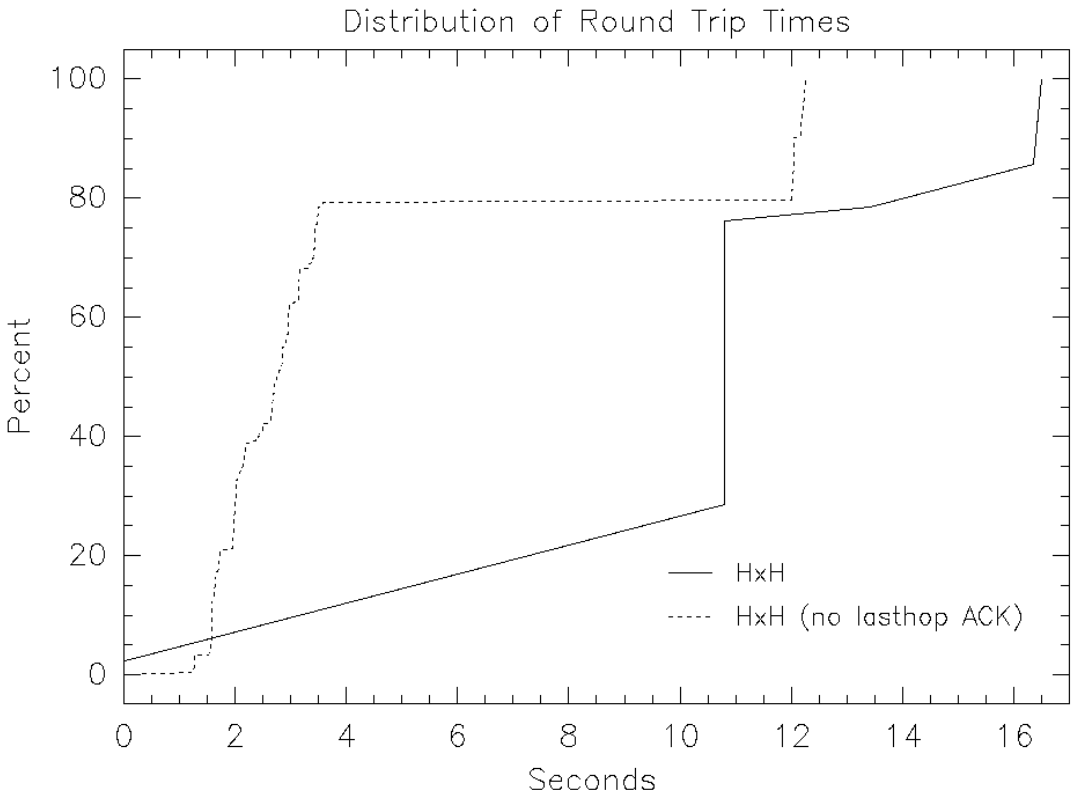

Figure 5.14: *Round trip times from mesh8 through mesh9, mesh1, mesh4, mesh2, mesh5, mesh7, and mesh6 to mesh3:* This graph shows the cumulative distribution function of the intervals between the transmission of a packet from mesh8 to mesh3 and the receipt of a corresponding passive acknowledgment using HxH and HxH without last hop acknowledgments with a buffer size of 128 packets.

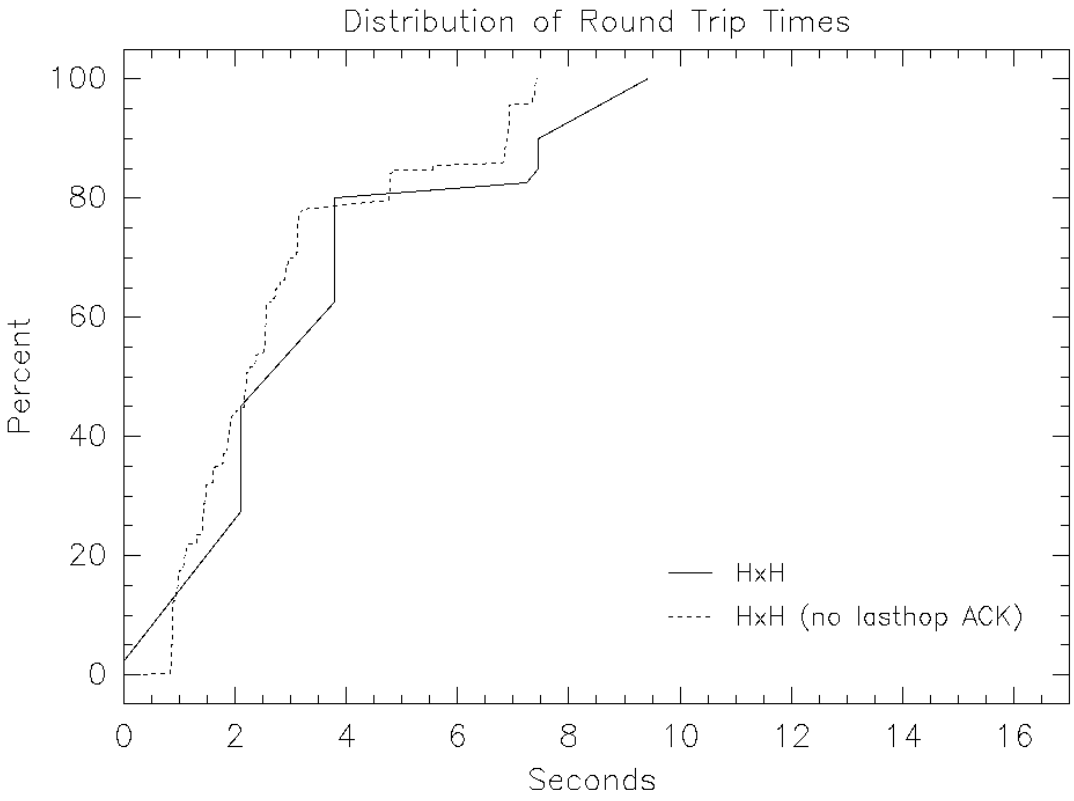

Figure 5.15: *Round trip times from mesh8 through mesh9, mesh1, mesh4, mesh2, mesh5, mesh7, and mesh6 to mesh3:* This graph shows the cumulative distribution function of the intervals between the transmission of a packet from mesh8 to mesh3 and the receipt of a corresponding passive acknowledgment using HxH and HxH without last hop acknowledgments with a buffer size of 256 packets.

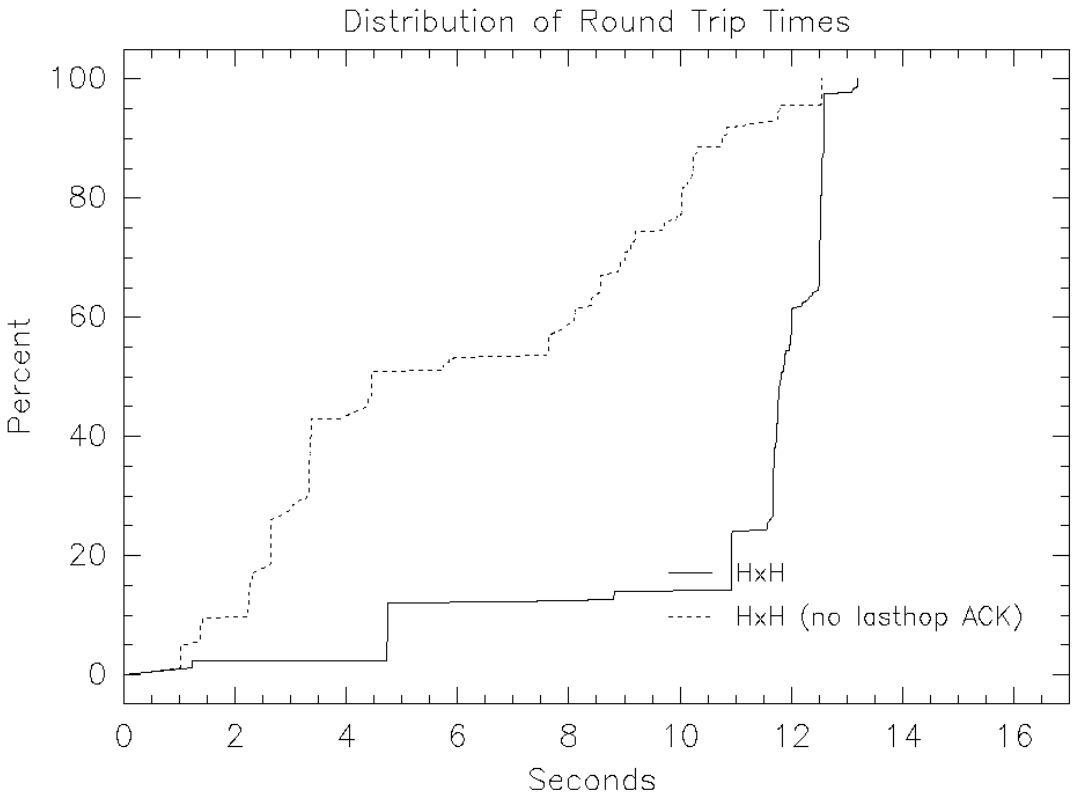

Figure 5.16: *Round trip times from mesh8 through mesh9, mesh1, mesh4, mesh2, mesh5, mesh7, and mesh6 to mesh3:* This graph shows the cumulative distribution function of the intervals between the transmission of a packet from mesh8 to mesh3 and the receipt of a corresponding passive acknowledgment using HxH and HxH without last hop acknowledgments with a buffer size of 512 packets.

enabled for packets larger than 500 bytes. With the RTS/CTS exchange always enabled the performance of both versions of HxH is similar, and it is not as good as when the exchange is disabled. With the exchange only enabled for packets of 500 bytes or larger, however, an interesting thing happens. The version of HxH without last hop acknowledgments has extremely low round trip times and the other version has much higher times. It makes sense that enabling the RTS/CTS exchange for data packets would allow the MAC layer to prevent a node from transmitting when its downstream neighbor is transmitting which would cause fewer passive acknowledgments to be lost. It also makes sense that with fewer passive acknowledgments lost the round trip times would decrease.

The reason that the standard HxH implementation had such high round trip times with this setting is that the RTS/CTS exchange protects the data packets from the second-to-last node to the destination, but does not protect the last hop ACKs being sent from the destination. Whenever this last hop ACK is lost the second-to-last node retransmits the packet and waits again. This asymmetry in the MAC layer access control is responsible for the significant growth in round trip times.

5.3 Rate of Packet Loss

The HxH protocol was designed to take advantage of the MAC layer reliability built into the RTS/CTS exchange. It does not send its own acknowledgments, but assumes that once the packet has been transmitted that the next hop must have received it. By using the MAC layer reliability the HxH protocol can reduce its own reliability overhead. Because our implementation of HxH is not able to wait for the MAC layer ACK it cannot know that the next hop actually received the packet. We measure the rate of packet loss at each pair of nodes in order to understand how significant a role packet loss plays in disrupting the HxH protocol implementation. We define the rate of packet loss as the percentage of packets sent at a given node that were never received by the next hop.

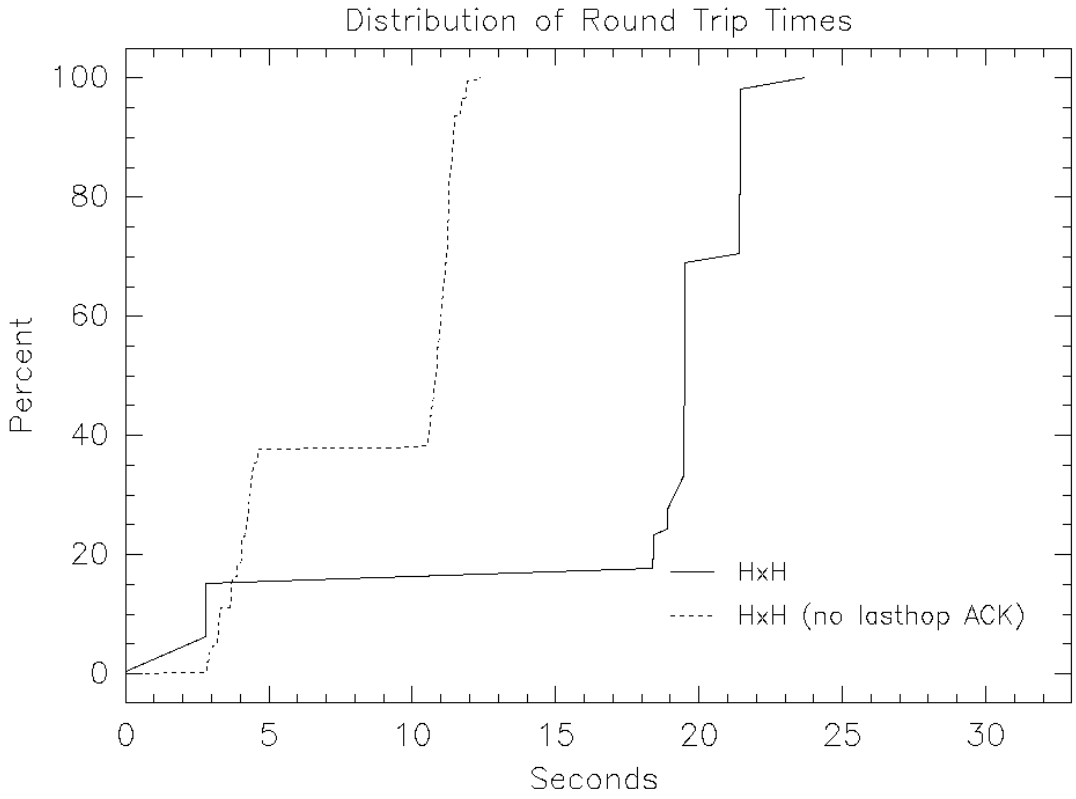

Figure 5.17: *Round trip times from mesh8 through mesh9, mesh1, mesh4, mesh2, mesh5, mesh7, and mesh6 to mesh3:* This graph shows the cumulative distribution function of the intervals between the transmission of a packet from mesh8 to mesh3 and the receipt of a corresponding passive acknowledgment using HxH and HxH without last hop acknowledgments with a buffer size of 256 packets and the RTS/CTS exchanged enabled for all packets.

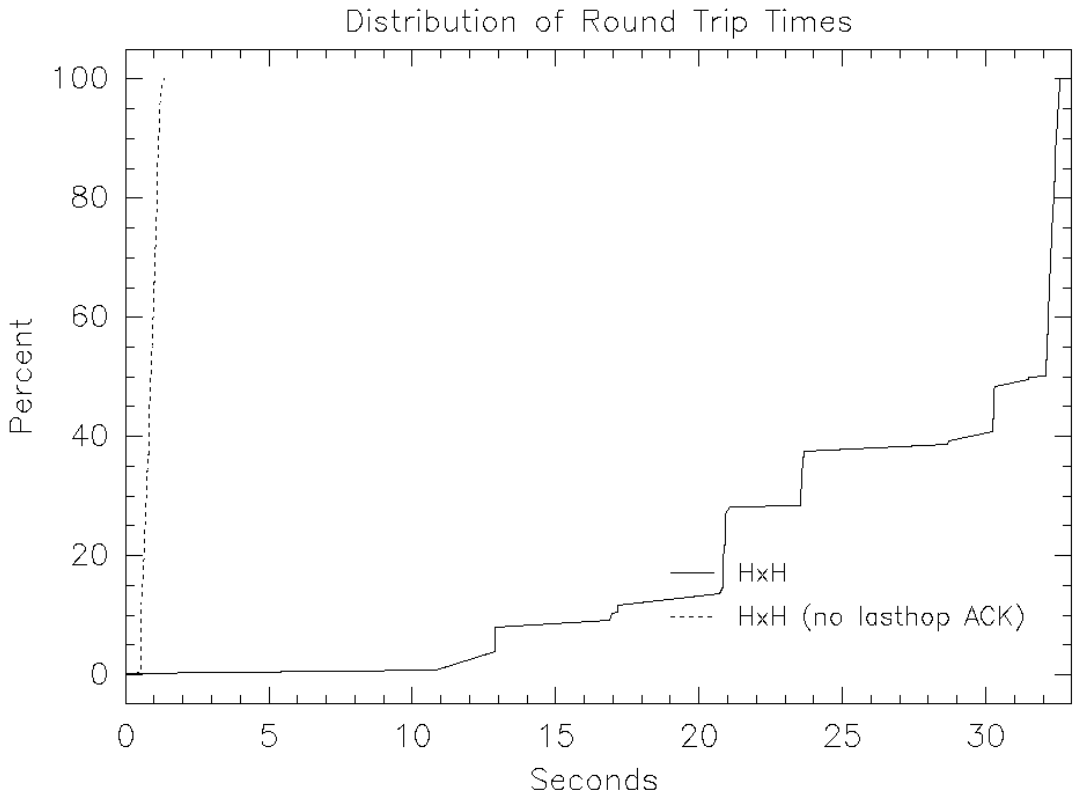

Figure 5.18: *Round trip times from mesh8 through mesh9, mesh1, mesh4, mesh2, mesh5, mesh7, and mesh6 to mesh3:* This graph shows the cumulative distribution function of the intervals between the transmission of a packet from mesh8 to mesh3 and the receipt of a corresponding passive acknowledgment using HxH and HxH without last hop acknowledgments with a buffer size of 256 packets and the RTS/CTS exchanged enabled for packets larger than 500 bytes.

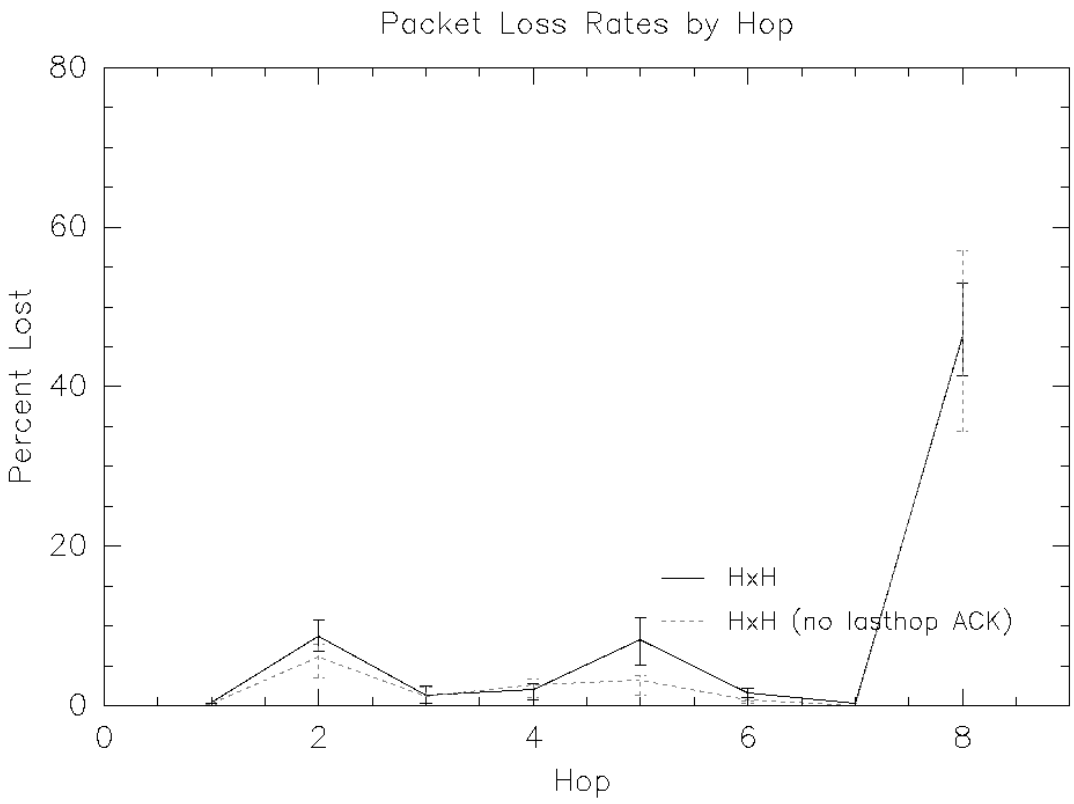

Figure 5.19: *Packet Loss Rates for transmissions from mesh8 to mesh9, mesh9 to mesh1, mesh1 to mesh4, mesh4 to mesh2, mesh2 to mesh5, mesh5 to mesh7, mesh7 to mesh6, and mesh6 to mesh3:* This graph shows the rates of packet loss at each hop of a transmission from mesh8 to mesh3 using HxH and HxH without last hop acknowledgments with a buffer size of 128 packets.

The amount of variability in the mesh network itself makes it difficult to draw any strong conclusions from these observations, but in general it appears that the rate of packet loss has little to do with the number of hops the data traverses or whether or not last hop acknowledgments are being used. Figure 5.19 shows a typical result, where the loss rates on most of the hops is negligible but there are one or two problematic hops with high loss rates. In general the rate of loss is high enough to cause trouble for the HxH protocol implementation.

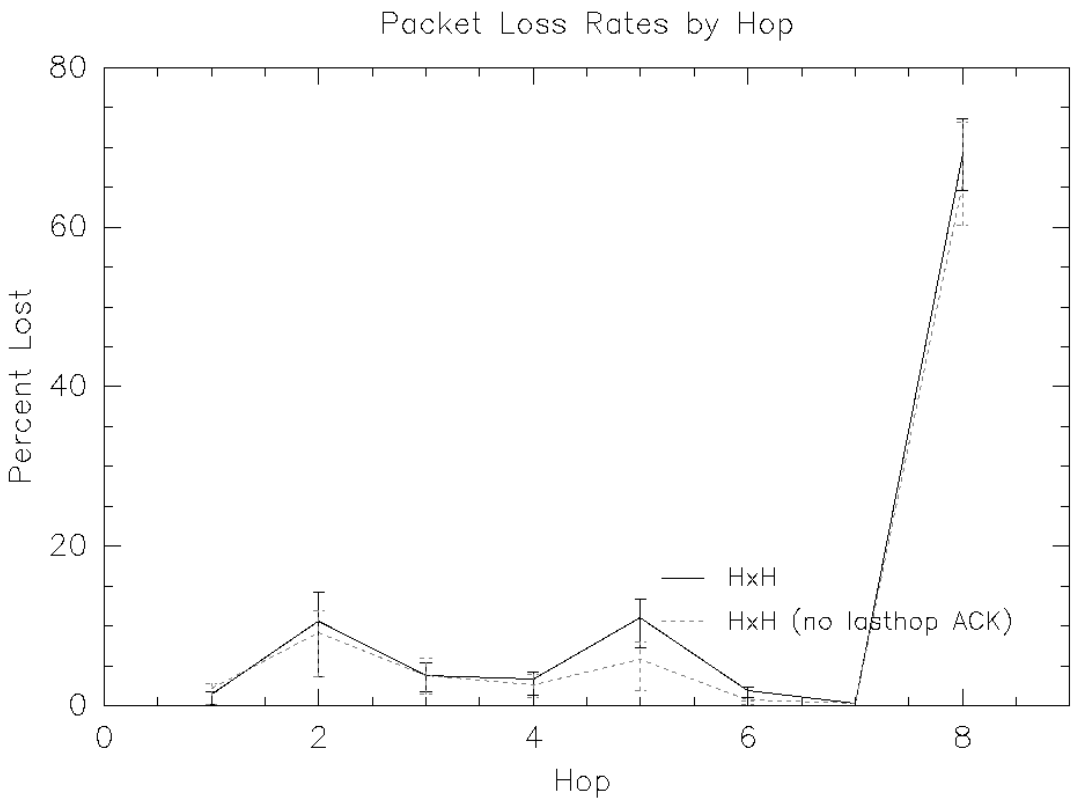

Figure 5.20: *Packet Loss Rates for transmissions from mesh8 to mesh9, mesh9 to mesh1, mesh1 to mesh4, mesh4 to mesh2, mesh2 to mesh5, mesh5 to mesh7, mesh7 to mesh6, and mesh6 to mesh3:* This graph shows the rates of packet loss at each hop of a transmission from mesh8 to mesh3 using HxH and HxH without last hop acknowledgments with a buffer size of 256 packets.

Increasing the buffer size had no consistent effect on the packet loss rates, but turning on the RTS/CTS exchange did appear to increase the rate of packet loss. Figure 5.20 shows the loss rates for a transfer with the RTS/CTS exchange disabled, figure 5.21 shows the loss rates with RTS/CTS always enabled, and figure 5.22 shows the rates when RTS/CTS is enabled for packets larger than 500 bytes. The second hop (mesh9 to mesh1) and fifth hop (mesh2 to mesh5) are clearly more problematic with the RTS/CTS exchange enabled, most likely because the signal doesn't propagate as well in the reverse direction as it does in the forward direction.

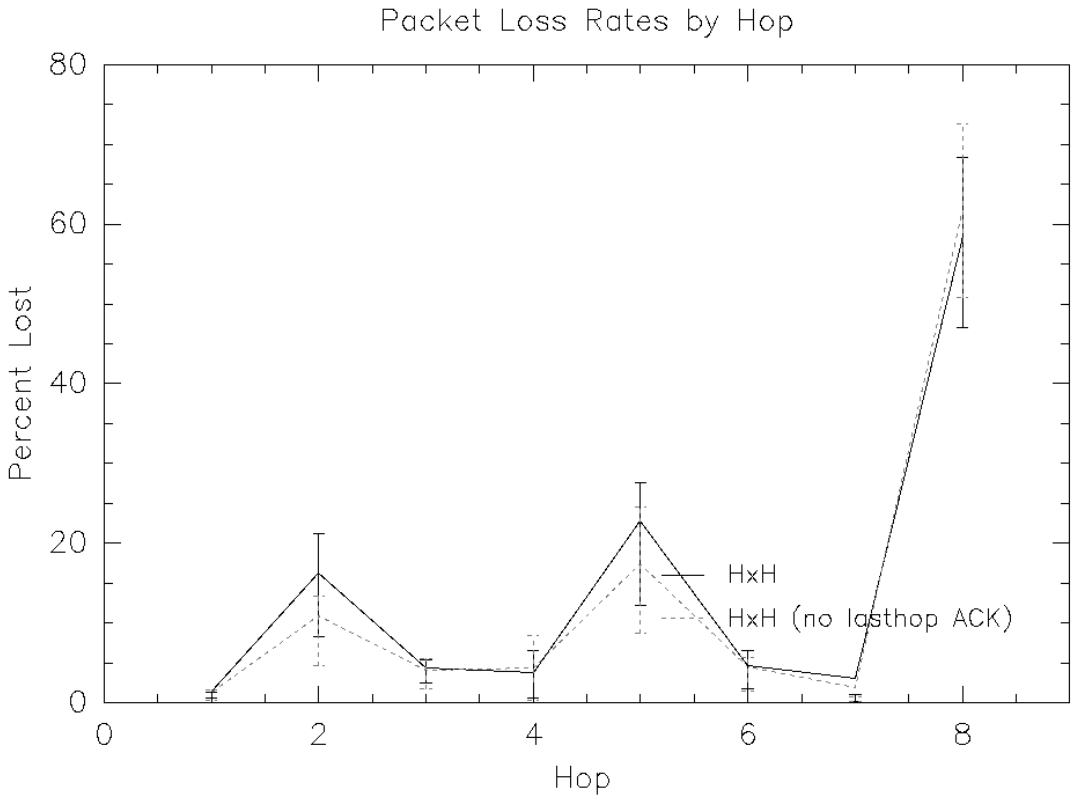

Figure 5.21: *Packet Loss Rates for transmissions from mesh8 to mesh9, mesh9 to mesh1, mesh1 to mesh4, mesh4 to mesh2, mesh2 to mesh5, mesh5 to mesh7, mesh7 to mesh6, and mesh6 to mesh3:* This graph shows the rates of packet loss at each hop of a transmission from mesh8 to mesh3 using HxH and HxH without last hop acknowledgments with a buffer size of 256 packets and the RTS/CTS exchange enabled for all packets.

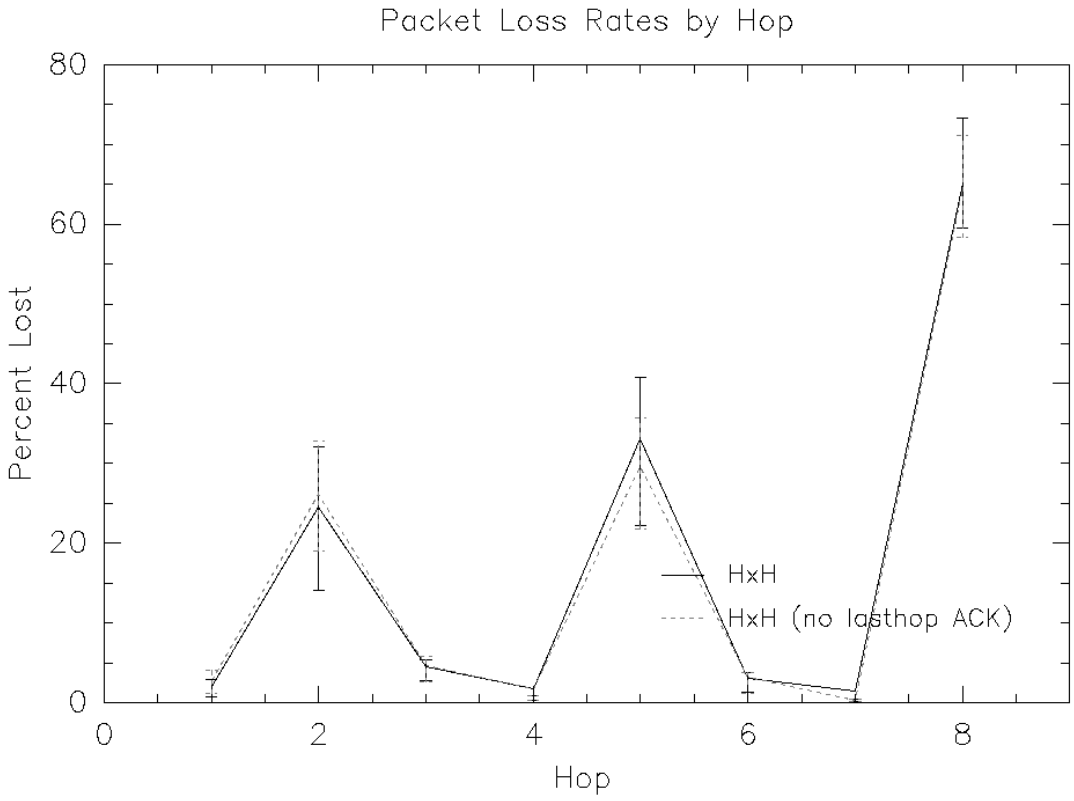

Figure 5.22: *Packet Loss Rates for transmissions from mesh8 to mesh9, mesh9 to mesh1, mesh1 to mesh4, mesh4 to mesh2, mesh2 to mesh5, mesh5 to mesh7, mesh7 to mesh6, and mesh6 to mesh3:* This graph shows the rates of packet loss at each hop of a transmission from mesh8 to mesh3 using HxH and HxH without last hop acknowledgments with a buffer size of 256 packets and the RTS/CTS exchange enabled for packets larger than 500 bytes.

5.4 Effectiveness of Passive Acknowledgments

Another way that HxH reduces its own overhead is by primarily utilizing passive acknowledgments. In simulations this allows the protocol to maintain a fairly constant transmission rate and report reliability information in a timely manner to the source node without using extra bandwidth. In our mesh network, however, many passive acknowledgments are never heard by the upstream nodes, which negatively affects the protocol's performance. We measure the passive acknowledgment loss rate by computing the percentage of times a node sends a packet to its downstream neighbor and its upstream neighbor does not overhear it. We do not measure a passive acknowledgment loss rate for the last hop as the destination does not send passive acknowledgments. The passive acknowledgment loss rate is very high overall, with several hops regularly missing more than 50 percent. As with the overall packet loss rate the rate of passive acknowledgment loss appears to be independent of the number of hops the data traverses and the buffer size. The RTS/CTS exchange also appears to have no impact on the passive acknowledgment loss rate. This makes sense as the RTS/CTS exchange protects against transmissions from other nodes in the vicinity of the transmitting node but does nothing to prevent transmissions from nodes that do not hear the RTS. Figure 5.23 shows a typical graph of the passive acknowledgment loss rates at each hop of a set of 30 transfers.

5.5 Transmission Rates

The transmission rate in HxH is controlled by the credits mechanism, which relies on a node being able to overhear the transmissions of its downstream neighbor. The credits and passive acknowledgment values in the packet headers are updated at each hop, and the upstream nodes update their own state accordingly. In simulations, where packet loss is minimal, the HxH protocol is able to use this information to maintain a nearly constant transmission rate and to nearly eliminate packet loss due to congestion. Our measurements show that packet

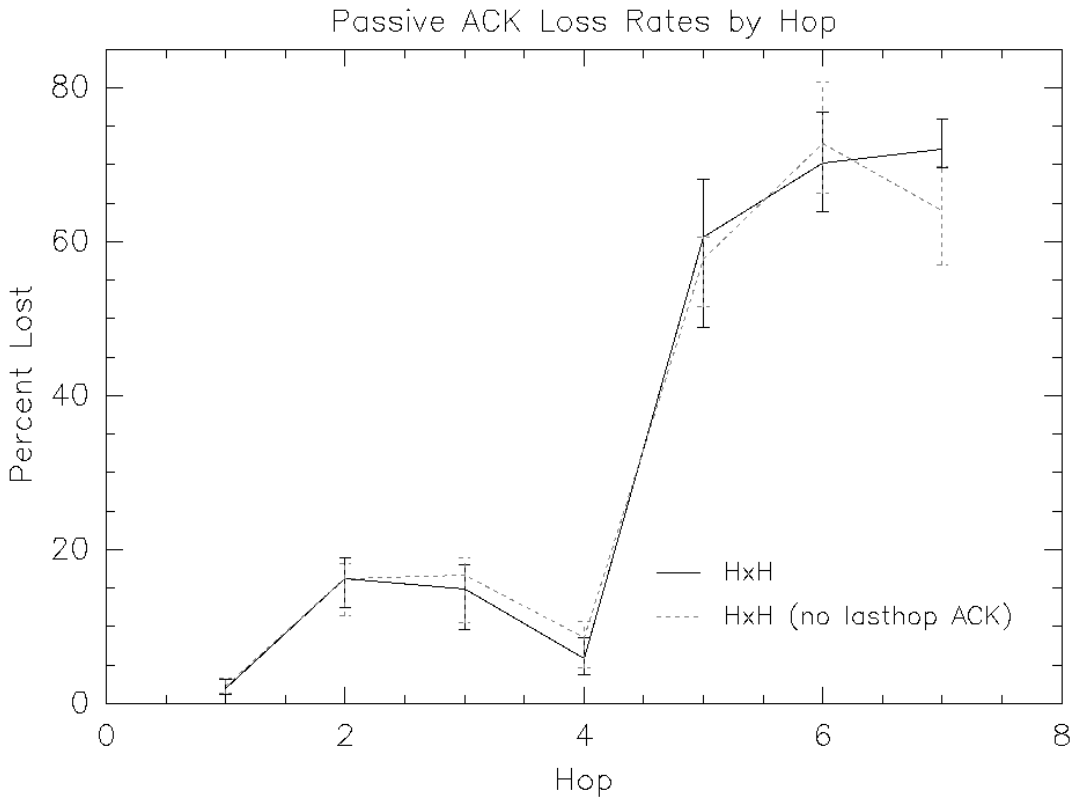

Figure 5.23: *Passive Acknowledgment Loss Rates for transmissions from mesh8 to mesh9, mesh9 to mesh1, mesh1 to mesh4, mesh4 to mesh2, mesh2 to mesh5, mesh5 to mesh7, mesh7 to mesh6, and mesh6 to mesh3:* This graph shows the rates of passive acknowledgment loss at each hop of a transmission from mesh8 to mesh3 using HxH and HxH without last hop acknowledgments with a buffer size of 256 packets.

loss for both data and passive acknowledgment packets is much higher in our mesh network than it is in the simulations. This higher packet loss renders the credit-based congestion control less-effective and means that the transmission rate is lower and much less constant than in the simulator.

Figure 5.24 shows a graph of the transmission rates at mesh8 over time for a transmission from mesh8 through mesh9, mesh1, mesh4, mesh2, mesh5, mesh7, and mesh6 to mesh3. Graphs of the transmission rates at the other nodes are not included as they are very similar to the graph of rates at mesh8. The transmission rate itself is rather erratic, ranging repeatedly from zero to about 300 Kbps. Analysis of the log files for this transmission show that many passive acknowledgments are being lost, which causes the protocol to frequently enter a timeout state as it believes the downstream node's buffers are full. This trial was run with a buffer size of eight packets. Only half of the buffer is routinely used, so this means that any time four subsequent passive acknowledgments are lost the protocol enters a timeout state before sending another packet. It takes a relatively long time to transfer one megabyte using HxH with a buffer size of 8 packets, so it is not surprising to find that the transmission rate is low. With a buffer size of 8 packets HxH is not distinguishable from HxH without last hop acknowledgments.

As the buffer size increases, the number of consecutive passive acknowledgments that must be lost in order to cause HxH to stall increases as well. With a buffer size of 128 packets the protocol does not completely stall until 64 consecutive passive acknowledgments are lost. Overall the transmission rates are higher and more consistent with this buffer size. Figures 5.25 and 5.26 show the transmission rates of HxH with lasthop acknowledgments at mesh8 and mesh6 respectively for a transmission from mesh8 to mesh3 using this new buffer size. These graphs are typical of the transfers that completed quickly in that the transmission rate starts out at about 2 Mbps and decreases slowly until all of the data has been sent. It is a reflection of the amount of packet loss that the rates for mesh6 are significantly lower than those for mesh8. At about 6 seconds into this transfer the destination node is asked for an

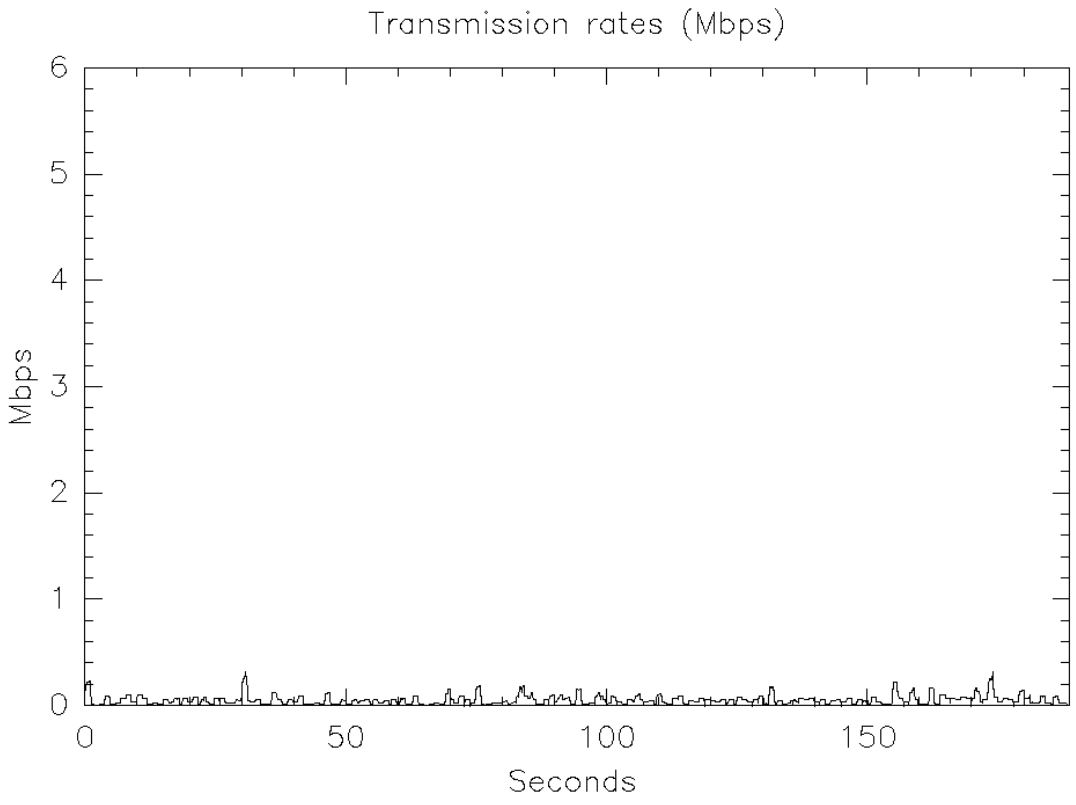

Figure 5.24: *Transmission Rates at mesh8 over time for a transmission from mesh8 through mesh9, mesh1, mesh4, mesh2, mesh5, mesh7, and mesh6 to mesh3:* This graph shows the amount of data transmitted per second using a one-second sliding window plotted every 100ms. The transfer uses last hop acknowledgments and a buffer size of 8 packets.

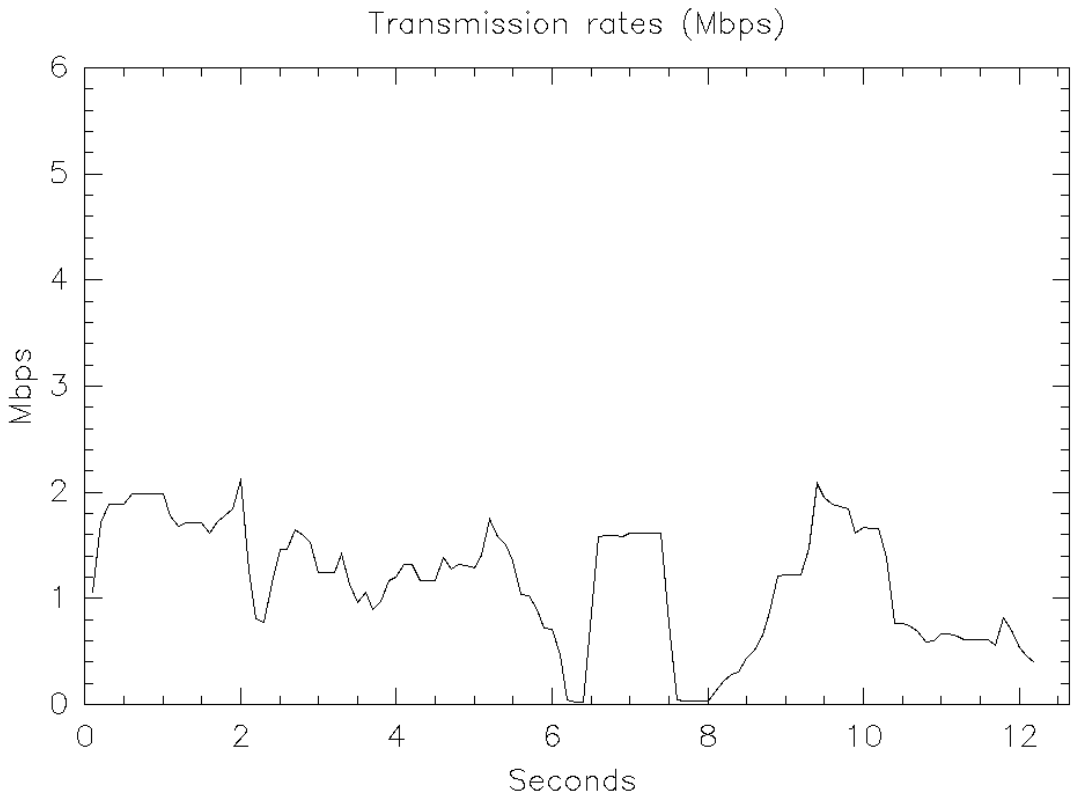

Figure 5.25: *Transmission Rates at mesh8 over time for a transmission from mesh8 through mesh9, mesh1, mesh4, mesh2, mesh5, mesh7, and mesh6 to mesh3:* This graph shows the amount of data transmitted per second using a one-second sliding window plotted every 100ms. The transfer uses last hop acknowledgments and a buffer size of 128 packets.

ACK. Upon receiving the ACK with its list of missing packets the source node immediately sends a number of packets that were missed and requests another ACK, but most of the packets are lost over the first three hops and never reach mesh6. At about 8 seconds the next ACK sent by the destination arrives at the source node and the transmissions continue. This time not nearly so many packets are lost and the transfer completes.

The transmission rate graphs for trials of HxH without lasthop acknowledgments are similar to Figures 5.25 and 5.26, but they tend to have fewer occasions in which the transfer stalls. Figures 5.27 and 5.28 show typical transfers for HxH without last hop acknowledgments. There is still a significant dip in transmissions between 4 and 5 seconds, but the

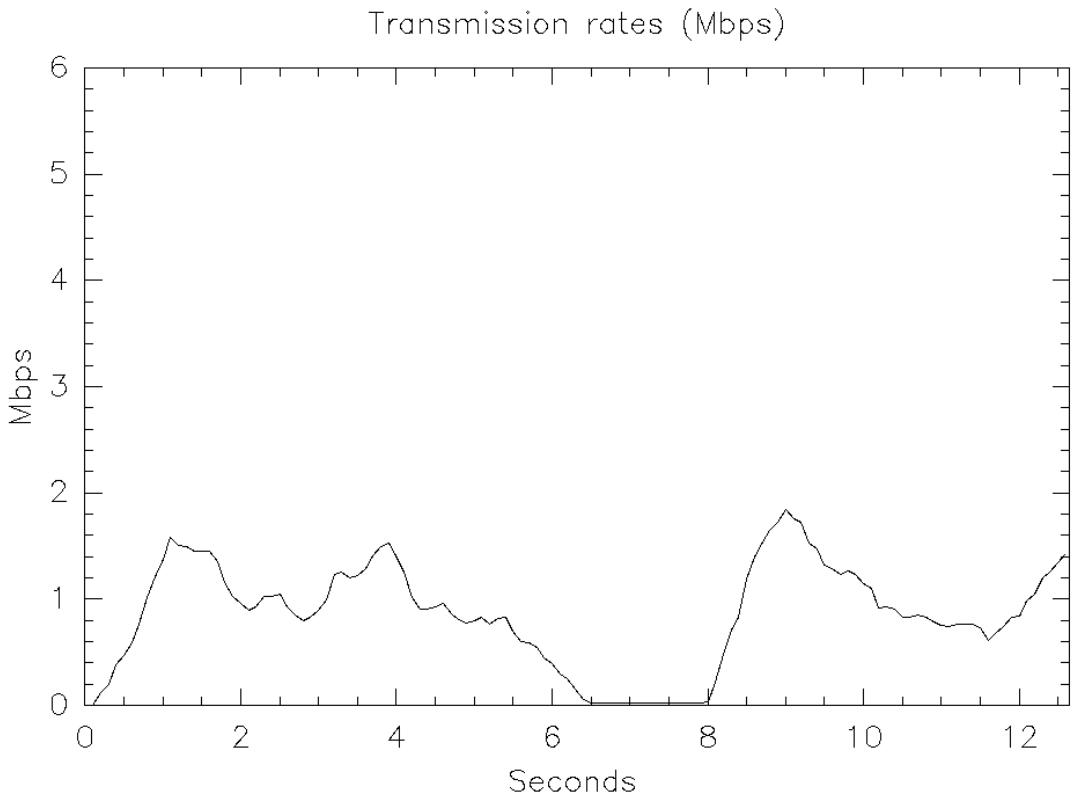

Figure 5.26: *Transmission Rates at mesh6 over time for a transmission from mesh8 through mesh9, mesh1, mesh4, mesh2, mesh5, mesh7, and mesh6 to mesh3:* This graph shows the amount of data transmitted per second using a one-second sliding window plotted every 100ms. The transfer uses last hop acknowledgments and a buffer size of 128 packets.

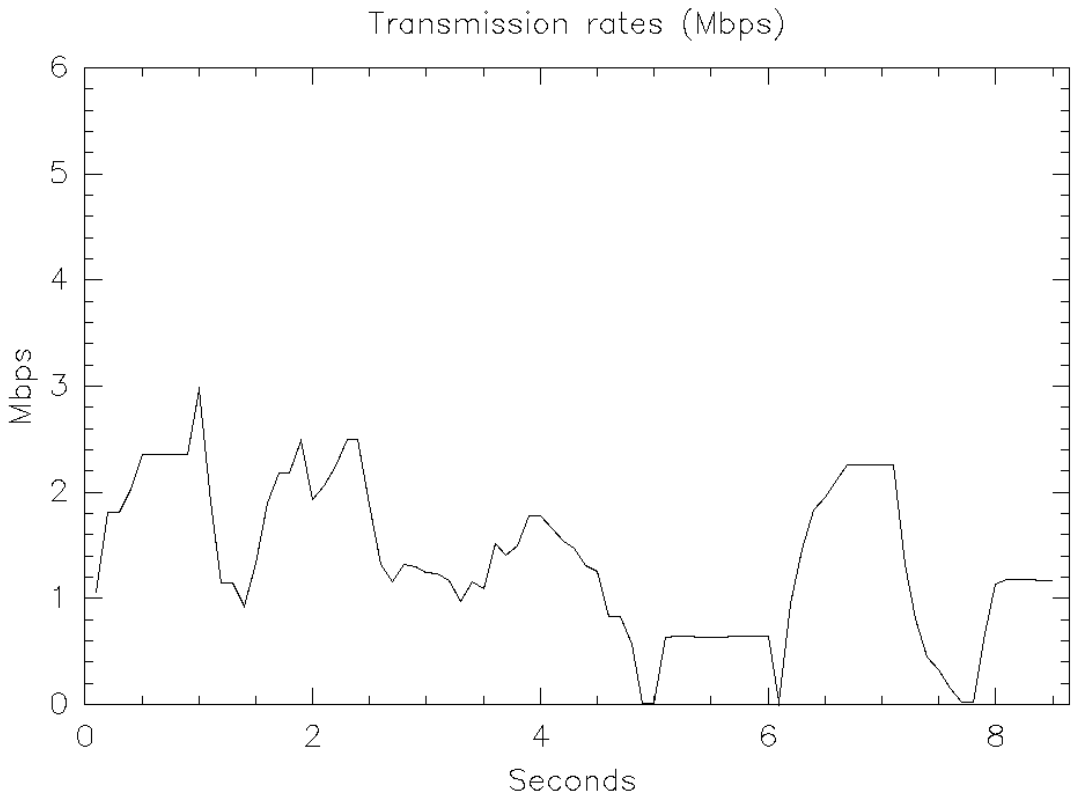

Figure 5.27: *Transmission Rates at mesh8 over time for a transmission from mesh8 through mesh9, mesh1, mesh4, mesh2, mesh5, mesh7, and mesh6 to mesh3:* This graph shows the amount of data transmitted per second using a one-second sliding window plotted every 100ms. The transfer does not use last hop acknowledgments and uses a buffer size of 128 packets.

dip does not last as long. As can be seen in the throughput graphs presented earlier, both variations of HxH have significant performance degradation in a portion of the trials. Figure 5.29 shows the transmission rates for mesh8 for a transfer where HxH performs quite poorly. The number of passive acknowledgments lost in this particular trial is high enough that HxH is constantly waiting for timeouts.

When the buffer size is increased to 256 packets the transmission rates at the source node and the first few hops are increased significantly. The amount of increase decreases with each hop, but the transmission rate at the end is still generally higher than it was

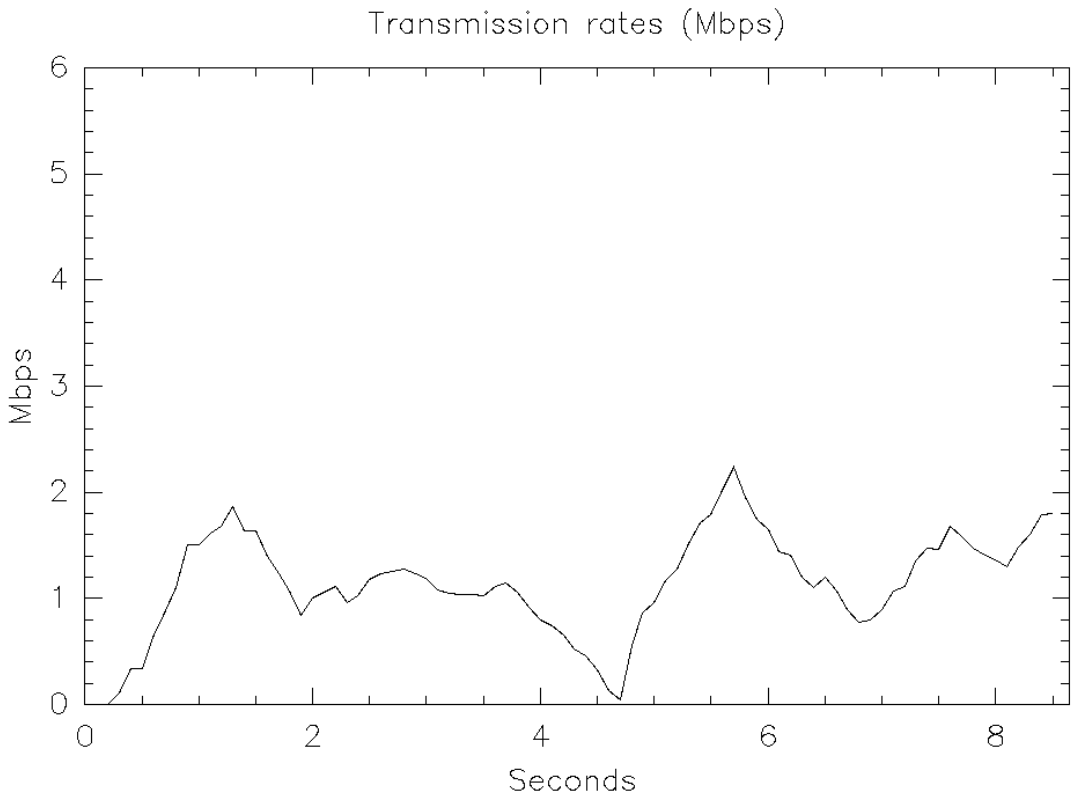

Figure 5.28: *Transmission Rates at mesh6 over time for a transmission from mesh8 through mesh9, mesh1, mesh4, mesh2, mesh5, mesh7, and mesh6 to mesh3:* This graph shows the amount of data transmitted per second using a one-second sliding window plotted every 100ms. The transfer does not use last hop acknowledgments and uses a buffer size of 128 packets.

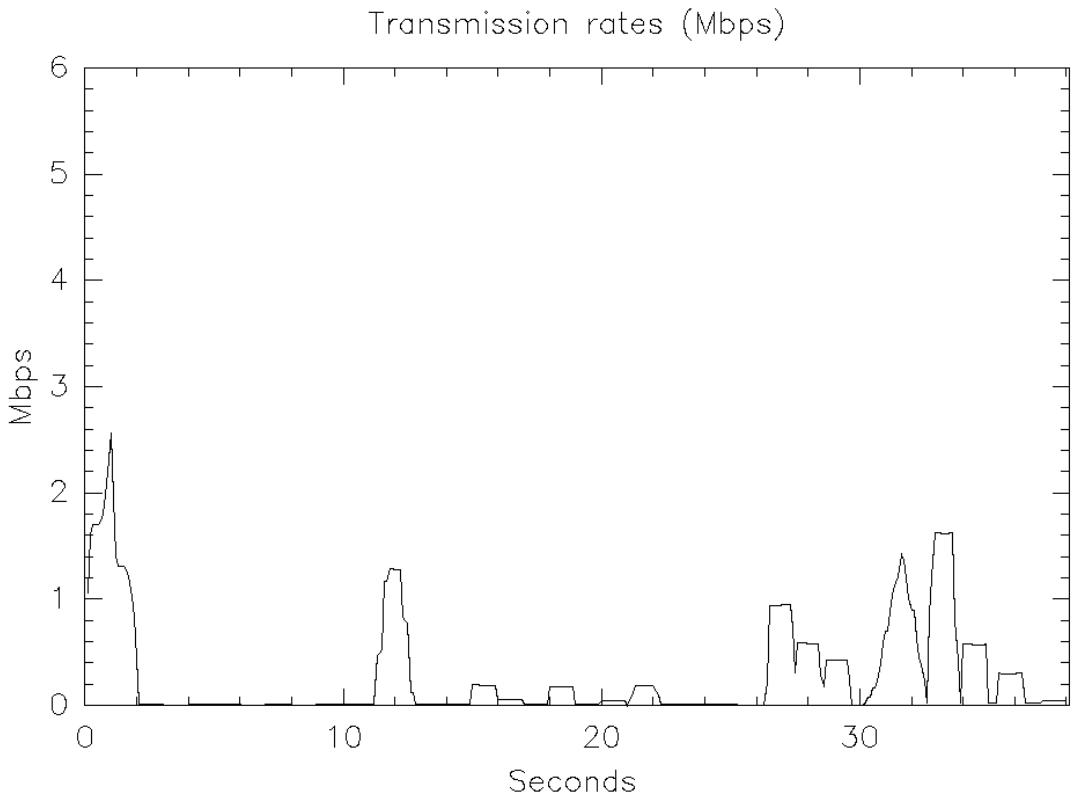

Figure 5.29: *Transmission Rates at mesh8 over time for a transmission from mesh8 through mesh9, mesh1, mesh4, mesh2, mesh5, mesh7, and mesh6 to mesh3:* This graph shows the amount of data transmitted per second using a one-second sliding window plotted every 100ms. The transfer uses last hop acknowledgments and a buffer size of 128 packets.

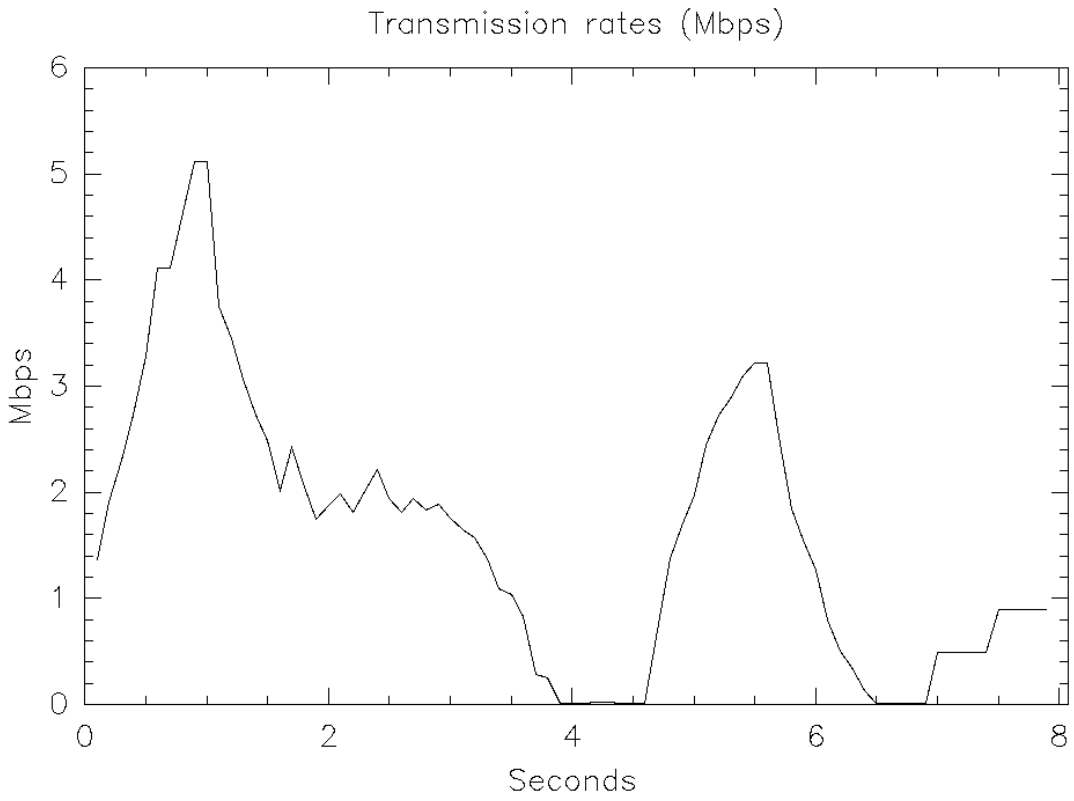

Figure 5.30: *Transmission Rates at mesh8 over time for a transmission from mesh8 through mesh9, mesh1, mesh4, mesh2, mesh5, mesh7, and mesh6 to mesh3:* This graph shows the amount of data transmitted per second using a one-second sliding window plotted every 100ms. The transfer does not use last hop acknowledgments and uses a buffer size of 256 packets.

with smaller buffer sizes. Figures 5.30 and 5.31 show transmission rate graphs for a typical transfer with a buffer size of 256 packets. As before there are a number of trials that look more like Figure 5.29, but this is again the exception and not the rule.

Increasing the buffer size to 512 packets does not change much as far as the transmission rates are concerned. The primary difference is that the transmission rate at the source node after the first ACK is received tends to be higher, in general closer to 5.5 Mbps instead of 3.5 Mbps. This rate drops off rapidly over the first few hops due to packet loss, meaning

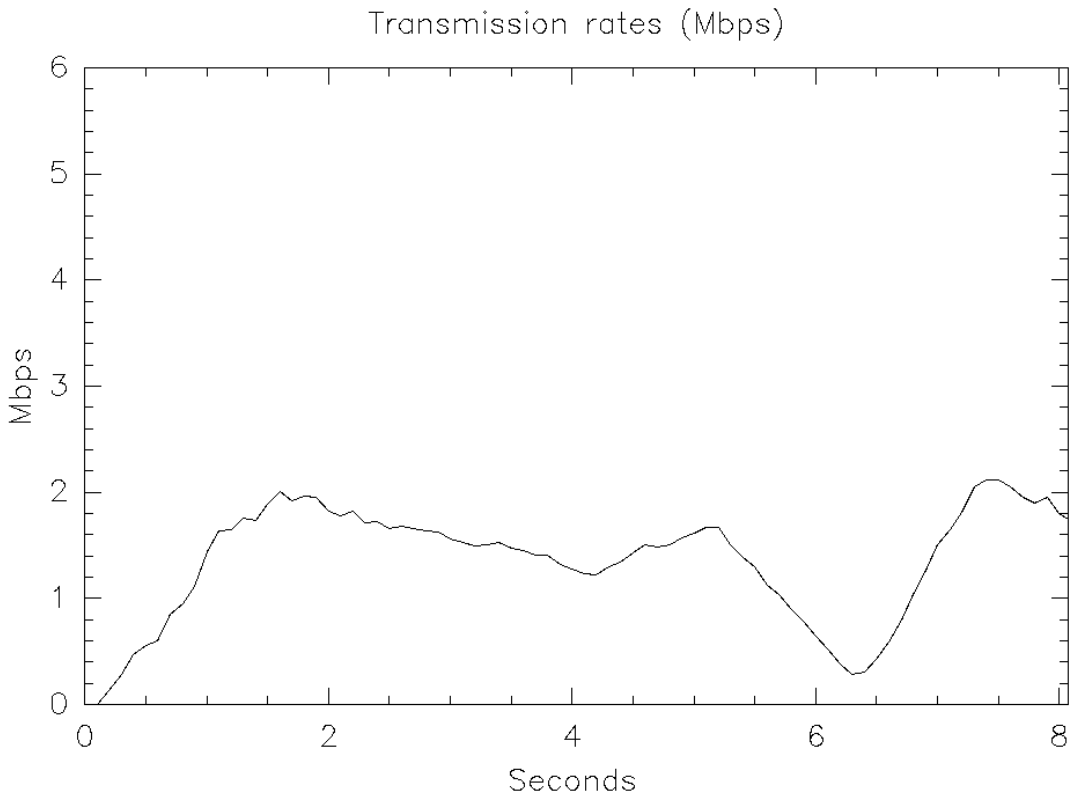

Figure 5.31: *Transmission Rates at mesh6 over time for a transmission from mesh8 through mesh9, mesh1, mesh4, mesh2, mesh5, mesh7, and mesh6 to mesh3:* This graph shows the amount of data transmitted per second using a one-second sliding window plotted every 100ms. The transfer does not use last hop acknowledgments and uses a buffer size of 256 packets.

that the increased transmission rate is probably causing more packet loss and amounts to nothing more than extra contention in the network.

Turning on the RTS/CTS exchange causes the transmission rate to be more bursty. Figures 5.32 and 5.33 show transmission rate graphs for a transfer using HxH with last hop acknowledgments with the RTS/CTS exchange enabled for all packets. These graphs are typical of all of the transfers where RTS/CTS is enabled. The transmission rates dip much more frequently and create a bursty traffic pattern. Figures 5.34 and 5.35 show graphs for transfers without last hop acknowledgments. The graphs have fewer peaks and values but are still much more bursty than when RTS/CTS is disabled. Setting the threshold for RTS/CTS so that it is enabled for packets 500 bytes and larger does not have any noticeable effect on the transmission rates.

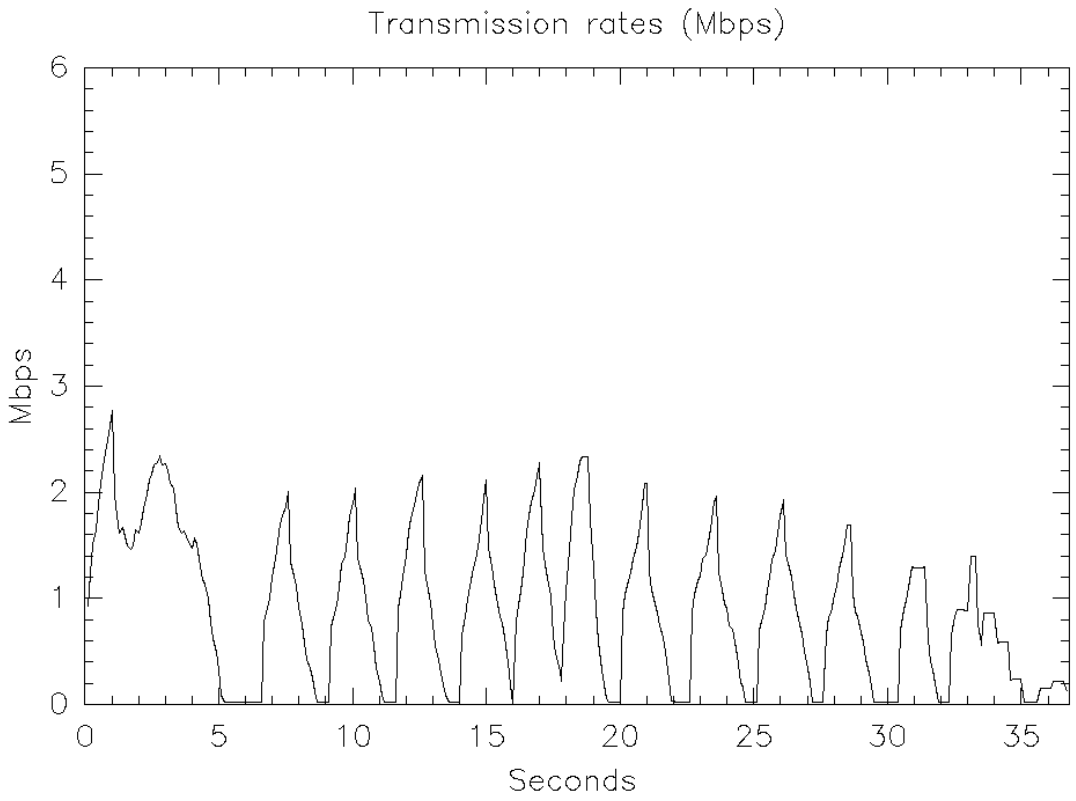

Figure 5.32: *Transmission Rates at mesh8 over time for a transmission from mesh8 through mesh9, mesh1, mesh4, mesh2, mesh5, mesh7, and mesh6 to mesh3:* This graph shows the amount of data transmitted per second using a one-second sliding window plotted every 100ms. The RTS/CTS exchange is enabled for all packets and the transfer uses last hop acknowledgments and a buffer size of 256 packets.

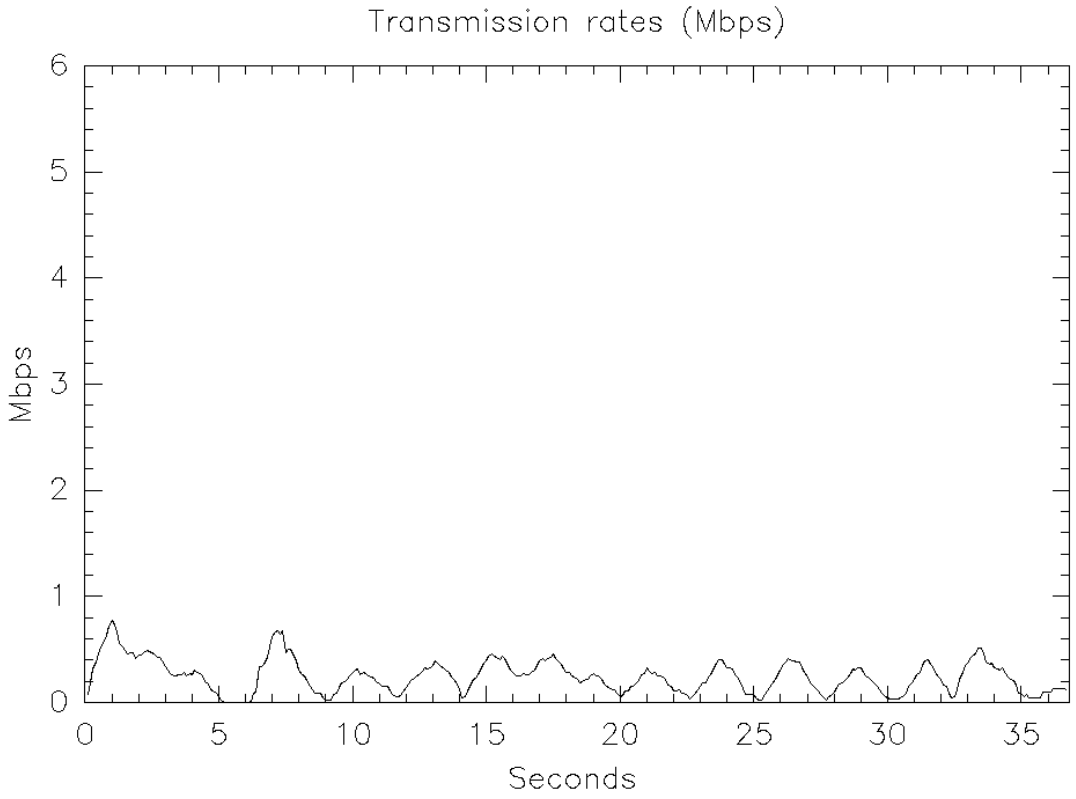

Figure 5.33: *Transmission Rates at mesh6 over time for a transmission from mesh8 through mesh9, mesh1, mesh4, mesh2, mesh5, mesh7, and mesh6 to mesh3:* This graph shows the amount of data transmitted per second using a one-second sliding window plotted every 100ms. The RTS/CTS exchange is enabled for all packets and the transfer uses last hop acknowledgments and a buffer size of 256 packets.

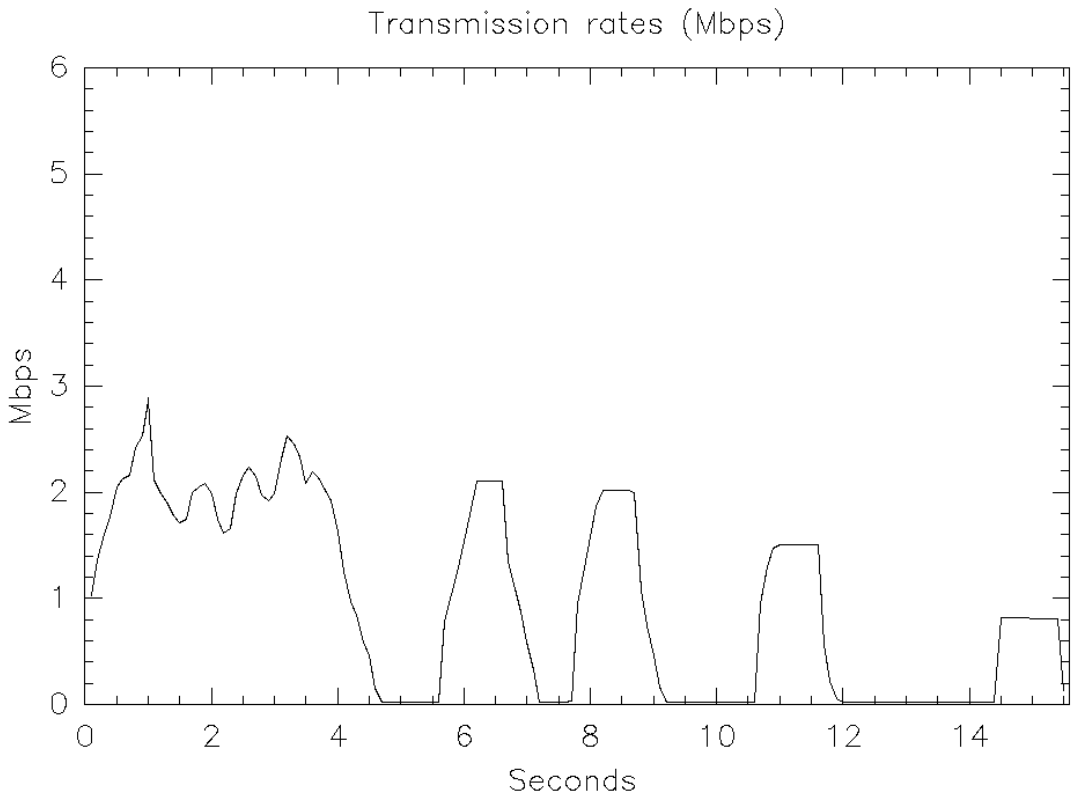

Figure 5.34: *Transmission Rates at mesh8 over time for a transmission from mesh8 through mesh9, mesh1, mesh4, mesh2, mesh5, mesh7, and mesh6 to mesh3:* This graph shows the amount of data transmitted per second using a one-second sliding window plotted every 100ms. The RTS/CTS exchange is enabled for all packets and the transfer uses a buffer size of 256 packets and does not use last hop acknowledgments.

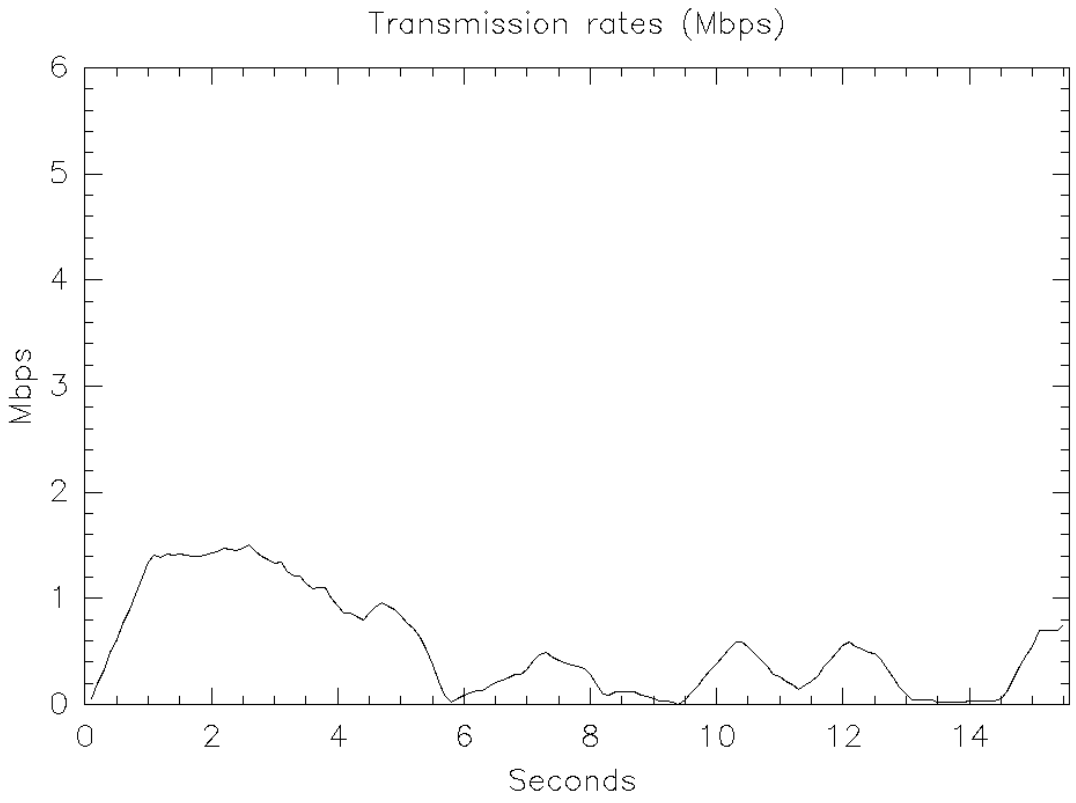

Figure 5.35: *Transmission Rates at mesh6 over time for a transmission from mesh8 through mesh9, mesh1, mesh4, mesh2, mesh5, mesh7, and mesh6 to mesh3:* This graph shows the amount of data transmitted per second using a one-second sliding window plotted every 100ms. The RTS/CTS exchange is enabled for all packets and the transfer uses a buffer size of 256 packets and does not use last hop acknowledgments.

Chapter 6

Conclusions

This thesis has detailed the creation of a wireless mesh network and a user-space implementation of the HxH protocol for multi-hop wireless networks. HxH relies on passive feedback and the reliability mechanism built into the MAC layer in order to avoid using up bandwidth with explicit feedback. In simulations this reliance results in significant performance gains over other protocols. By measuring various aspects of our HxH implementation we are able to understand whether the underlying assumptions of the protocol that held in simulations remain true in real networks.

Our implementation of HxH has some modifications from the simulator version. The framework we use does not allow us to overhear the MAC layer exchanges and so the protocol has to simply assume that packets transmitted have been received by the next hop. In order to avoid letting the passive acknowledgments get out of sync with reality, we implement an explicit acknowledgment at the last hop. We also add more information to the explicit ACK packets so we can avoid sending a packet from the destination to the source for every packet that is missed.

The mesh network is much more complex than the simulator. The signal strengths for different radios varies in different directions, and nodes that can reliably transfer data to a second node often lose a large percentage of the packets transmitted in the opposite direction. Moreover, at different times the same pair of nodes can have substantially different loss rates. This complexity and variability in the network affect HxH, making it less consistent and less effective than it is in simulations.

The amount of packet loss observed in HxH is much higher than in simulations, but this can be mitigated in part by increasing the buffer size. With the right buffer settings the HxH protocol can be made to have higher throughput than TCP as long as the rate of packet loss is not too high. Packet loss on the network cannot be controlled directly by HxH, but HxH could be redesigned to allow it to work better in such environments.

The packet loss hurts throughput in that if enough passive acknowledgments are lost the transmission stalls, because an upstream node believes that the downstream node has no buffer space for the packets it needs to send. Because the implementation cannot overhear the RTS/CTS exchange, whenever data packets are lost the source must retransmit them. If many packets are lost this means several round-trips from the destination to the source and back requesting missing packets.

The credit-based congestion control of HxH is also highly sensitive to packet loss. In order to increase throughput by reducing the likelihood that lost passive acknowledgments will cause the protocol to stall, we increase the buffer size at each hop. This also means that transmissions are more bursty than with smaller buffers, which affects the fairness of HxH by making it more difficult for HxH to respond to changing network congestion.

The underlying philosophy of HxH shows great promise, even if much of the original ns-2 implementation does not work as well in a real network. The mesh network built for this thesis will be a useful tool for future protocol development and testing and will help us make informed decisions on how to build multi-hop wireless protocols. Lessons learned from implementation choices made during the course of this research are already being used to improve upon this initial implementation.

6.1 Open Issues and Future Work

The failure of the HxH implementation to perform as well in a real mesh network as in the simulator suggests several modifications to the implementation that could be made to

improve its performance. It also suggests some ways we could tune the mesh network itself to make it a more hospitable environment for HxH.

6.1.1 Modifications to the HxH implementation

When the HxH implementation performs poorly it is nearly always because the transfers end up in a stalled state due to the loss of passive acknowledgments. In order to mitigate this situation several modifications could be made to this portion of HxH.

As implemented HxH does not transfer a packet to its downstream neighbor unless it either has a positive credits value or it has waited for a predetermined timeout period without receiving any credits messages. This mechanism could be modified to include a variable timeout mechanism where the more times it enters the wait state the shorter the wait. It could alternately be modified to request that the downstream neighbor explicitly send it the number of credits it has whenever it reaches the wait state. Either of these modifications could improve overall throughput by keeping the transmission going, but it could also introduce packet loss due to buffer overflow.

The local recovery mechanism built into HxH does not work as implemented, except at the last hop. Because libpcap does not let us listen to the RTS/CTS exchange we have no way to know that the packet actually gets to the next hop, but, more importantly, we have no way to know that the packet did not get to the next hop. In cases where the packet does not reach the next hop it is very simple to have the upstream neighbor repeat the transmission. In order to mitigate this side effect of using libpcap we could modify HxH to send explicit acknowledgments at predetermined intervals. Each hop would keep track of the packets it has sent that have not yet been acknowledged by the next hop and would retransmit them as necessary. End-to-end recovery is much less efficient than local recovery and is generally responsible for a good portion of the time used to transfer the data.

Another modification that might significantly improve throughput is to stop using libpcap to transmit and receive packets on the wireless interface. A mechanism that has

shown great promise in early trials is to use iptables. A kernel implementation could also significantly improve performance.

6.1.2 Modifications to the wireless mesh network

Most of the problems the HxH implementation encounters can be traced back to higher-than expected packet loss in the network itself. If we can reduce the amount of packet loss on the network HxH will likely perform much better. Any number of components and settings could be responsible for this packet loss, including the routing protocol, the wireless interface settings, the wireless interface driver, and external interference on the frequencies we use. These factors could be modified to optimize the wireless mesh network itself.

References

[1] Bay Area Wireless User Group Web Site. http://www.bawug.org/.

[2] Chaska.net. http://chaska.net/.

[3] Champaign-Urbana Community Wireless Network. http://www.cuwireless.net.

[4] P. Dinda. The Minet TCP/IP Stack. *Northwestern University Department of Computer Science Technical Report NWU-CS-02-08*, 2002.

[5] D. Ely, S. Savage, and D. Wetherall. Alpine: A User-level Infrastructure for Network Protocol Development. In *Proceedings of the 3rd Conference on USENIX Symposium on Internet Technologies and Systems*, volume 3, 2001.

[6] Z. Fu, H. Luo, P. Zerfos, S. Lu, L. Zhang, and M. Gerla. The Impact of Multi-hop Wireless Channel on TCP Performance. *IEEE Transactions on Mobile Computing*, 4(2):209–221, 2005.

[7] Kismet. http://www.kismetwireless.net/.

[8] MeshDynamics. http://www.meshdynamics.com/.

[9] MIT Roofnet. http://pdos.csail.mit.edu/roofnet/doku.php.

[10] olsrd. http://www.olsr.org/.

[11] libpcap. http://sourceforge.net/projects/libpcap/.

[12] P. Pradhan, S. Kandula, W. Xu, A. Shaikh, and E. Nahum. Daytona: A User-Level TCP Stack. http://nms.lcs.mit.edu/kandula/data/daytona.pdf.

[13] D. Scofield. Hop-by-hop Transport Control for Multi-hop Wireless Networks. *MS Thesis, Brigham Young University*, 2007.

[14] Seattle Wireless. http://www.seattlewireless.net/.

[15] SFLan. http://www.sflan.org/.

[16] Southampton Open Wireless Network. http://www.sown.org.uk/.

[17] K. Sundaresan, V. Anantharaman, H.-Y. Hsieh, and R. Sivakumar. ATP: A Reliable Transport Protocol for Ad-hoc Networks. In *MobiHoc '03: Proceedings of the 4th ACM International Symposium on Mobile Ad Hoc Networking & Computing*, pages 64–75, New York, NY, USA, 2003. ACM Press.