Experiments on a Multichannel Multi-Interface Wireless Mesh Network

Technical Report (May 2008)

Thomas B. Shen and Nitin H. Vaidya
Department of Electrical and Computer Engineering and
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
{tbshen,nhv}@uiuc.edu

Abstract

Wireless mesh network technology provides a quick method of providing network access. Traditional wireless mesh networks face problems regarding spatial reuse. The simultaneous use of multiple interfaces and channels attempts to solve this problem. Support for these multiple channels and devices has been implemented through the Net-X system. Performance of the network depends on the amount of cross channel interference because wireless transmissions use a shared medium and interfere with each other unless properly separated in distance and frequency. We test the interference experienced in single, dual, and triple interface nodes.

A protocol needs to support the most common types of TCP and UDP traffic for it to be useful. One popular type of UDP traffic is that of voice over internet (VoIP). VoIP traffic is more demanding on a network connection, but we find that the protocol is able to support its requirements for one and two-hop traffic. We propose a delay reduction scheme to reduce the delay caused by the frequent channel switching.

Another method of increasing performance of the network is to utilize more interfaces. The current implementation of the testbed uses two interfaces per node. Adding a third interface will require some changes in the hybrid multichannel protocol. Experiments show that multihop TCP transfers benefit from this added interface.

Based on these experiments, we propose some ideas that can be used in future multichannel work.

I. Introduction

Over the years, wireless internet access has become more and more popular, with wireless access points popping up everywhere. Wireless mesh technology is also becoming more popular as a cost-effective means of providing wireless access. In a wireless mesh, intermediate nodes are crucial in providing access to end nodes. This is similar to the ad-hoc mode in 802.11, but usually refers to a larger scale network like a neighborhood or community.

The demands we place on an internet connection continue to increase with users expecting to use voice over IP and watch internet TV. Development in new wireless protocols like 802.11n makes it possible to meet the continuously rising demands of users. Another solution is to use existing technology in parallel to increase performance. There are multiple frequency channels in 802.11a and 802.11b that can be utilized simultaneously to improve performance. System support for the use of multiple frequencies and channels is provided by the Net-X framework which has been developed by previous members of the Wireless Networking Group at the University of Illinois [1]. Details about the current Net-X system that are necessary for following the discussion can be found in Section II.

Utilizing multiple interfaces and channels may not always result in better performance. Great care must be taken in selecting channels to use as cross channel interference undermines performance. Section III of this thesis will investigate the effects of interference from transmitters on other channels. Cross channel interference of multiple transmitters and receivers located in close proximity is also investigated to determine the suitability of a multiple interface system. A set of experiments is also performed in an anechoic chamber. The performance of real traffic will be one of the factors that determines the usability of a protocol. Section IV tests the performance of voice-over-IP (VoIP) on the testbed and proposes a method of reducing the delay. In Section V, we discuss implementation issues of adding a third interface. Experiments to test the benefits of the third interface are also carried out. Lastly, Section VI includes concluding remarks and ideas for the future.

II. Background

The results in this paper are from working on a unique multichannel multi-interface testbed (Net-X) implemented by the Wireless Networking Group at the University of Illinois-Urbana Champaign. To help the reader better understand the rest of the thesis, a summary of the relevant aspects of the testbed follows.

Multiple Interfaces: The number of interfaces physically limits the number of transmissions and receptions that can take place simultaneously. By increasing the number of interfaces, we increase the upper bound on the capacity of the network [2]. As expected, the number of interfaces cannot be increased arbitrarily high as hardware limitations (expansion slots, CPU, power) and the number of wireless channels will be limiting factors. Additionally, the financial cost of adding extra interfaces

may be too high, although the price has certainly come down in recent years and made multiple interface devices more feasible. Two wireless interfaces are used in the current Net-X implementation.

Multiple Channels: Wireless devices share the wireless medium in which they communicate. Multiple channels are needed for multiple interfaces to transmit simultaneously. Depending on the country, there can be up to 14 channels available in 802.11b/g and 13 channels in 802.11a. There is often cross interference on neighboring channels. Adya et al. [3] found that the number of orthogonal channels depends on the model/vendor of the wireless interface as well as the physical separation between the interfaces. Through experimentation, Chereddi found five orthogonal channels in 802.11a [1]. Five channels are used in the current Net-X testbed.

Hybrid Multichannel Protocol: A hybrid multichannel protocol (HMCP), as proposed in [4], [5], makes it possible for a network of **N**-interface devices to make use of **M** wireless channels. In our network, N=2 and M=5. One of the interfaces is fixed on a channel for a relatively long time while the other interface is deemed "switchable," switching channels to communicate with neighboring nodes. To amortize the cost of switching channels, which takes 5 ms, the switchable interface will spend a minimum of 20 ms on a channel. When there are packets waiting on other channels, the interface will spend a maximum time of 60 ms + five deferrals of 10 ms each before forcing a switch. The HMCP is also responsible for notifying the kernel of any changes to the routing tables.

Hello Message System: The Hello Message Subsystem is required by the HMCP to establish connectivity. Nodes exchange messages at fixed intervals to keep their neighbors up to date on their status. The fixed interface of a node may change if the network conditions dictate that a change is necessary. This allows the network to update and rebalance itself. To exchange this information, the node sends out a message on all possible channels. Currently, the fixed interval is set to 5 s.

Routing Protocol: The routing protocol currently used on the Net-X testbed is a modified version of Weighted Cumulative Expected Transmission Time (WCETT) [6]. This is a reactive protocol that attempts to find a lowest cost path according to expected transmission time and channel diversity. Modifications were made by Kyasanur to select channel diverse paths [2].

Further details about the Net-X system, links to papers, and source code can be found at http://www.crhc.uiuc.edu/wireless/netx.html.

III. CROSS CHANNEL INTERFERENCE

To increase throughput and spatial reuse, a multichannel network receives and transmits on multiple channels. There are 12 802.11a channels sanctioned by the FCC for use in the USA. Each channel is 20 MHz wide and centered 20 MHz apart. In practice, Chen et al. found that there is often signal leakage between adjacent channels causing interference [7]. They verified that the amount of signal leakage into neighboring channels conforms with IEEE 802.11a standards. The IEEE standard on high-speed physical layer design in the 5 GHz band [8] shows on page 29, Figure 120, that signal power 11 MHz from the center frequency is to be 20 dB lower than the signal at the center and 28 dB lower at 20 MHz away. While the authors in [7] have shown there is interference caused between two co-located antennas, we have also seen the interference problem in interfaces placed 10 feet apart.

We would like to determine the extent of this interference through experiments. Although the results will be dependent on the specific wireless interface as well as antenna used, we believe the general characteristics of all wireless cards with the same chipset will be similar. The disparity between interfaces is due to each vendor implementing different filters for the transmit mask. The results of these experiments will aid in the selection of channels to use in a multichannel network. Then we extend the experiments to two and three interfaces, since the extra interference of the added interfaces may prevent some channel combinations from being used.

A. Experimental Setup

1) Hardware: We utilize the Soekris net4521 mainboard to run version 41 of Pebble Linux. The mini-PCI slot is populated with a SL-5354MP ARIES wireless 802.11a/b/g card that uses the Atheros AR5212A chip. Tests with one PCMCIA card are carried out using a Senao multiband card. When running two and three interface tests, we use the Senao PCMCIA cards along with the ARIES mini-PCI.

2) Software: The modified version of *madwifi* used in Net-X is also used in these experiments. We also tried the latest version 0.9.4 of the *madwifi* driver but found the results to be very inconsistent in certain simulations. The advantage of using the new version is the ability to change the multicast rate so that the interference can be tested at higher data rates.

We use *iperf* to send packets at a desired rate. The reason for using multicast packets is to avoid the retransmission of unacknowledged data packets in the 802.11 medium access control layer. By avoiding the automatic retransmissions, we get to see exactly how many times a transmission is successful. Unicast transmissions are also carried out to see the effect of having to acknowledge the data. Ultimately, most applications will utilize unicast transmissions and it is important to see if unicast performs well.

3) Experiment details: Three nodes are placed in a line, approximately 10 feet from each other as in Figure 1. Each node is within line of sight of the next. Node A at one end transmits packets to multicast address 224.0.0.1, of which node C is a member. Meanwhile, node B in the middle will transmit packets to multicast address 224.0.0.2. There is no node listening on the 224.0.0.2 multicast address because the purpose of this transmission is to create interference. While the main transmission will have a load at the full rate of the connection, the interfering transmission will have a variable load to measure its effect on the main transmission. The channel of the interfering transmission will also be varied between two channels below and two channels above the channel of the main transmission.

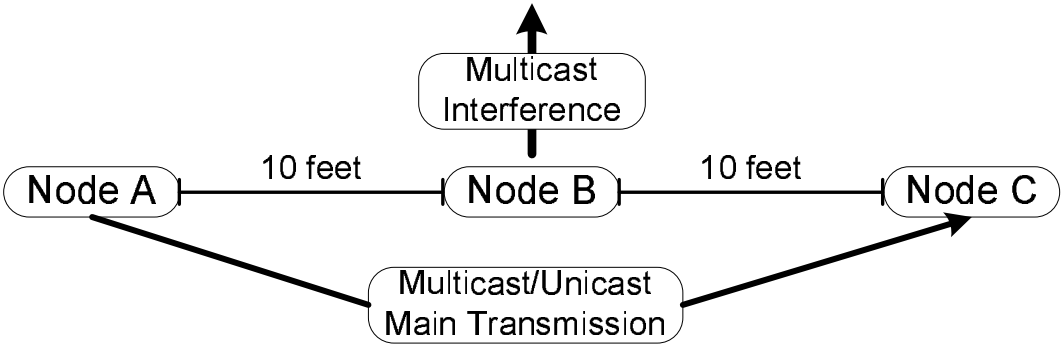


Fig. 1 Node placement

The transmit power setting is set using *iwconfig* to be 99 mW for all of the experiments unless otherwise stated. This is the highest setting allowed. (Note: There were many difficulties encountered when setting the transmit power, rate, and ad-hoc mode. We found that writing the settings in a script executed at startup time provided the most consistent and reliable results.)

B. Driver Data Rate Tests

First, we conducted tests to find the maximum throughput for unicast and multicast traffic in the absence of interfering traffic from node B. As expected, the multicast throughput was slightly higher than the unicast throughput (5.53 Mbps vs. 5.37 Mbps). Multicast packets are similar to broadcast packets in that there is no acknowledgment. This translates to more time to transmit data packets, resulting in higher throughput. In the presence of interference, a given multicast packet will not reach its destination because there is no automatic retransmission.

Next, we tried the newer version of *madwifi* to test for any performance improvements and the ability to change the multicast rate. The unicast throughput for all data rates is reasonable as seen in column two of Table I. However, the multicast throughput for 24, 36, and 54 Mbps data rates seems to be throttled down to the 18 Mbps rate. Numerous trials were repeated but were unsuccessful in achieving a higher throughput. Thus, we suspect either the driver or the card has been limited to a maximum of 18 Mbps when in broadcast/multicast mode. In the end, we tested only with the Net-X version of *madwifi* and fixed the data rate to 6 Mbps as in row two of Table I.

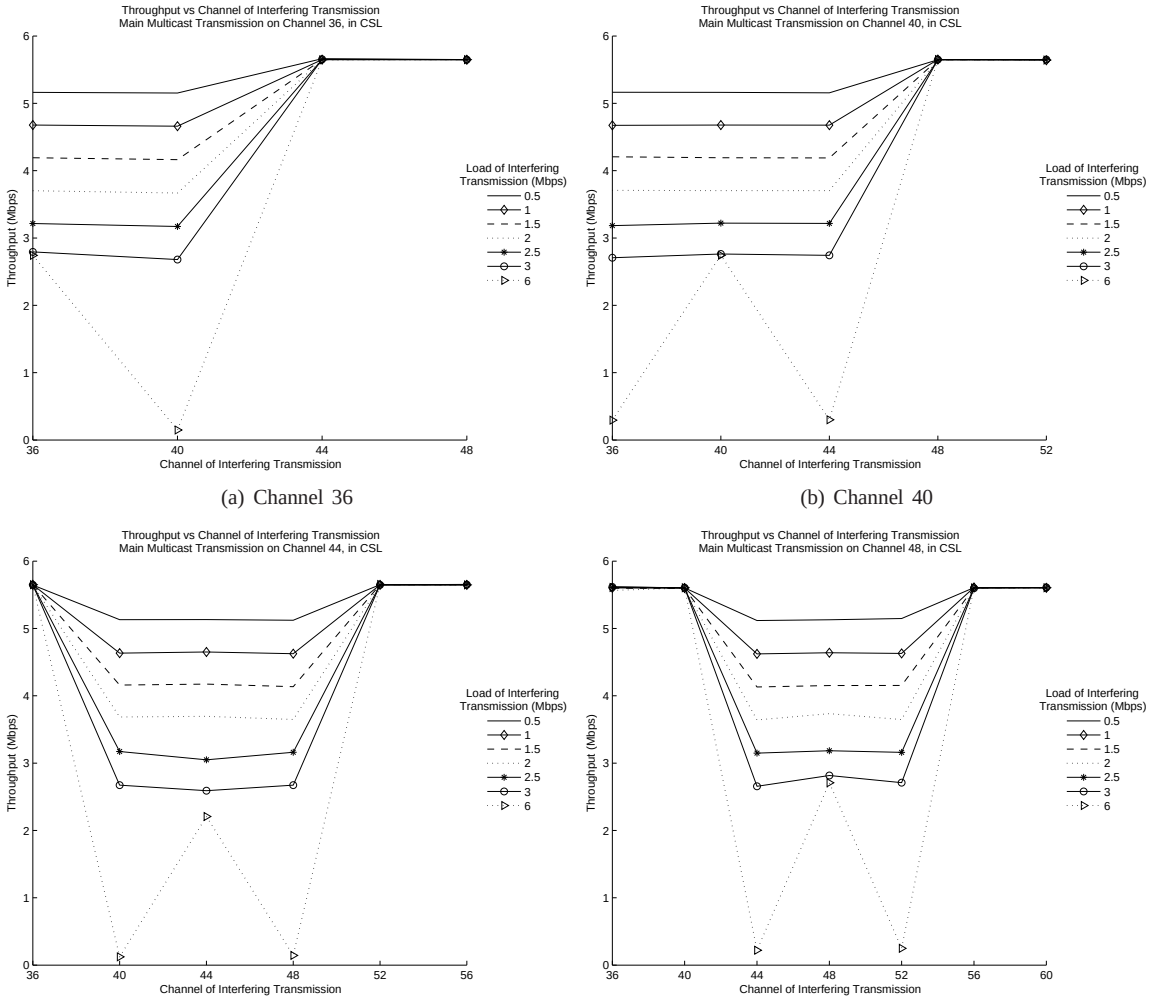
TABLE I Data rate throughput tests

Data Rate	Unicast Throughput (Mbps)	Multicast Throughput (Mbps)
6 Mbps (Net-X)	5.34	5.53
6 Mbps	5.37	5.53
9 Mbps	8.10	8.23
12 Mbps	10.47	10.65
18 Mbps	15.02	15.54
24 Mbps	19.30	14.09
36 Mbps	18.80	14.12
54 Mbps	21.56	14.56

C. Experiments in a Typical Setting

These tests are performed in an office in the Coordinated Science Laboratory. The Coordinated Science Laboratory provides wireless internet connectivity through a 802.11b/g network. There is no other known 802.11a network in use. However, this is an office environment where there may be cordless telephones using the 5 GHz spectrum. Thankfully, we are not using 802.11b/g in the 2.4 GHz band which is also susceptible to interference from microwave ovens. This location would be one of the possible deployment settings of a multichannel network.

The first tests are carried out with the mini-PCI card. The multicast results in CSL are displayed in Figure 2. The main transmission is on channel 36 in Figure 2(a) and the load is 6 Mbps. When the interfering transmission is also on the same channel 36, the throughput of the main transmission goes down an amount approximately equal to the interference load. When both loads are set to be 6 Mbps, the throughput is approximately half of the case without interference which is expected because the two are sharing the channel equally. Switching the interfering transmission to the adjacent channel (40) yields similar throughput at interference load rates up to 3 Mbps. However, fully loading the interfering transmission at 6 Mbps shows approximately 18 times worse performance compared to when the transmission is on the same channel. We see a similar trend on channels 40 (Figure 2(b) and 44 (Figure 2(c) and 48 (Figure 2(d). Previous work by Chereddi has surmised that this is due to the carrier sensing mechanism's ability to sense and thus defer transmission when the interference is on the same channel, but not when on the adjacent channel [1].



(a) Channel 36

(b) Channel 40

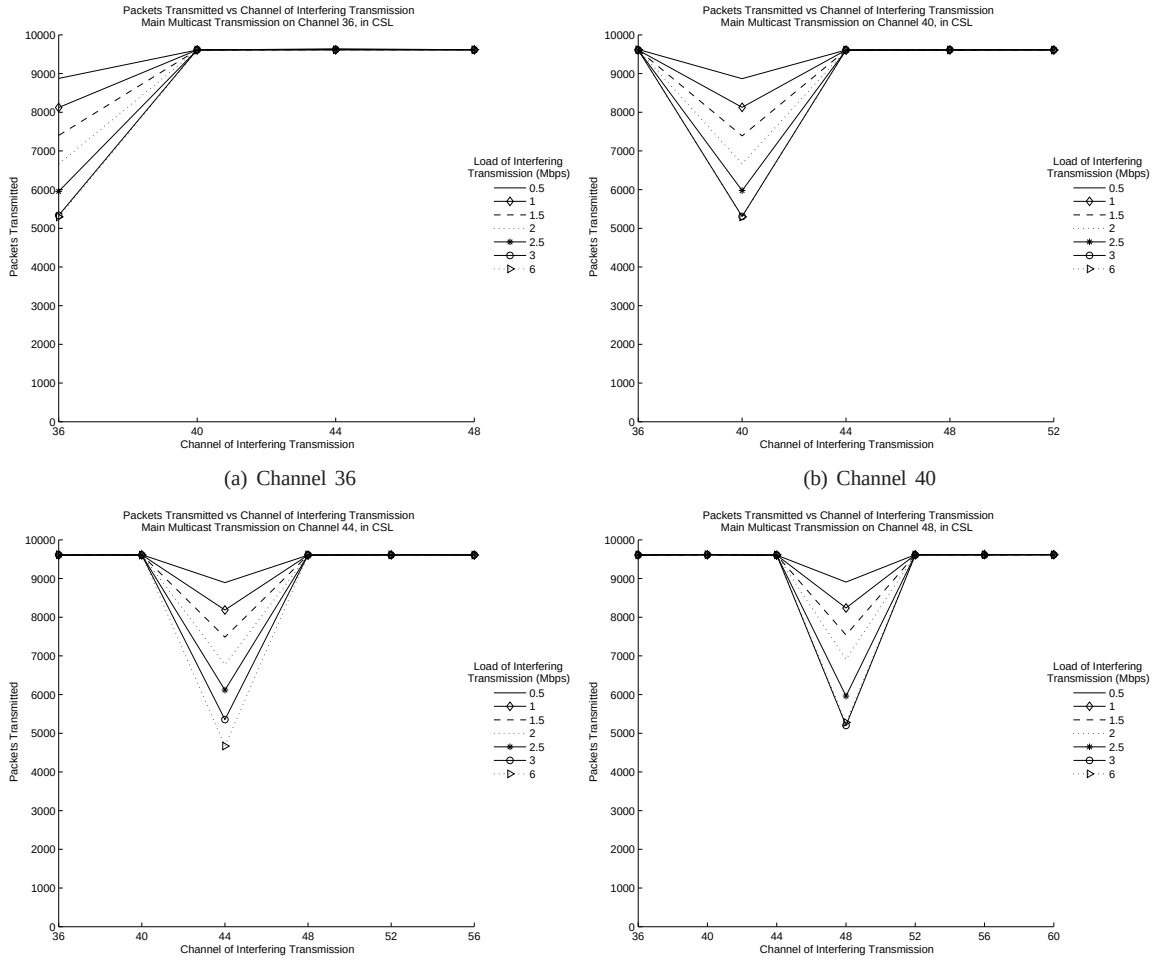
(c) Channel 44

(d) Channel 48

Fig. 2 Throughput with interfering transmission in CSL (mini-PCI)

To test this hypothesis, we recorded the number of packets that *ifconfig* reports to have been transmitted by node A. The results of this are plotted in Figure 3. Notice that when the interference is on the same channel, there are fewer attempted transmissions by node A. Moving the interference to the adjacent channel results in the same number of attempted transmissions as when there is no interference. The 802.11 MAC layer is not deferring in the face of adjacent channel interference. Two interfaces on adjacent channels are both flooding the network and causing collisions, resulting in poor throughput for both. This supports Chereddi's hypothesis and explains why adjacent channel interference can be more detrimental than same channel interference.

The same tests were repeated using unicast traffic for the main transmission. Unicast packets are retransmitted up to seven times when no MAC level ACK is received. The interference caused by the interfering transmission will not only prevent the



(a) Channel 36

(b) Channel 40

(c) Channel 44

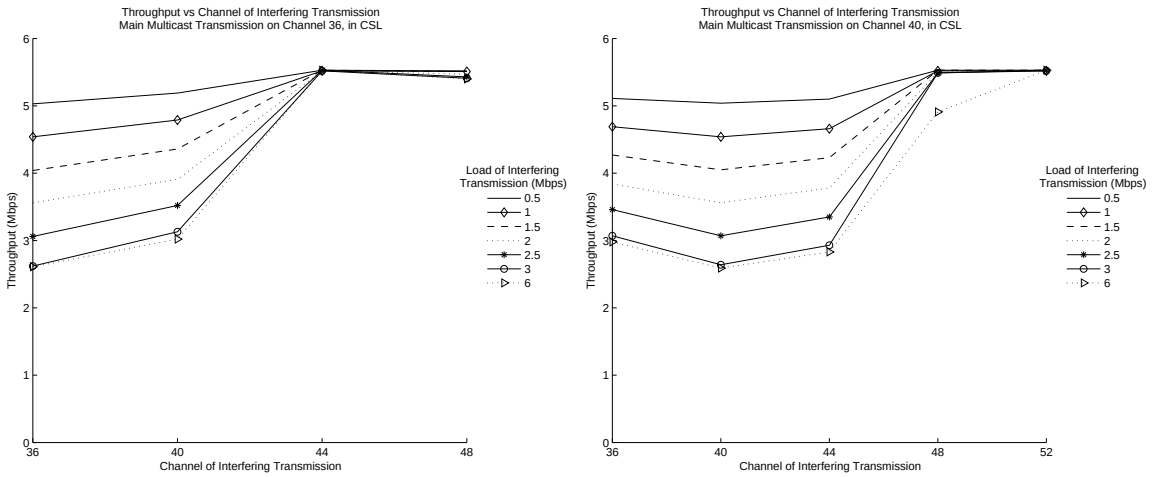
(d) Channel 48

Fig. 3 Packets transmitted with interfering transmission in CSL (mini-PCI)

data from arriving, but will also interfere with the MAC ACK and thus cause unnecessary retransmissions. Unicast results were similar to multicast results.

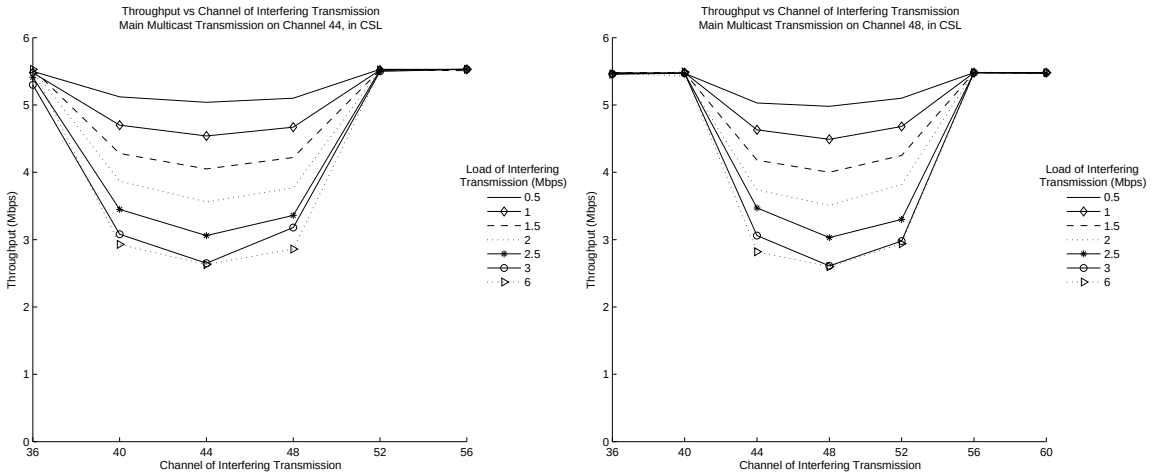
Our testbed also includes a variety of PCMCIA wireless cards. Testing each type of card would be too time consuming, so we chose the Senao cards that are used in our testbed. We ran the same experiments as before but changed all the interfaces to PCMCIA cards and found the results to be different from the mini-PCI results. The throughput results for multicast traffic from node A using only PCMCIA are displayed in Figure 4. The amount of interference when on the same channel is comparable to results with a mini-PCI interface, but the adjacent channel interference is drastically different. Referring back to Figure 2, we see that the throughput is less than 500 kbps for the mini-PCI interface while the throughput is almost 3 Mbps for the PCMCIA interface. The sent packet graphs in Figure 5 show that these PCMCIA cards are deferring transmission for adjacent channels. This could mean the transmission filters are not as good, allowing higher signal leakage and/or there is higher gain due to the different antennas being used. Interestingly, Figure 4(b) shows that there is some interference when the main transmission is on channel 40 while the interferer is on channel 48. This was the only test case in which transmitting on a nonadjacent channel had a noticeable impact. Notice that the sent packet graphs in Figure 5 look almost identical to Figure 4, suggesting that the transmission success rate is more consistent and that the deferral is succeeding in sharing access to the channel.

One way to mitigate adjacent channel interference is to lower the carrier sense threshold so that the interfaces can sense the leakage from other channels. Implementing this requires changes in the hardware abstraction layer, which is proprietary. There are other open source drivers which might provide this function, but we leave that for future work. We currently avoid adjacent channel interference in Net-X by selecting nonadjacent channels. To further reduce the chance of interference, we use channels that are at least three channels away.



(a) Channel 36

(b) Channel 40



(c) Channel 44

(d) Channel 48

Fig. 4 Throughput with interfering transmission in CSL (PCMCIA)

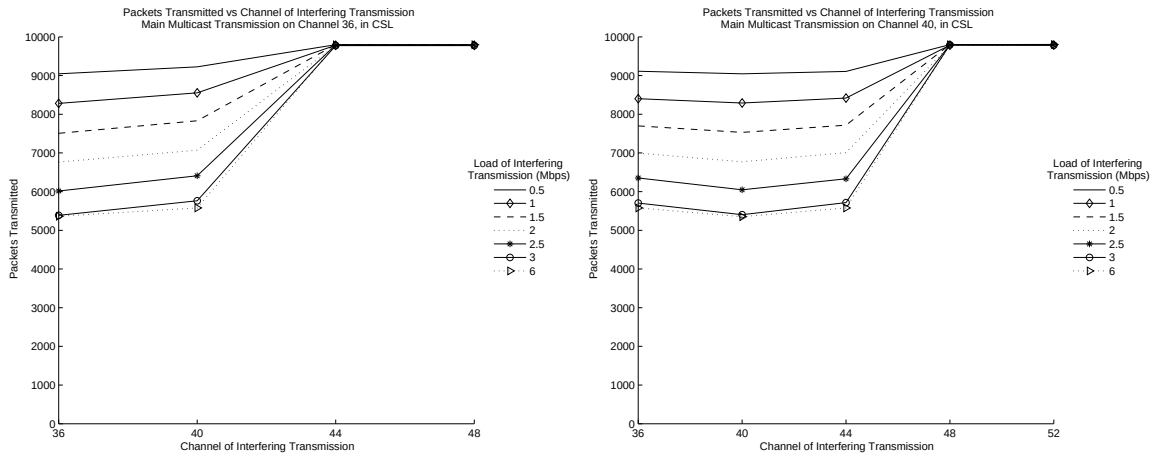
D. Experiments in Wireless Wind Tunnel

The completion of the Illinois Wireless Wind Tunnel (iWWT) allowed us to complete some of our experiments in a controlled environment [9]. The iWWT is an anechoic chamber so there is little interference from external sources. This is important because we do not want other sources of interference in the 5 GHz band in which we are testing. Furthermore, the walls of the iWWT are lined with absorbing foam that do not reflect much energy, reducing the effect of multipath fading. The results of these experiments will also help determine the applicability of performing experiments in the iWWT.

Our time in the chamber was limited, so we chose a subset of the experiments performed in the previous section. The mini-PCI cards with external antennas provided inconsistent data from day-to-day. This seemed to be due to the orientation of the antenna and the pig-tail connectors. Merely placing a hand above the antenna or standing behind the node could help establish connectivity, whereas before there was no connectivity. We decided to reduce the number of factors by working with the Senao PCMCIA cards which do not require an external antenna. We also reduced testing to the adjacent channel.

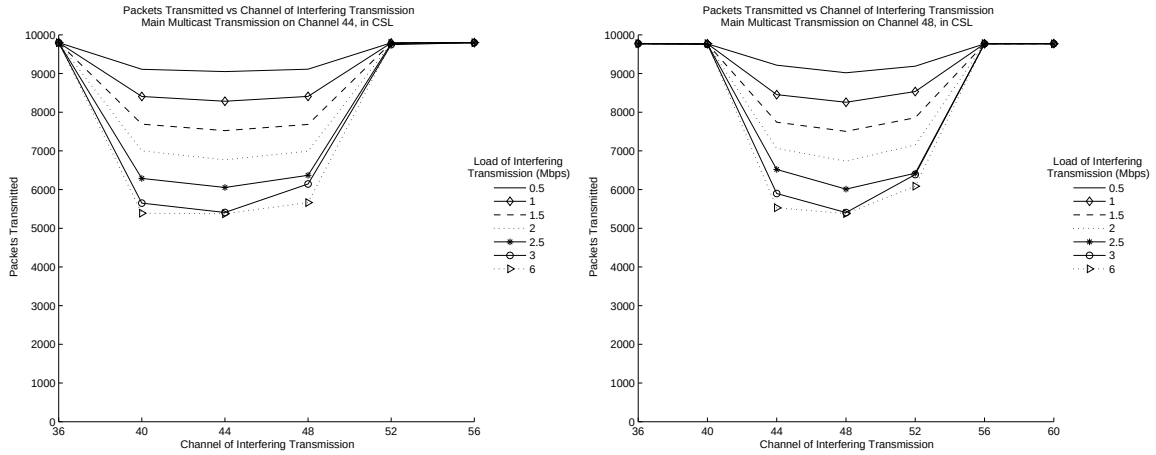
The throughput results in Figure 6 for the PCMCIA card in the wind tunnel look similar to the results from the mini-PCI experiments in CSL (Figure 2) and actually contrast with results of the same PCMCIA hardware in CSL. The attempted transmissions data show that the interface failed to defer when the interference was on adjacent channels, causing the throughput to be poor. A possible explanation for this is the lower ambient noise in the wind tunnel as well as the lack of multipath fading contributing to the received power. In turn, this smaller received power is lower than the carrier sense threshold, so the interface does not defer. Ironically, we want the received noise power to be higher so that it correctly defers.

Testing in the wind tunnel provided a controlled environment where it was guaranteed that no outside interference would affect the data. However, in our case, the data from the wind tunnel did not match with data gathered outside in a real world



(a) Channel 36

(b) Channel 40



(c) Channel 44

(d) Channel 48

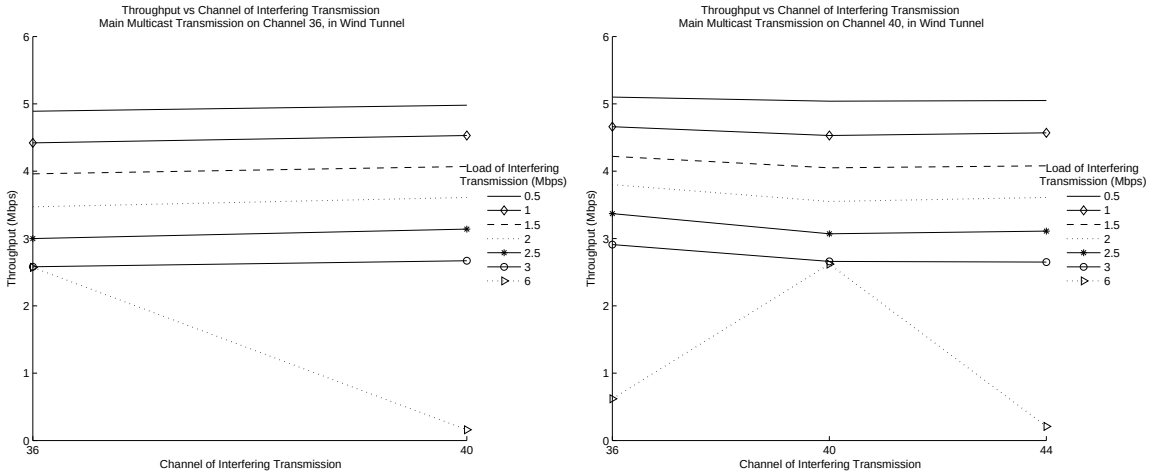
Fig. 5 Packets transmitted with interfering transmission in CSL (PCMCIA)

environment.

E. Multiple Interfaces

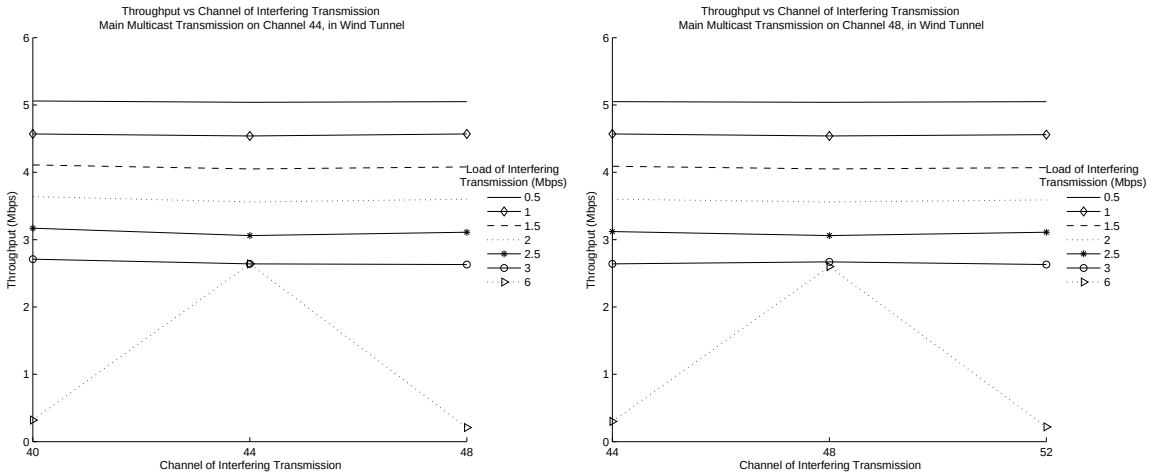
1) Two interfaces: Data presented in the previous sections suggests that it is possible to successfully transmit on two different interfaces at the maximum power level if they are not on adjacent channels and if they are separated by a distance of 10 feet. A multichannel multi-interface implementation will require the ability to simultaneously send and receive on separate channels from closely located interfaces. Previous work by Chereddi [1] has shown that when using two interfaces (one for reception and one for transmission), the channels utilized need to be at least three channels away for there to be no interference. One possible set of channels in 802.11a is $\{36, 48, 64, 149, 161\}$. Chereddi used a flooding broadcast ping as the interference and measured the throughput of the main transmission. We tried this method but found there to be little interference. Instead, we used *iperf* for both the main transmission and the interfering transmissions.

As Chereddi has pointed out previously, utilizing one transmitter and one receiver at the same node creates the most interference due to the high signal power of the transmitter interfering with the low signal power at the receiver. The setup of the nodes is displayed in Figure 7. Interface 1 is a mini-PCI card while interface 2 is a Senao PCMCIA card. The channel set was chosen from the channels that were deemed usable on our current Net-X testbed: $\{36, 48, 64, 149, 161\}$. Unicast transmission is used instead of multicast because there is no longer any need to measure the number of packets transmitted. The results of this test are displayed in Table II with the following color coding:



(a) Channel 36

(b) Channel 40



(c) Channel 44

(d) Channel 48

Fig. 6 Throughput with interfering transmission in wind tunnel (PCMCIA)

white	Multiple use of the same channel
light grey	Aggregate throughput x $\geq$ 10Mbps
medium gray	Aggregate throughput x, 8Mbps $\leq$ x $<$ 10Mbps
dark gray	Aggregate throughput x $<$ 8Mbps

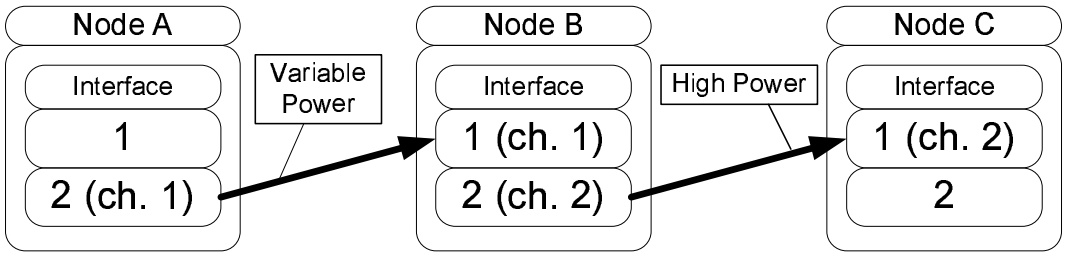


Fig. 7 Two interface configuration

When both interfaces utilize the same channel, they will have to share the channel and the aggregate throughput will be limited to around 5.37 Mbps. This can be seen on the downward diagonal. The only channel combination that exhibits a noticeable amount of interference is $\{48, 36\}$. (Note: It is not expected that $\{36, 48\}$ will also exhibit the same amount of

TABLE II Aggregate throughput (Mbps) of two transmissions

		Channel 1				
Channel		36	48	64	149	161
Channel 2	36	5.37	9.94	10.69	10.69	10.69
	48	10.66	5.32	10.69	10.69	10.68
	64	10.69	10.69	5.35	10.69	10.70
	149	10.70	10.69	10.70	5.37	10.67
	161	10.70	10.70	10.70	10.70	5.37

interference because the two interfaces are different.) To further explore the relation between transmit power and interference, the transmit power of node A is set to 1 mW. This is in some ways similar to increasing the distance between two nodes because the received signal power will be decreased. This will decrease the SINR and possibly affect the throughput. The low transmit power results are shown in Table III. At low power, there continues to be interference between channels 36 and 48. Channels 149 and 161 now exhibit interference as well, but not before in Table II. This information must be taken into account when planning the locations of the nodes to ensure that these channels do no interfere with each other.

TABLE III Aggregate throughput (Mbps) of two transmissions using low power

		Channel 1				
Channel		36	48	64	149	161
Channel 2	36	5.38	9.76	10.70	10.70	10.70
	48	9.97	5.38	10.67	10.69	10.69
	64	10.70	10.69	5.39	10.70	10.70
	149	10.69	10.68	10.69	5.32	6.60
	161	10.60	10.57	10.59	6.96	5.33

2) Three interfaces: The number of interfaces per node is increased to three, of which two are sending and one receiving at node B. The SINR will decrease due to the added interference of the third interface. This may cause other pairs or triplets of channels to be unusable. The channel and transmission configuration is displayed in Figure 8.

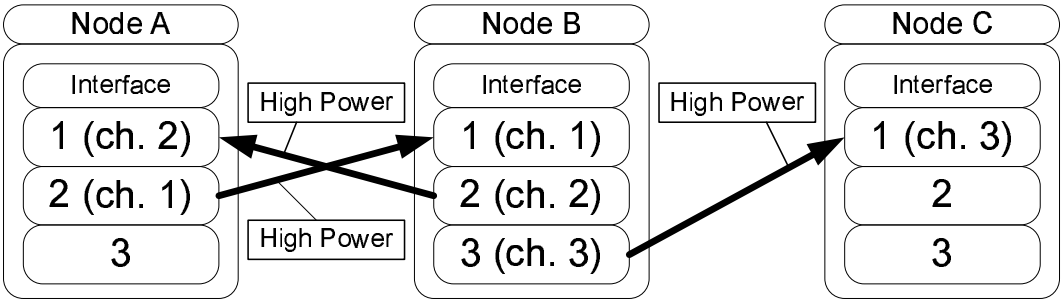


Fig. 8 Three interface configuration

The data is split into different tables according to the channel of interface 1. Tables IV-VIII are for interface 1 on channel 36, 48, 64, 149, and 161. Each channel is capable of approximately 5.37 Mbps of throughput when used individually. Aggregate throughput of 16 Mbps is necessary to show true orthogonality, but we slightly relax this requirement to 15 Mbps. A color coding scheme similar to that used in two interfaces is used:

white	Multiple use of the same channel
light grey	Aggregate throughput x $\geq$ 15Mbps
medium gray	Aggregate throughput x, 13Mbps $\leq$ x $<$ 15Mbps
dark gray	Aggregate throughput x $<$ 13Mbps

Cases where interface 2 and 3 are using the same channel are in the downward right diagonal. The row and column corresponding to the channel of interface 1 are also cases where reduced throughput is expected. There are 12 cases in each table where there might be full utilization of three different channels. The interference seen when using two interfaces on {36, 48} and {149, 161} continues to be a problem: every combination of these channels yields less than ideal throughput. The number of cases where the throughput is $\geq$ 15 Mbps is 6, 4, 8, 8, and 6 out of 12 for each of the Tables IV-VIII, respectively. This means that about half of the channel combinations have noticeable interference.

TABLE IV Aggregate throughput of three transmissions (interface 1 on channel 36)

36		Channel 2				
Channel		36	48	64	149	161
Channel 3	36	5.45	7.11	9.99	10.7	10.74
	48	10.67	8.77	13.66	16.00	16.02
	64	10.61	12.60	10.73	16.02	16.03
	149	10.66	12.32	15.97	10.65	10.86
	161	10.71	12.29	15.85	10.88	10.72

TABLE V Aggregate throughput of three transmissions (interface 1 on channel 48)

48		Channel 2				
Channel		36	48	64	149	161
Channel 3	36	10.71	10.6	12.02	13.62	13.57
	48	10.37	5.73	10.32	10.99	10.97
	64	10.90	10.74	10.70	16.03	16.02
	149	12.50	10.85	15.71	10.65	10.86
	161	12.14	10.85	15.94	10.87	11.15

TABLE VI Aggregate throughput of three transmissions (interface 1 on channel 64)

64		Channel 2				
Channel		36	48	64	149	161
Channel 3	36	10.71	10.91	10.88	16.00	16.03
	48	11.20	10.87	10.84	16.02	16.03
	64	10.86	10.49	5.55	10.91	10.85
	149	15.97	16.03	10.78	10.62	10.87
	161	15.97	16.00	10.78	11.18	10.72

TABLE VII Aggregate throughput of three transmissions (interface 1 on channel 149)

149		Channel 2				
Channel		36	48	64	149	161
Channel 3	36	10.72	11.05	15.03	10.69	16.02
	48	11.50	10.82	14.42	10.71	15.92
	64	15.17	12.62	10.74	10.47	15.98
	149	10.96	10.93	10.96	5.49	10.20
	161	15.99	15.99	15.91	10.58	10.70

TABLE VIII Aggregate throughput of three transmissions (interface 1 on channel 161)

161		Channel 2				
Channel		36	48	64	149	161
Channel 3	36	10.70	11.19	14.99	15.80	10.67
	48	11.54	10.89	14.66	16.02	10.67
	64	14.93	13.40	10.74	16.00	10.67
	149	15.92	16.03	15.86	10.60	10.65
	161	10.87	10.93	10.86	10.18	5.35

Choosing three channels from a set of five yields ten combinations. Only two of these combinations {36, 64, 149} and {36, 64, 161} yielded little to no interference as seen in Table IX. This ultimately means that there are only three 802.11a channels that can be used simultaneously in a three interface node to ensure close to interference-free operation.

F. Summary

The mini-PCI interfaces exhibited severe adjacent channel interference in our office while the PCMCIA interfaces did not. Adjacent channel interference on the PCMCIA interface was not as bad as same channel interference. The packets transmitted tell us that the mini-PCI was not deferring transmission while the PCMCIA deferred. This explains why adjacent channel interference results in poor performance and verifies what Chereddi suspected [1]. Tests in the wireless wind tunnel yielded different results, showing that the environment also makes a big difference.

Tests with two interfaces show that channels {36, 48} and {149, 161} interfere with each other. Further tests with three

TABLE IX Interference check of channel combinations when using three interfaces

Channel Combination	Table IV		V		VI		VII		VIII		Little/No Interference
{36,48,64}	x	x	x	x	x	x					
{36,48,149}	x	✓	x	x			x	x			
{36,48,161}	✓	x	x	x					x	x	
{36,64,149}	✓	✓			✓	✓	✓	✓			✓
{36,64,161}	✓	✓			✓	✓			✓*	✓*	✓
{36,149,161}	x	x					✓	✓	✓	✓	
{48,64,149}			✓	✓	✓	✓	x	x			
{48,64,161}			✓	✓	✓	✓			x	x	
{48,149,161}			x	x			✓	✓	✓	✓	
{64,149,161}					x	x	✓	✓	✓	✓	

✓: Throughput $\geq$ 15 Mbps ✓* : Throughput $\geq$ 14.9 Mbps

x : Throughput $<$ 14.9 Mbps

interfaces in each node show that in order to achieve 90% of maximum throughput, only channels {36, 64, 149} or {36, 64, 161} can be used simultaneously.

IV. VOIP ON NET-X

With the advent of faster internet connections, real-time multimedia applications on the internet are gaining popularity as a means of communicating with people all over the world. Two popular forms of real-time multimedia applications on the internet are voice-over-IP (VoIP) and internet TV. VoIP enables voice calls to be made around the world to anyone with a computer and internet connection. Internet TV allows the user to watch live TV on a computer, eliminating the need for cable service. The public has readily embraced VoIP as we have seen with the popularity of Skype. In this section, we will focus on VoIP and determine the suitability of the Net-X framework in meeting the needs of VoIP.

A. VoIP Requirements

The public switched telephone network (PSTN) is known to be quite reliable. It is rare for a call to be dropped and you do not have to worry about your telephone connection being too slow. Unfortunately, the success of VoIP depends not only on having an internet connection, but also on the characteristics of that connection. Unlike web browsing, real-time applications have specific deadlines to meet. The ultimate determination of call quality is subjective and depends on the user.

The International Telecommunication Union (ITU) defines a standard detailing the requirements of a data connection to ensure a high quality voice call. ITU-T G.114 recommends that the one-way delay not exceed 150 ms [10]. For international calls, delays up to 400 ms are acceptable. This recommendation is a mouth-to-ear delay which includes time to record, encode, transmit, receive, decode, and play the audio. Other factors include packetization, queuing, and delay due to the jitter buffer. Goode provides a sample "delay budget," breaking down the various factors that contribute to the delay, and finds that it can take 120 ms for these other factors, leaving approximately 30 ms of delay for the backbone network [11].

VoIP is usually implemented over the user datagram protocol (UDP) which does not guarantee that a packet will arrive at the destination. This is in contrast to the transmission control protocol (TCP) that provides reliable in-order delivery of packets. While the reliability feature of TCP is nice, it is often unnecessary for real time applications. Losing a small snippet of voice or a couple packets of a video stream does not seriously affect the result. Having to retransmit as in TCP may also force the call to be paused, which makes the phone call seem half-duplex. Various forms of packet loss concealment are employed to mitigate the effects of isolated packet losses. In general, a loss rate of up to 4% can still be of acceptable quality [12].

Thus we wish to see experimentally if the following two criteria are met:

1) Delay $\leq$ 30 ms
2) Loss rate $\leq 4\%$

B. Experimental Setup

1) Traffic characteristics: VoIP is a broad term that refers to any type of voice call placed over a packet switched network. The difference in quality among the different types of VoIP offered by different companies is primarily due to the choice of audio codec. There are many free and proprietary audio codecs with which to encode the audio. More advanced codecs have features such as packet loss concealment and voice activity detection. Packet loss concealment attempts to conceal the loss of packets to the user by replacing the lost packet(s) with the last packet or attempting to interpolate the values of the last packet. Voice activity detection is a technique where no packets are sent during silent periods.

One of the oldest and most popular audio codecs is the G.711 codec. The G.711 codec uses pulse code modulation and either A-law or μ-law encoding [13]. No compression is utilized in this codec, meaning that its bandwidth requirement is

greater than that of others. We choose to emulate a standard G.711 codec that operates at 64 kbps, with 10 ms of audio data in each packet. This type of traffic is generated by the Distributed-Internet Traffic Generator (D-ITG) [14]. D-ITG is a traffic generation tool that is capable of logging the sending and receiving time of a packet. Through this feature, we are able to calculate the one-way delay of the packets. We simulated a 60 s VoIP phone call using D-ITG. Each experiment was averaged over five trials.

2) Calculating delay: To calculate the one-way delay in our testbed, it was necessary to synchronize the clocks of all our systems. The Soekris net4521 boards on which the system is implemented utilize a nonstandard crystal frequency, causing the clock to run at an incorrect rate. In our measurements, the clock is slow by approximately 410 ms after merely 120 s. The delay measurements we need to make are on the millisecond level, and having this large of a discrepancy is unacceptable. Typically, the fix is to set up a network time protocol (NTP) server and have all nodes set their clocks with a NTP daemon. This method works for clocks that are slightly off, but our clock skew is too severe. To fix this problem, the clocks are synchronized five times a second.

C. Results

1) One-hop results: We simulated a one-hop unidirectional VoIP flow and display the results in Table X and Figure 9(a). (All figures for Section IV appear in Subsection IV-G.) On average, the delay experienced by a packet is 2.2 ms. The cumulative distribution function shows that the vast majority of the packets experience a delay of less than 10 ms. A couple packets are delayed by approximately 60 ms every 5 s. This periodic delay is due to packets from the Hello Message subsystem which are generated every 5 s. The switchable interface is used to send "hello" packets on four channels (the "hello" packet for the fifth channel is sent using the fixed interface). Of the four channels, one channel will be the channel on which our VoIP packets are currently being transmitted, leaving three extra channels to service. Due to the design of spending at least 20 ms on each channel to average out the cost of switching, we will see a 60 ms delay every 5 s.

A test of five simultaneous unidirectional flows was carried out to test the capability of the system in supporting multiple flows of small packets. As shown in Table X, there is a slight increase in average delay to 3.8 ms. This increase in delay may be due to queuing delay, since the application creates multiple packets at or near the same time but must transmit them one by one. The characteristics of the delay in Figure 10(a) are similar to that in the one flow case.

A VoIP call, similar to a telephone call, consists of two parties communicating over a channel and thus requires flows in both directions. With a single flow in both directions, the average delay increases to 2.7 ms as shown in Table X. In the usual single-channel ad-hoc network, it makes sense for one-hop bidirectional traffic to experience higher delay than one-hop one-way traffic. Since both nodes are transmitting, they must take turns and share the channel. In our multichannel case, the two nodes transmit on orthogonal channels that do not interfere with each other. The increase in time could be due to CPU processing. Each node must transmit and receive 100 packets/s. Instead of dedicating all resources to transmission, time must be taken to process reception as well. Running five flows in each direction also increases the delay to 5.7 ms. The delay characteristics can be seen in Figures 11(a) and 12(a).

From these results, we see that the multichannel multi-interface protocol is capable of supporting one-hop VoIP traffic.

TABLE X Average delay for one-hop VoIP traffic

Number of Flows	Average Delay (ms)
1 - one way	2.24
1 - two way	2.71
5 - one way	3.86
5 - two way	5.76

2) Two-hop results: We then expanded to a two-hop route using three nodes and simulated VoIP traffic between the end-nodes. The results for unidirectional traffic can be found in Table XI, Figure 13(a), and Figure 14(a). The average delays are still fairly small being just 4.3 ms and 5.2 ms for one and five flows respectively. There is some packet loss when we have five flows along this two-hop path, but still within the tolerance of VoIP. Due to the addition of another node along the path, we see two spikes in delay every 5 s: one due to the channel switching mechanism of the source node and another due to the intermediate node. Synchronization of the channel switching would cause the delay to add up to over 100 ms. Although still currently well within the 400 ms of acceptable VoIP traffic, this can become problematic with more channels.

For two-way traffic, we again see the two spikes of delay in Figure 15(a). The average delay is greater as there are more packets to transmit and also because the intermediate node now has to switch between two channels in order to service both flows. Assuming a uniform distribution of packets in time (one packet per ten ms, each for two flows), there will be a 7.5 ms delay due to having to switch between two channels at the intermediate node. (This also assumes waiting at least 20 ms on each channel.) Add to this the additional delay due to backoff and the average delay 12.9 ms is reasonable.

TABLE XI Average delay for two-hop VoIP traffic

Number of Flows in each direction	Average Delay (ms)	Drop Rate
1 - one way	4.33	0
1 - two way	12.91	0
5 - one way	5.17	0
5 - two way	29.20	0.1433%

Finally, we extend the experiment to five flows in each direction, with the results shown in Table XI and Figure 16(a). The average delay is around 29.2 ms and the drop rate is only 0.14%. This loss rate can be masked using some loss concealment technology. Note that we are only transferring 350 kbps in each direction and 500 packets/s. 802.11a/b/g is inherently bad for small packets due to overhead, since small packets are not buffered up and transmitted as a large packet. Instead, the system must send a packet, wait for an ACK, and then send the next packet. The achievable throughput of the system is then lower when we have small packets. If the packet size were larger with the same throughput rate, the delay and loss characteristics should be better.

These two-hop results show that the multichannel multi-interface protocol is capable of meeting the drop rate and delay requirements of VoIP in a two-hop setting. Expanding beyond two hops will create multiple bottlenecks that will undoubtedly increase the delay to more than the 30 ms recommendation. The addition of more simultaneous flows will also impact the delay.

D. Reducing the Delay

Through these experiments, we have seen that the channel switching mechanism artificially introduces delay into the system. The majority of the delay we are witnessing is due to the requirement of staying on a channel for at least 20 ms. Lowering the requirement to 10 ms or completely getting rid of the requirement would improve the delay time. The requirement was instituted because of the long channel switching time of the wireless cards. Changing to wireless cards that have a shorter channel switching time would reduce this need. Unfortunately, a quick channel switch is usually not required in 802.11 networks because the majority of networks are single channel. Manufacturers do not readily list the switching time in their specifications. At this time, we have not found a suitable wireless card that can switch channels quickly.

One obvious method of reducing the delay would be to reduce the minimum amount of time to stay on the channel. Reducing the time from the current setting of 20 ms to something lower would help. Varying the minimum time to spend on a channel depending on the history of the average queue length might improve performance. This would reduce the amount of idle time on a channel while still attempting to amortize the channel switching cost. We were unable to reduce the minimum time to 10 ms because the function responsible for transmission is not always called within 10 ms. Another problem is the 10 ms resolution of the Linux kernel timer, which prevents the use of some intermediate length of time. Without modification to the kernel, 20 ms is the least amount of time to stay on a channel.

Since we cannot reduce the delay associated with each channel switch, we will instead try to reduce the occurrence of unnecessarily long delays. Long delays are created due to the “hello” packets being transmitted consecutively on all the channels. An interface that services four channels must switch to three other channels and spend at least 20 ms on each channel before returning to the loaded channel, thus causing a delay of 60 ms. Staggering the transmission of “hello” packets prevents this phenomenon from occurring. Another method that achieves the same result is to service the channels differently. Instead of servicing the channels in round-robin fashion, we will service more often the channels that have a higher average queue length. This is done in lieu of staggering the creation of the “hello” packets due to the overall benefit of servicing queues that have a longer average length. Thus, this strategy could work with all types of traffic and not be limited to reducing the delay due to “hello” packets. This assumes that a channel with few packets queued will not have delay sensitive packets.

E. Delay Reduction Scheme

The original algorithm used to find the next channel to switch to utilizes a round-robin scheme, which is presented here in pseudocode:

```
1  start_at = 0
2  if (current_channel >= 0 && current_channel < num_channels)
3     start_at = (current_channel + 1)%num_channels;
4  for( i = start_at; i <num_channels; i++){
5     if (queue for channel i not empty)
6        return i;
7  }
```

```
 8  for(i = 0; i < start_at; i++) {
 9     if (queue for channel i not empty)
10        return i;
11  }
```

This is quick to compute, but may not be optimal for delay. No extra information is used to make an intelligent decision on the next channel to service. One piece of information that could be helpful is the average queue length of each channel which is updated each time the channel is serviced, similar to a running average. The update equation is:

$$\text{avg\_queue\_length (packets)} = 0.5 \times \text{avg\_queue\_length} + 0.5 \times \text{queue\_length} \quad (1)$$

The new algorithm to find the next channel will categorize a channel as heavily or lightly loaded by making use of the average queue length. Using this categorization, one of the lightly loaded channels will be serviced after all of the heavily loaded channels have been serviced. The algorithm is presented below in pseudocode:

```
 1  for (each channel i){ // categorizes the channel as high load,
 2                        // low load, or no load
 3     if (queue for channel i not empty) {
 4        if (average_queue_length of channel i > 1)
 5           set high load=TRUE and low load=FALSE for channel i
 6        else
 7           set high load=FALSE and low load=TRUE for channel i
 8     }
 9     else {
10        set high load=FALSE and low load=FALSE for channel  i
11     }
12  }
13  if (current channel is high load) {
14     for (i = current_channel+1; i < num_channels; i++)
15        if(channel i high load == TRUE)
16           return i;
17     for (i = previous_low_channel+1; i < num_channels; i++)
18        if(channel i low load == TRUE)
19           return i;
20     for (i = 0; i < previous_low_channel+1; i++)
21        if(channel i low load == TRUE)
22           return i;
23     for (i = 0; i < current_channel; i++)
24        if(channel i high load == TRUE)
25           return i;
26  }
27  else {  //current channel is low load
28     for (i = 0; i < num_channels; i++)
29        if(channel i high load == TRUE)
30           return i;
31     for (i = previous_low_channel+1; i < num_channels; i++)
32        if(channel i low load == TRUE)
33           return i;
34     for (i = 0; i < previous_low_channel+1; i++)
35        if(channel i low load == TRUE)
36           return i;
37  }
```

Lines 1-12 categorize the currently used channel as high, low, or no load. If the load of the current channel is high, then the next channel is found through code in lines 13-26. Using the index *i*, attempt to find the next greater index for a channel that has high load. If none is found, all high load channels have been serviced and it is time to find a low load channel (line 17). The previous_low_channel l_c keeps track of the previous low load channel serviced and we start searching at $l_c + 1$. Without this variable, the low load channel with the smallest index will always be serviced and channels with a high index will be

starved. If there are still no results, then we wrap around and search for a low load channel starting from the beginning (line 20). Lastly, a lack of results means there are no low load channels, so another high load channel should be serviced (line 23). The algorithm for the case when the current channel is low load is similar.

We carried out the four different VoIP tests as before and display the results below in Tables XII and XIII. The one-hop one-flow results are similar in both schemes, probably due to low number of packets being transmitted. A low number of packets to transmit translates to a low number of packets which can be affected by the new channel switching scheme. Thus, the average delay is mostly unaffected. There is actually a reason that packets would experience higher delay in the new scheme when there is only one flow being transmitted from a node. The new scheme results in more switches, which means less transmission time. However, the increase should be minimal in this case and so the increase/decrease we witnessed is likely to be circumstantial. A graph of the delay can be found in Figure 9(b) and 11(b). Notice that the periodic 60 ms delay no longer exists, as we distributed the delay. Instead, there are 20 ms and 40 ms spikes in delay, in addition to the occasional 60 ms spike.

TABLE XII Average delay for one-hop VoIP traffic for both schemes

	Old Scheme		New Scheme	
Number of Flows	Avg. Delay (ms)	Drop Rate	Avg. Delay (ms)	Drop Rate
1 - one way	2.44	0	2.09	0
1 - two way	2.38	0	2.71	0
5 - one way	3.86	0	2.71	0
5 - two way	5.76	0	4.51	0

TABLE XIII Average delay for two-hop VoIP traffic for both schemes

	Old Scheme		New Scheme	
Number of Flows	Avg. Delay (ms)	Drop Rate	Avg. Delay (ms)	Drop Rate
1 - one way	4.33	0	4.65	0
1 - two way	12.91	0	12.10	0
5 - one way	5.17	0	5.08	0
5 - two way	29.20	0.1433%	24.90	0.13%

The one-hop five-flow results are more interesting since we now have many packets to transmit and we might actually see some improvement here. In both one-way and two-way cases, the average delay is reduced by 29.6% and 21.7% respectively (Figures 10(b) and 12(b)). Moving on to two-hop traffic, we again see little change for a single flow of traffic (Figures 13(b) and 15(b)). There is also little change (Figure 14(b)) for five-flow one-way traffic. There is a decrease of 14.7% in average delay for five-flow two-way traffic (Figure 16(b)). The drop rate slightly decreased as well. From this data, it seems the delay sensitive channel switching scheme works just as well, if not better than the round-robin channel switching scheme.

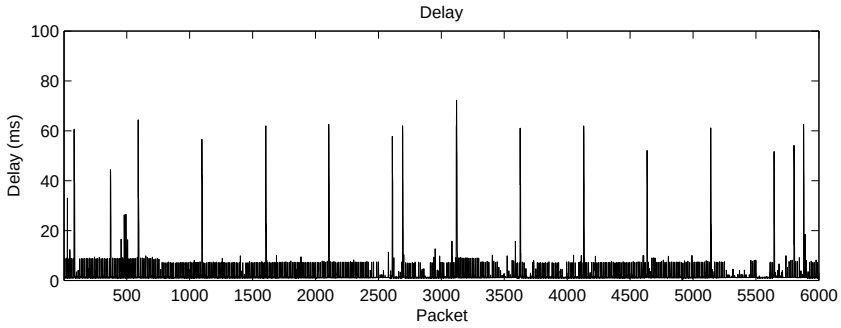
F. Summary

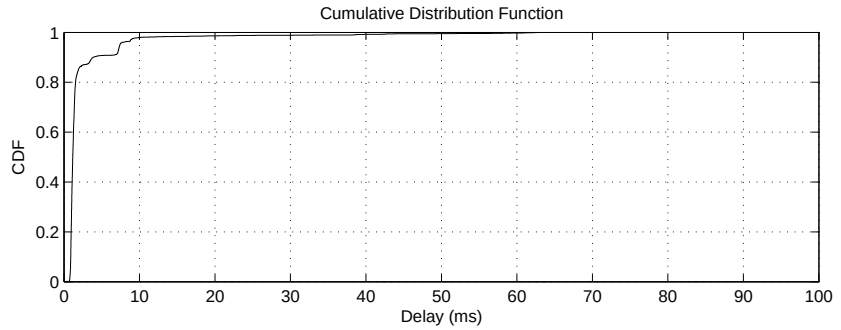
We simulated VoIP traffic on the Net-X testbed and found the performance characteristics to be satisfactory for one- and two-hop communication. However, the average measured delay for two-hop communication is near the suggested limit, so we proposed a delay reduction scheme that services certain channels more often. Channels that have historically had more packets are given priority. This succeeded in reducing the average measured delay from 29.20 ms to 24.90 ms. Communication across three or more hops will result in poor performance.

G. Graphs of Delay Results

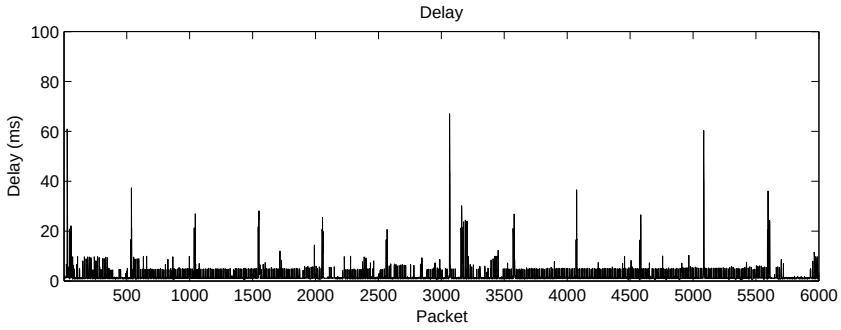
V. SUPPORT FOR MULTIPLE INTERFACES

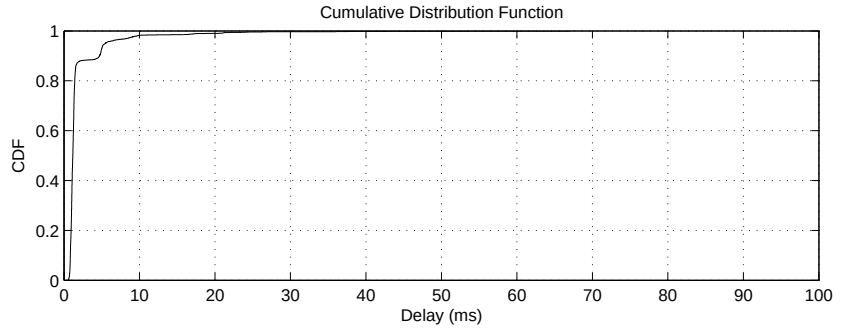
The purpose of using multiple interfaces in the Net-X testbed is to increase spatial reuse of the channels. With the addition of a second interface, results have been higher throughput compared to a single-interface, single-channel network. However, as we will show later, one important case where the performance increase has been small is in the area of TCP throughput. Many applications on the internet utilize Transmission Control Protocol (TCP). The advantages of TCP include reliability and automatic tuning of load in response to network conditions. TCP accomplishes this by using feedback from the destination. The sender will tune the amount of data sent in response to the receiver's acknowledgments. The services provided by TCP are vital to many applications, making it vital to assess and improve the performance of TCP traffic in our testbed. We add a third interface at each node for this purpose. Adding a third interface also has the potential of improving UDP performance.





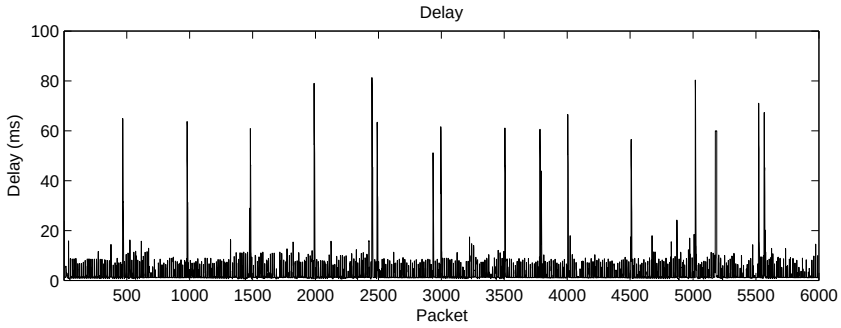
(a) Round Robin Channel Switching

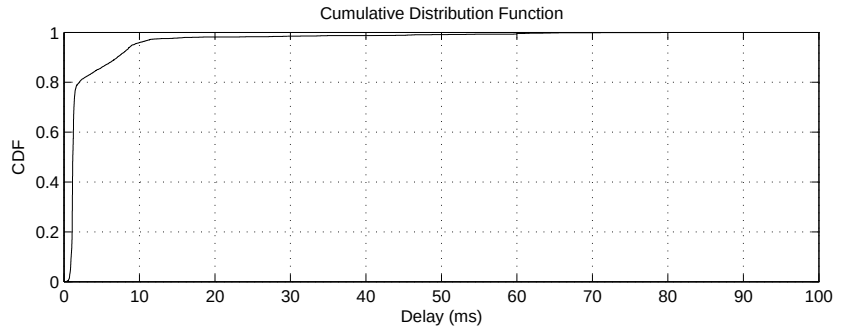




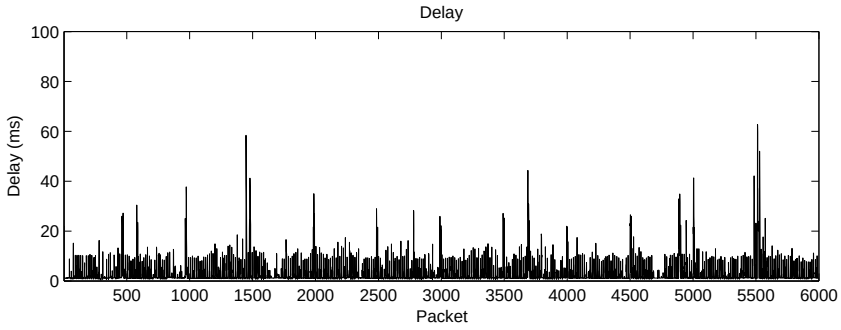
(b) Delay Sensitive Channel Switching

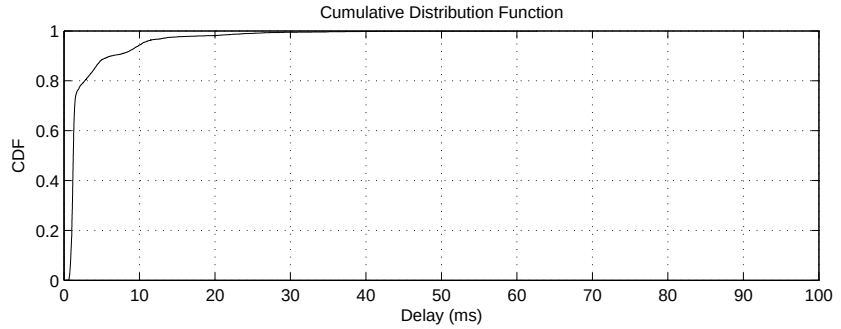
Fig. 9 Delay of 1 one-way one-hop flow





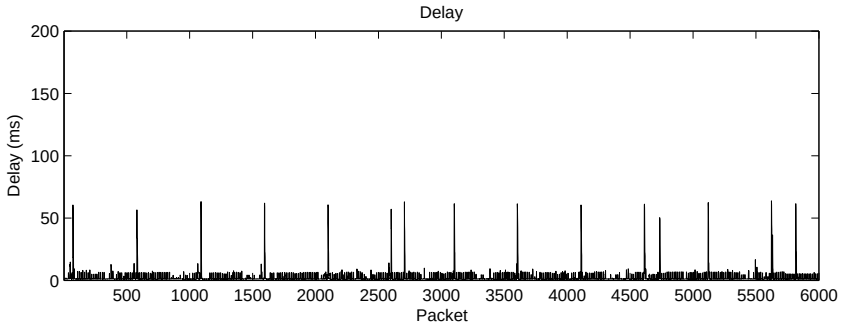
(a) Round Robin Channel Switching

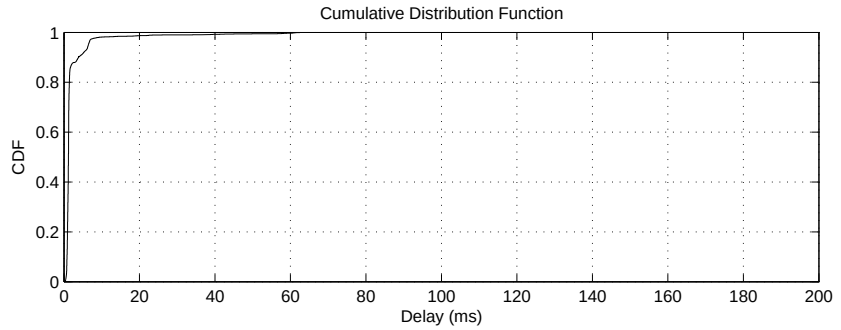




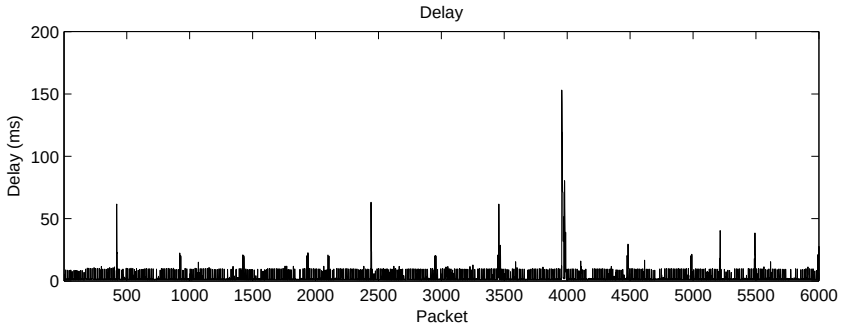
(b) Delay Sensitive Channel Switching

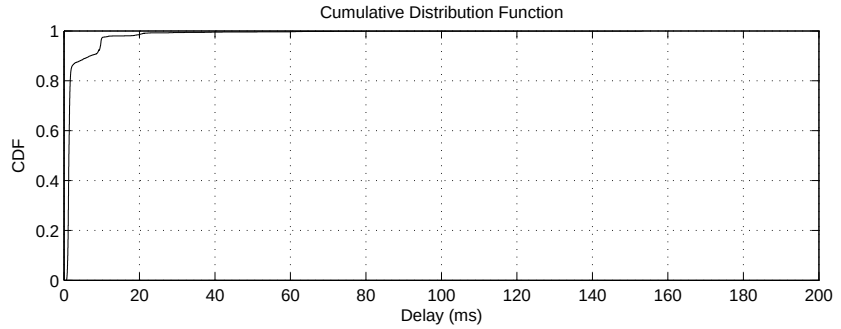
Fig. 10 Delay of five simultaneous one-way one-hop flows (1 flow shown)





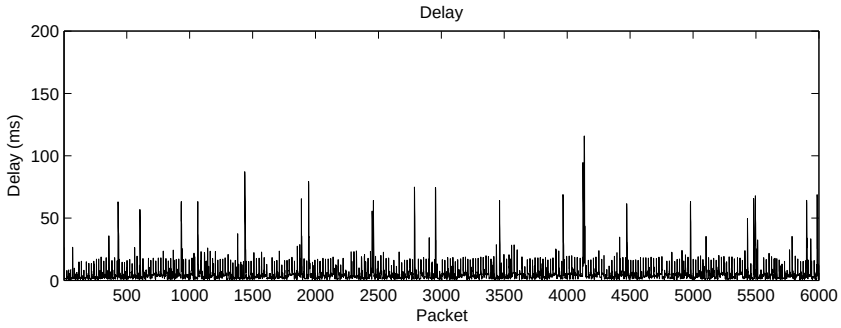
(a) Round Robin Channel Switching

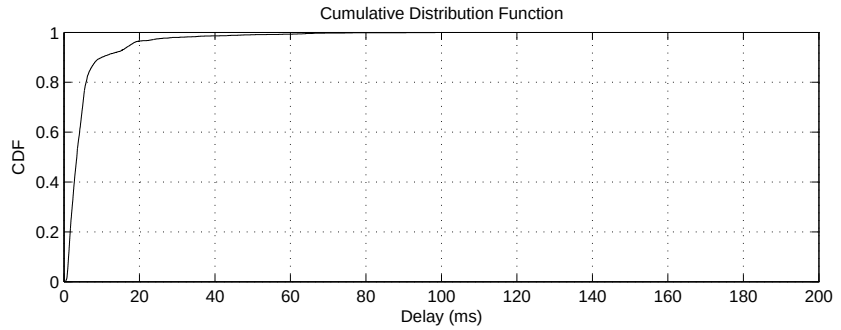




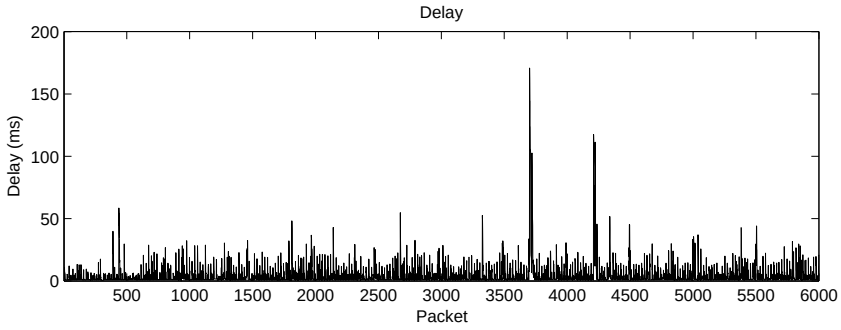
(b) Delay Sensitive Channel Switching

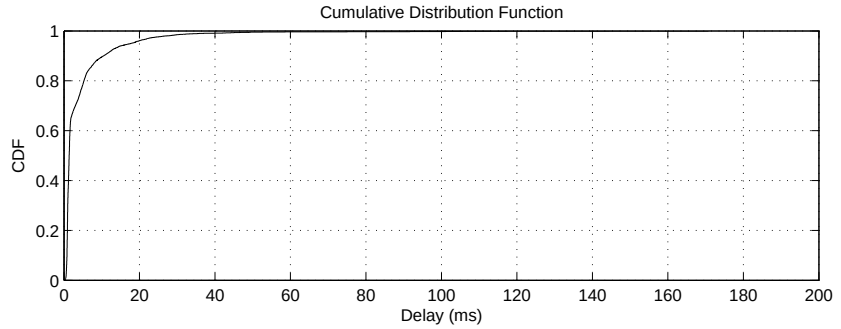
Fig. 11 Delay of 1 two-way one-hop flow





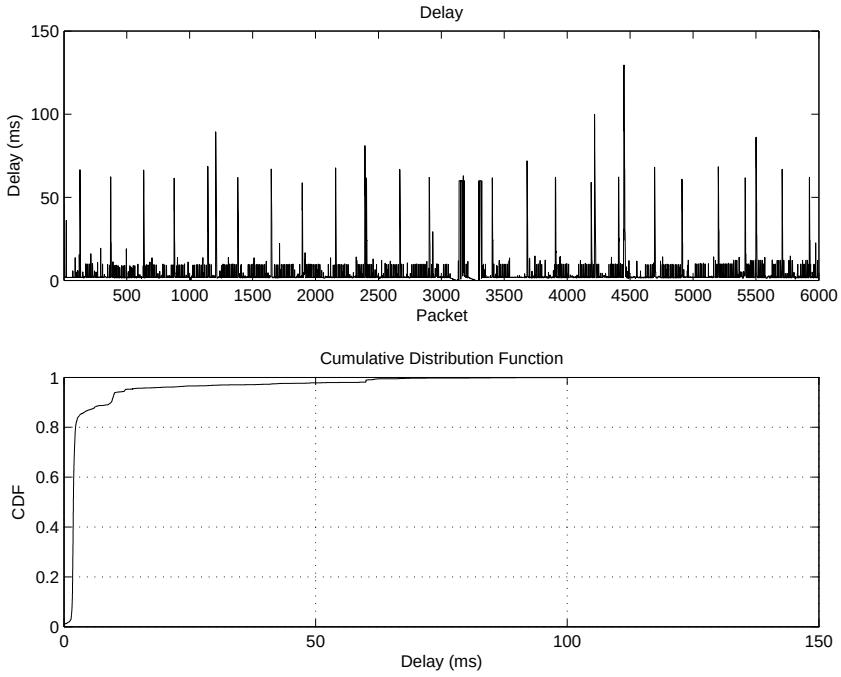
(a) Round Robin Channel Switching



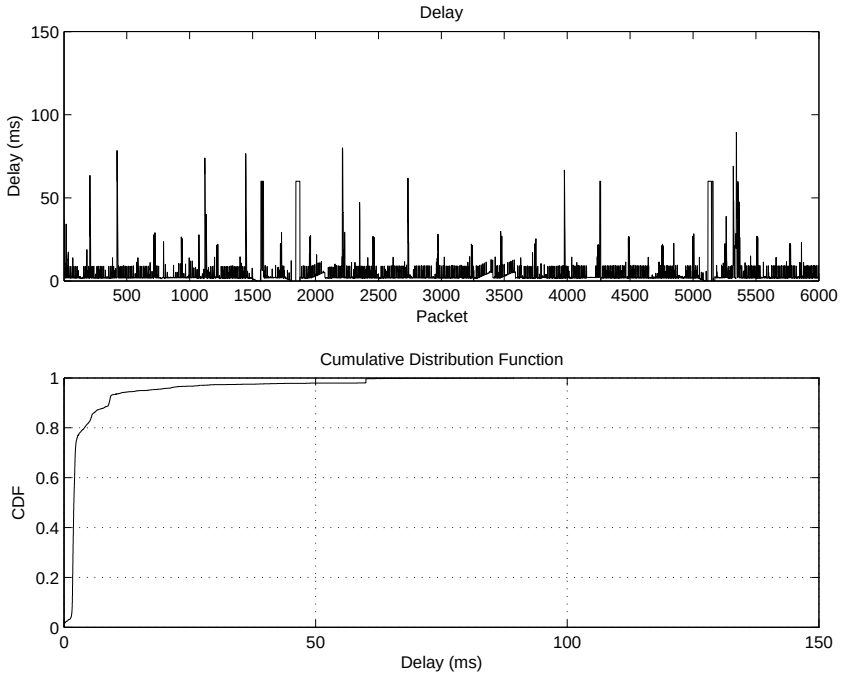


(b) Delay Sensitive Channel Switching

Fig. 12 Delay of five simultaneous two-way one-hop flows (1 flow shown)

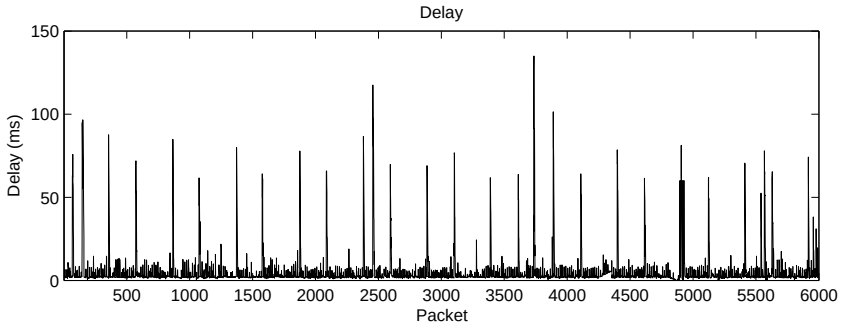


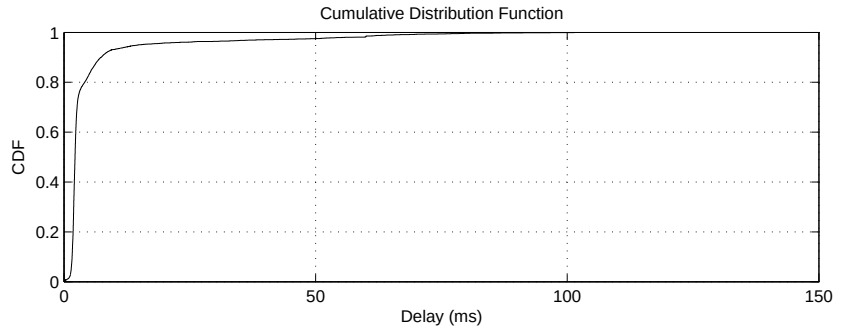
(a) Round Robin Channel Switching



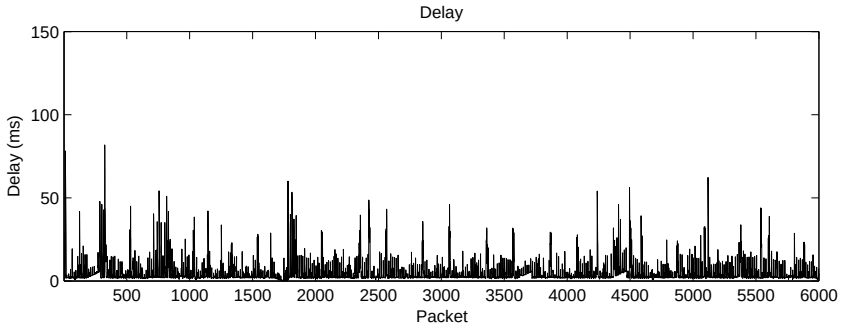
(b) Delay Sensitive Channel Switching

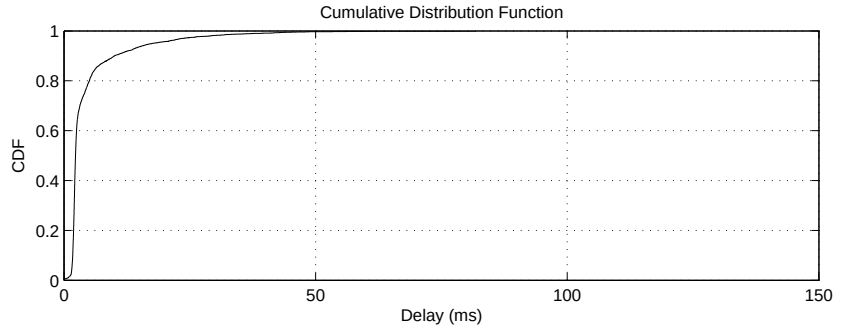
Fig. 13 Delay of 1 one-way two-hop flow





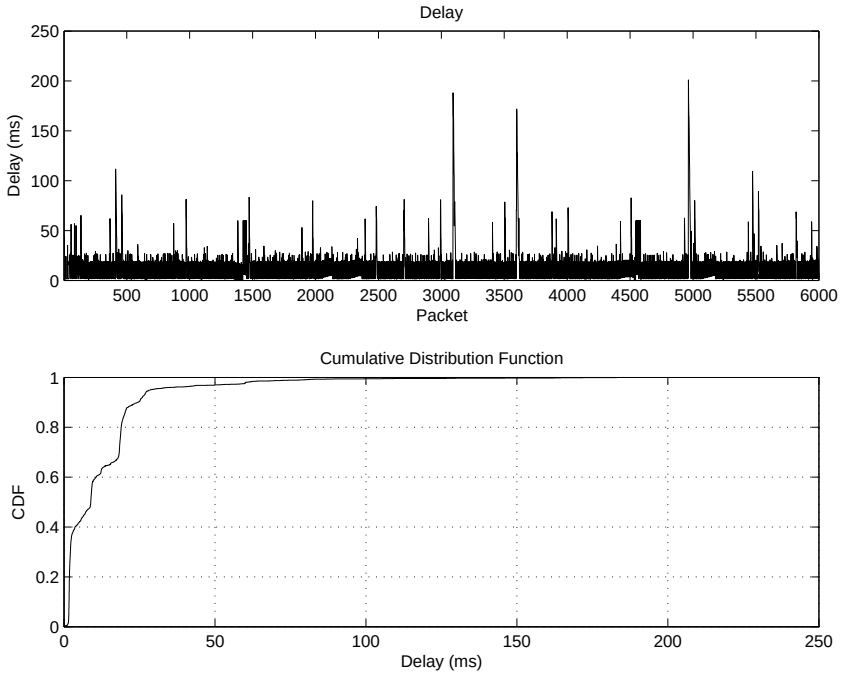
(a) Round Robin Channel Switching



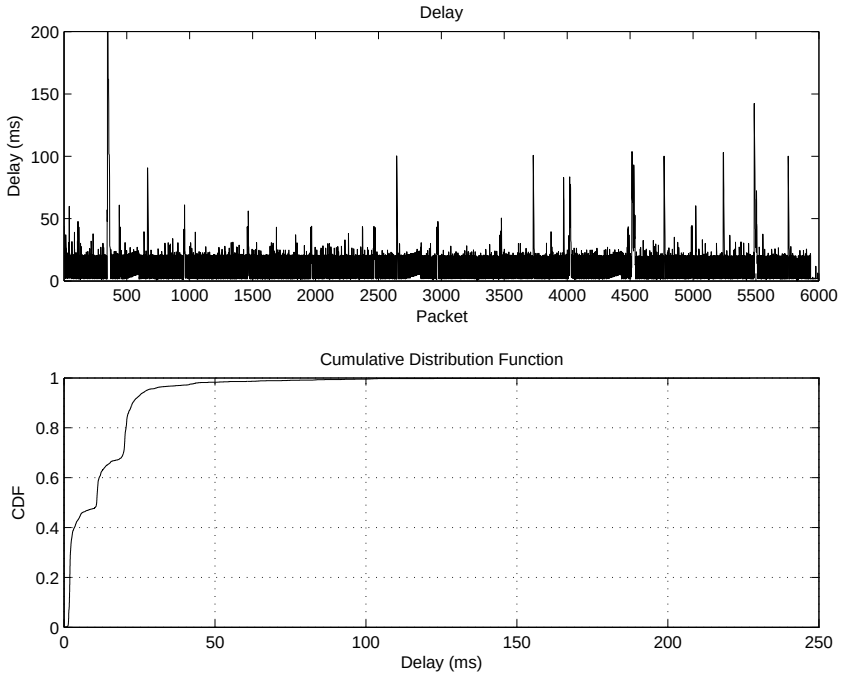


(b) Delay Sensitive Channel Switching

Fig. 14 Delay of five simultaneous one-way two-hop flows (1 flow shown)

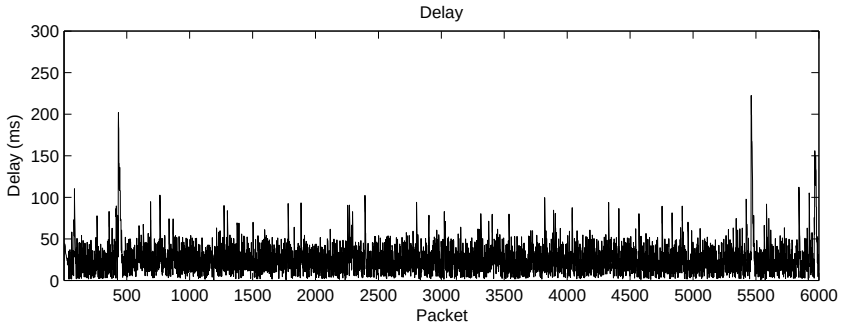


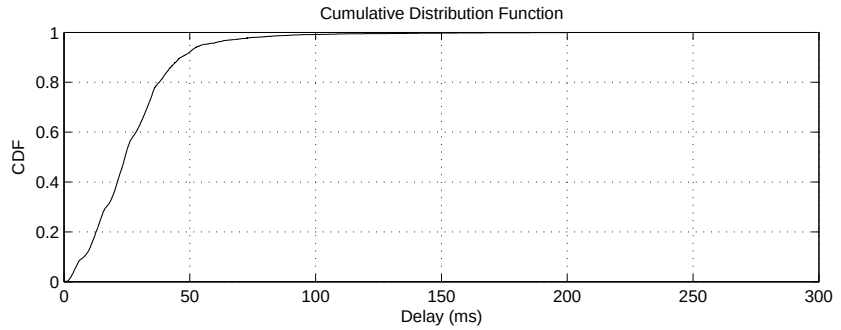
(a) Round Robin Channel Switching



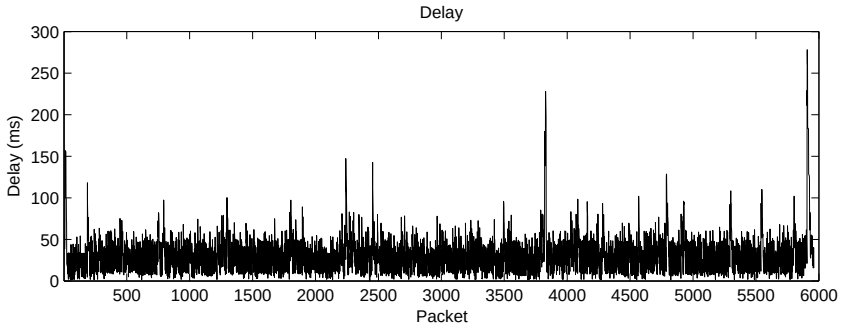
(b) Delay Sensitive Channel Switching

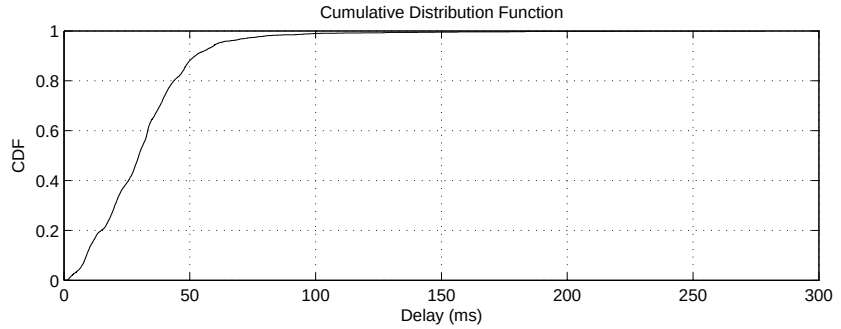
Fig. 15 Delay of 1 two-way two-hop flow





(a) Round Robin Channel Switching





(b) Delay Sensitive Channel Switching

Fig. 16 Delay of five simultaneous two-way two-hop flows (1 flow shown)

A. Previous Work

Previous experiments by Bhardwaj have shown that single-hop TCP traffic performs well in single-interface and multi-interface multichannel networks [15]. However, moving to multihop networks has shown in the general case that using two interfaces fails to solve all the problems faced in single-channel networks. Bhardwaj suggested that this was due to intermediate nodes having to alternate sending TCP DATA and TCP ACK packets, resulting in less time on a channel to transmit TCP DATA packets. He further found that TCP throughput is good in cases where the return route does not share any intermediate nodes with the sending route. In this scenario, each of the nodes only has to send packets to one other node, instead of having to switch back and forth between two channels in order to send the TCP DATA and TCP ACK. A routing protocol that attempts to set up forward and reverse routes that do not share a common intermediate node will solve this problem, provided that there is no other traffic using other channels but sharing the routes. Another suggestion made by Bhardwaj was to utilize a third interface to avoid the cost of having to switch an interface among two different channels. The addition of the third interface was undertaken in hopes of solving this problem.

B. Implementation Issues

In the Net-X system, interfaces are defined as "fixed" or "switchable" [1]. A "fixed" interface is primarily used for receiving since it cannot change to its neighbor's channel. We decided to use the third interface as a "switchable" interface to solve the problem mentioned above.

The designers of the Net-X framework saw the need for support of numerous interfaces and most of the code was easily ported. No changes were necessary in the kernel multichannel routing (KMCR) module. However, it was necessary to make changes in the hybrid multichannel protocol to support the third interface. In the current implementation, all of the channels are added as valid channels for each of the interfaces. A problem arises when attempting to add a routing entry because the algorithm will search for the first interface that supports the desired channel. This will leave the third interface totally unused as the second switchable interface will always return a positive result. There are two possible fixes for this: (1) assign a subset of channels to each of the switchable interfaces or (2) load-balance the assignment of routes to interfaces. In both schemes, all channels will be added to the fixed interface as valid channels. All of the channels can then be assigned disjointly to the remaining interfaces. For example, with three interfaces (one fixed f_1 and two switchable s_1, s_2) and five channels (c_1, c_2, c_3, c_4, c_5), we can assign (c_1, c_3, c_5) to s_1 while assigning (c_2, c_4) to s_2. Underutilization becomes a problem if transmissions are only happening on channels c_1 and c_3. It might be possible to design the hybrid protocol so that the fixed channel of adjacent nodes will not result in this formation, but that complicates the protocol and unnecessarily constrains the possible channels to use.

A better scheme is to load-balance the assignment of channels to interfaces. All channels are valid channels for each of the interfaces. The first routing entry is assigned to interface s_1. If another routing entry is added with the same channel as before, then it will also be assigned to interface s_1 as well. Otherwise, the second entry is assigned to interface s_2. The basic algorithm is to assign the channels equally to the available switchable interfaces. There will be underutilization if all of a node's routes have to go through the same neighbor, but the scheme does no worse than a two-interface protocol.

Another issue to think about is the interface to use when sending broadcast packets. In the two-interface case, all channels except the fixed channel are assigned to the switchable interface. With three interfaces, there are two distinct possibilities: fixed and adaptive. In the fixed scheme, we can either fix all of the remaining channels on one of the switchable interfaces, or a subset on interface s_1 and the rest on interface s_2. The latter scheme will equally distribute the delay associated with the broadcast packets on all the interfaces, cutting down on the maximum delay. However, for a highly time-sensitive application, it could be advantageous to assign all of the non-delay-sensitive broadcast packets to be sent on one interface, and then using the second interface for the time sensitive application. In the adaptive case, it is possible to assign broadcast packets for channel c_1 to the same interface that unicast packets will use. In the event of no unicast packets on the corresponding channel, the most recently used interface for that channel can be used for sending out broadcast packets.

The final implementation uses the load balancing method along with the adaptive broadcast interface scheme. A quick summary of how to assign routes is as follows:

- Initialization: Broadcast channels assigned as equally as possible among switchable interfaces.
- Unicast channels assigned as equally as possible among switchable interfaces if no route to channel exists. Each time a unicast channel c is assigned to an interface i, assign broadcast channel c to interface i as well. If a route to a channel already exists, then use the same route. Routes are lost after 10 s of non-use. Each channel is assigned to at most one switchable interface.

C. Setup

We set up routes of various lengths in order to test the usefulness of the third interface. As mentioned before, the third interface only becomes useful in a multihop network, or if there are transmissions for two different channels from the same node. Wireless connections are highly dependent on the environment, so a single-hop route may become a two-hop route and

vice versa. We utilized a software firewall (*iptables*) to drop packets from nodes that are not supposed to be adjacent for our tests. This fixes the problem of varying wireless conditions and also helps establish multihop routes. The neighbor tables of the hybrid multichannel protocol are allowed to populate and then frozen after 3 min. At this time, the firewall is turned off. In this manner, we can ensure that a two-hop route will stay at two hops, even if channel conditions temporarily improve so that there is a direct one-hop link between two nodes. A total of five nodes are used, with node connectivity as shown in the link graph in Figure 17.

When testing with a single interface, all nodes use the same channel. Static routes are set up to allow multihop communication. The dual and triple interface tests run the HMCP [1] on top of the Net-X framework. The fixed interface of each node is set to a different channel from the set $\{36, 48, 64, 149, 161\}$. In addition, a traceroute is performed immediately prior to each trial so that route discovery delay is avoided. Each of the cases was run for five trials of 60 s each. As before, *iperf* is used to generate traffic. To provide a basis for comparison, the same tests are run with a single-channel network and the two interface hybrid multichannel protocol, as well as the new three-interface implementation.

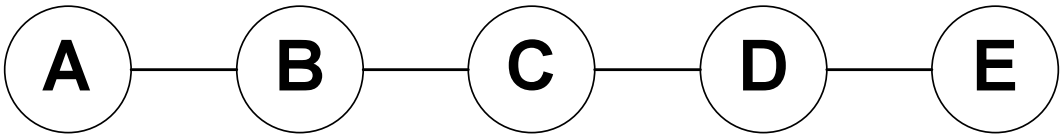


Fig. 17 Node connectivity

D. TCP Results

The results in Figure 18 show the poor throughput achieved when using a single interface. Since all the nodes must share the same channel, multihop transfers will have throughput upper-bounded by the single-hop throughput divided by the number of hops. Using the two interface HMCP results in higher throughput, but the dropoff is rapid as we increase the number of hops. The throughput is consistently higher than the single interface case by about 500 kbps. The triple interface implementation shows greatly improved throughput over all multihop cases. The three interface throughput is $1.56\times$, $2.30\times$, and $2.72\times$ that of the two interface throughput for the two-, three-, and four-hop flows respectively. This shows that the third interface makes a big impact on TCP performance.

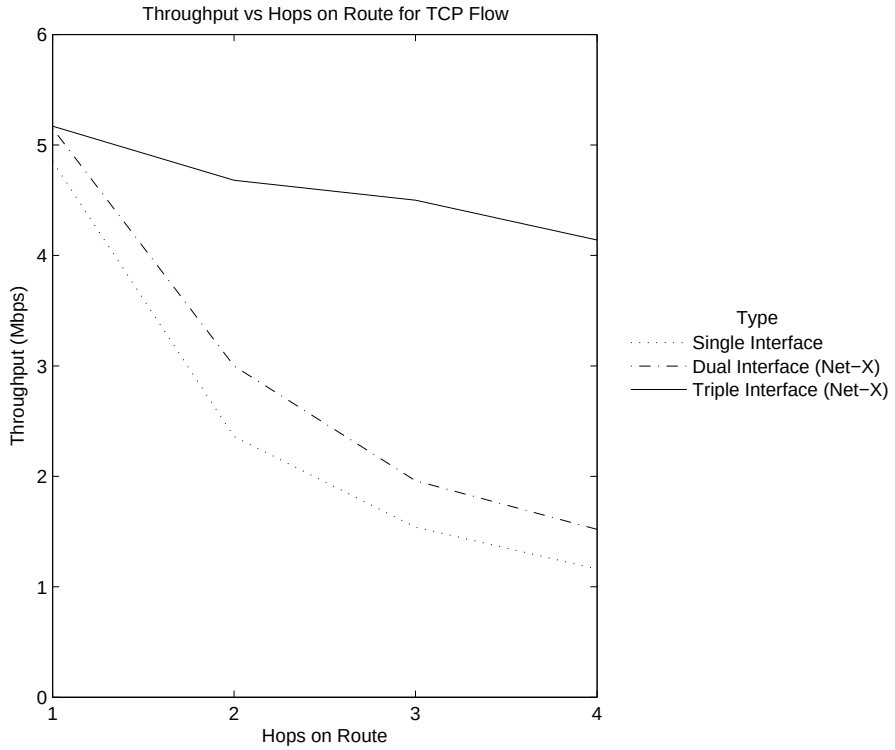


Fig. 18 TCP throughput results

E. UDP Results

As seen in Figure 19, there is already a huge performance gain in two interface Net-X over a single interface network. Adding a third interface should not make much of a difference since there is no need for bidirectional transmission in UDP. Our results show a slight improvement in throughput when using a third interface. This improvement is likely due to distributing the transmission of "hello" packets over two interfaces. Thus, the interface used to send data packets spends more time on the channel sending data packets instead of switching and sending "hello" packets. If there were UDP flows in both directions, the aggregate throughput of the flows would still be bounded by the rate of one interface. Due to the single fixed interface being used for receiving, it is currently impossible for aggregate throughput to be above 6 Mbps in a multihop chain topology.

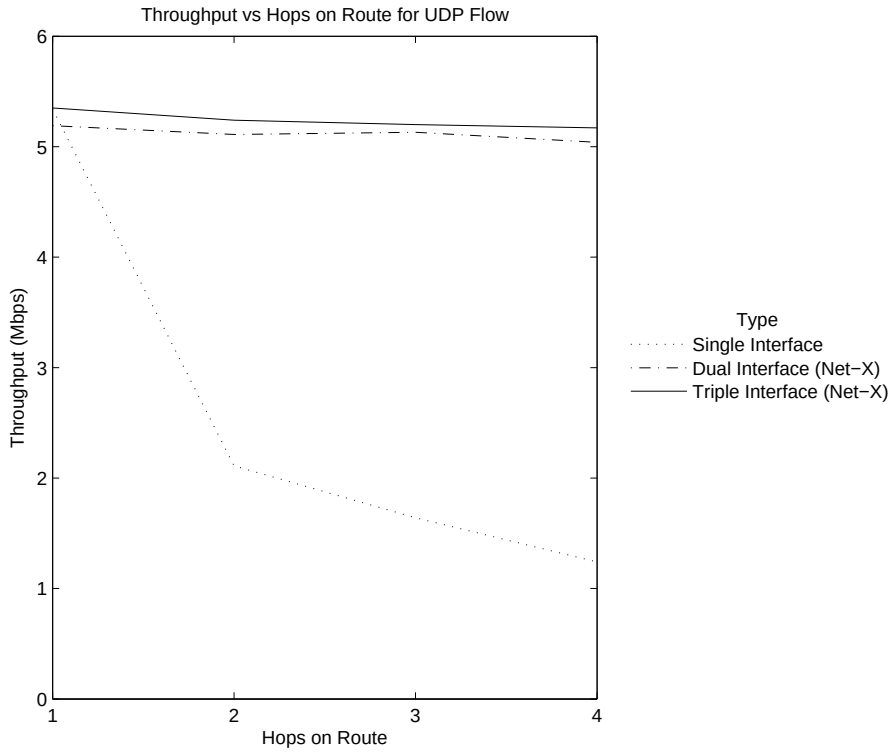


Fig. 19 UDP results

F. Summary

The poor performance of TCP over multiple hops is addressed by adding a third "switchable" interface. This enables the simultaneous sending of TCP DATA and TCP ACK packets. Implementation issues for utilizing the third interface are discussed and one method is implemented. This results in TCP performance are 1.56×, 2.30×, and 2.72× that of the two interface Net-X system. UDP throughput is slightly increased.

VI. CONCLUSIONS AND FUTURE WORK

We conducted a variety of tests on our testbed. Beginning with a single interface, we found that different wireless interfaces have varying throughput results in the face of adjacent channel interference. Using multicast packets, we determined that adjacent channel interference can be worse than same-channel interference due to lack of deferrals. The experiments performed in the Illinois Wireless Wind Tunnel show that experiments sensitive to SINR will yield different results in two different locations. When utilizing two or three interfaces in the same node, we witnessed interference between channels {36, 49} and {149, 161}. This information needs to be taken into account if the network is to be fully loaded. There are only three channels ({36, 64, 149} or {36, 64, 161}) that are close to interference-free when utilizing three interfaces. Since interference seems to be affected by interface hardware used and environment, it is better to use protocols that can dynamically estimate this.

Our experiments of VoIP on the Net-X testbed show that two-hop VoIP can perform well. Drop rates of 0.14% and average measured delay of 29.2 ms for two-hop traffic is acceptable for VoIP. To improve performance of VoIP, a delay reduction scheme was implemented and shown to be beneficial. However, three-hop and greater VoIP traffic will likely be poor in performance.

We witnessed the poor performance of multihop TCP in single-channel ad-hoc and two-interface Net-X networks. The performance of multihop TCP was improved with the addition of a third "switchable" interface. The throughput using a third interface was 1.56×, 2.30×, and 2.72× that of two interface TCP throughput for two-, three-, and four-hop communication. UDP performance was largely unchanged since multihop UDP on Net-X already performs well.

A. Future Work

1) Cross channel interference: The cross channel interference varies depending on wireless interface and antenna. Finding a wireless interface with a better design would result in less cross channel interference, possibly allowing the use of more channels.

2) Switching delay: The results from VoIP tests show that the large switching delay greatly affects the performance of VoIP, and also takes away from the transmission time. Finding a wireless interface with a smaller switching delay or modifying the driver to reduce this switching time would help.

3) Minimum channel time: The minimum channel dwelling time of 20 ms hurts the performance of VoIP traffic. To reduce this idle time, a timer with higher resolution must be used. Porting the Net-X system to a more recent version of Linux might make this possible.

4) Routing: During our experiments, we noticed that routes would sometimes be lost when under heavy traffic. The "hello" packets do not make it to the destination, even though data packets are still getting through. Sending multiple "hello" packets to improve the probability of reception can help. Another problem is the reactive nature of the WCETT routing protocol. Multihop routes take a long time to set up and sometimes time out. A proactive routing protocol might solve this problem.

5) Multiple interfaces: The third interface can be used as a "fixed" interface, or even a variable interface that can be "fixed" for periods of high receiving and "switchable" for high transmissions. This would require further changes in the hybrid multichannel routing protocol, but could result in better performance.

ACKNOWLEDGMENT

I would like to thank my adviser, Dr. Nitin Vaidya, for guiding me through the research and thesis writing process. I would also like to thank my first mentor in the group, Pradeep Kyasanur. Thanks to Vijay Raman and Cheolgi Kim for sharing their ideas and knowledge. Thanks to Chi-Hung Lu and Robert Chang for the discussions we had. Last but not least, thanks to my family for supporting me every step of the way.

Research reported here is supported in part by National Science Foundation grants 04-23431 and 06-27074. Any opinions, findings, and conclusions or recommendations expressed in this report are those of the author and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] C. Chereddi, "System architecture for multichannel multi-interface wireless networks," M.S. thesis, University of Illinois at Urbana-Champaign, April 2006.

[2] P. Kyasanur, "Multichannel wireless networks: Capacity and protocols," Ph.D. dissertation, University of Illinois at Urbana-Champaign, 2006.

[3] A. Adya, P. Bahl, J. Padhye, A. Wolman, and L. Zhou, "A multi-radio unification protocol for IEEE 802.11 wireless networks," *Broadband Networks, 2004. BroadNets 2004. Proceedings. First International Conference on*, pp. 344–354, 25-29 Oct. 2004.

[4] P. Kyasanur and N. Vaidya, "Routing and interface assignment in multi-channel multi-interface wireless networks," *Wireless Communications and Networking Conference, 2005 IEEE*, vol. 4, pp. 2051–2056 Vol. 4, 13-17 March 2005.

[5] P. Kyasanur and N. H. Vaidya, "Routing and link-layer protocols for multi-channel multi-interface ad hoc wireless networks," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 10, no. 1, pp. 31–43, 2006.

[6] R. Draves, J. Padhye, and B. Zill, "Routing in multi-radio, multi-hop wireless mesh networks," in *MobiCom '04: Proceedings of the 10th Annual International Conference on Mobile Computing and Networking*. New York, NY, USA: ACM, 2004.

[7] C. Cheng, P. Hsiao, H. Kung, and D. Vlah, "Adjacent channel interference in dual-radio 802.11 nodes and its impact on multi-hop networking," in *IEEE Global Telecommunications Conference (GLOBECOM 2006)*, November 2006.

[8] "Supplement to ieee standard for information technology telecommunications and information exchange between systems - local and metropolitan area networks - specific requirements. part 11: wireless lan medium access control (mac) and physical layer (phy) specifications: high-speed physical layer in the 5 ghz band," *IEEE Std 802.11a-1999*, 1999.

[9] N. H. Vaidya, J. Bernhard, V. V. Veeravalli, P. R. Kumar, and R. K. Iyer, "Illinois wireless wind tunnel: A testbed for experimental evaluation of wireless networks," in *E-WIND '05: Proceedings of the 2005 ACM SIGCOMM Workshop on Experimental Approaches to Wireless Network Design and Analysis*. New York, NY, USA: ACM, 2005, pp. 64–69.

[10] "ITU-T recommendation G.114," International Telecommunication Union, Tech. Rep., 1993.

[11] B. Goode, "Voice over internet protocol (voip)," *Proceedings of the IEEE*, vol. 90, no. 9, pp. 1495–1517, Sep 2002.

[12] A. Markopoulou, F. Tobagi, and M. Karam, "Assessment of voip quality over internet backbones," *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 1, pp. 150–159 vol.1, 2002.

[13] "Pulse code modulation (pcm) of voice frequencies, ITU-T recommendation G.711," International Telecommunication Union, Tech. Rep., 1993.

[14] "D-itg, distributed internet traffic generator," 2008. [Online]. Available: http://www.grid.unina.it/software/ITG/

[15] R. Bhardwaj, "Lessons from a multichannel wireless mesh network," M.S. thesis, University of Illinois at Urbana-Champaign, 2007.