# LESSONS FROM A MULTICHANNEL WIRELESS MESH NETWORK

BY

RISHI BHARDWAJ

B.Tech., Indian Institute of Technology, Guwahati, 2005

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2007

Urbana, Illinois

# Abstract

Wireless Mesh Networks offer a quick, easy and cost efficient solution to providing network connectivity. Mesh networks would be required to support high data rate in order to compete with other more conventional Ethernet based network solutions. The usage of multiple frequency channels and multiple interfaces at each wireless mesh node has been proposed earlier to increase the capacity of such networks. In this thesis we evaluate such a mesh network implementation where each node in the network is equipped with two wireless radio interfaces and uses multiple channels for transmission. The network employs a hybrid multichannel protocol which provides connectivity between nodes with multiple interfaces using multiple channels. The protocol is implemented on top of the Net-X system architecture. The Net-X framework provides support for frequent channel switching on an interface which is required by the hybrid multichannel protocol, with certain channel switching delay overheads added.

We look at the throughput improvements achieved for a single flow and for concurrent multiple flows in the multi-channel network as opposed to using a single channel in the network. We identify problems arising from the usage of multiple channels for concurrent transmissions in close range and discuss the effects of channel switching delay on the network. We propose certain changes in the Multi-Channel Routing (MCR) metric used by the hybrid multichannel protocol to address these problems. We then evaluate the throughput performance improvements observed when using the new routing metric as compared to using the old MCR metric. Finally, based on our observations and experience with the testbed, we discuss some ideas for developing new multi-channel protocols and mesh networks in the future.

*To Mother, Father and Sister.*

# Acknowledgments

I would like to thank my advisor Professor Nitin H. Vaidya whose able guidance and encouragement helped me immensely in my thesis work. I would also like to thank the former and current members of our research group Dr. Pradeep Kyasanur, Chandrakanth Chereddi, Nistha Tripathi and Vartika Bhandari. Their research work laid the foundation for my thesis.

I would also like to thank my friends Mohit Tiwari and Ashish Awasthi for reviewing my thesis and for all the good times. It is always fun talking to them and hearing them share their kind words of wisdom and wit.

# Table of Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

The IEEE 802.11 [25] standard and related technologies have been successfully used to provide connectivity between wireless devices. The 802.11 ad-hoc mode provides for setting up wireless network between a group of nodes. Wireless mesh networks are such ad-hoc mode based networks which are easily deployable and offer flexible and cost efficient solution to providing network connectivity.

Wireless mesh networks require high data rate support for good performance for which the mesh nodes need to use the available communication bandwidth as efficiently as possible. The 802.11 wireless technology standard divides the available bandwidth into multiple channels. Further, the commercially available wireless radio interfaces can transmit and receive on one channel at a time. With the costs of wireless interfaces lowering day by day, it is becoming economically feasible to equip nodes with multiple wireless radio interfaces. The use of multiple channels in the network with the help of multiple interfaces at a node offers an opportunity to significantly improve the capacity of the network as opposed to using a single channel in the network. In this thesis we look in detail at such a multi-channel multi-interface wireless mesh network implementation. The mesh network employs a hybrid multi-channel protocol proposed by Kyasanur and Vaidya [1, 2, 3], implemented on top of the Net-X system architecture [4, 5, 6]. By switching channels on an interface, as proposed in the hybrid multi-channel protocol, a node in the network can access the entire communication bandwidth.

We perform a complete system analysis of the various aspects of the multi-channel multi-interface network testbed. We describe the various issues that come up during the analysis and draw attention to points related to the usage of multiple channels in the network. The issues we report during our discussions are the ones we believe would be faced by other similar multi-channel wireless networks. Some of these issues originate due to frequent switching of channels on interfaces, while some of them relate to the usage of multiple channels and concurrent transmissions.

Chapter 2 summarizes related work. We describe our network set up and the hybrid multi-channel protocol including the multi-channel routing metric (MCR) [1, 2] employed on the testbed in Chapter 3. We then take a look at some significant implementation issues for the mesh network in Chapter 4. Chapter 5 looks at the effects of channel switching delay on the network. In Chapter 6 we take a look at the throughput gains achieved for a single flow and multiple concurrent flows in our multi-channel multi-interface network as opposed to using a single channel in the network. In Chapter 7 we propose certain changes to the MCR routing metric to take into account cross-channel interference and channel reuse on routes. Chapter 8 concludes with our discussion and lists some ideas and areas for future work.

# Chapter 2

# Related Work

Different attempts have been made to make use of multiple channels in wireless networks [8, 14, 17, 15]. Except for [15] the other schemes and implementations do not require frequent switching of channels on interfaces. In these other schemes channel assignment for different interfaces is fixed for long durations of time. The channel switching interface architecture developed in Multinet [15] is designed for nodes with a single-interface. The aim for Multinet is to have a wireless node with one physical wireless interface connect to an infrastructure network as well as an ad hoc network using two virtual interfaces.

The hybrid multi-channel protocol [1, 2, 3] proposed by Kyasanur and Vaidya aims to provide connectivity between nodes using multiple interfaces and multiple channels. The hybrid multi-channel protocol requires multiple wireless interfaces at each node and assumes that frequent channel switching is possible on at least one of the interfaces. The wireless mesh network testbed we analyze runs an implementation of this hybrid multi-channel protocol implemented on top of the Net-X system architecture [4, 5, 6]. Readings from an early prototype of our system were reported in [1] and [4]. The early prototype of the system had a very basic implementation of the routing protocol and lacked certain other key components. The readings reported in [1, 4] constituted a proof of concept of the Net-X system architecture and the throughput gains possible with the use of hybrid multi-channel protocol. In this thesis we build on the work done in [1, 4] and discuss some key system issues not properly addressed in the early prototype. We also evaluate different aspects of the system (routing, throughput for multiple concurrent flows etc.) and report different performance numbers from the network.

In the past there have been evaluations of other wireless protocols and testbed implementations such as MIT Roofnet [19, 21, 20], Uppsala University wireless network testbed [16] and others [22, 23, 24]. These have all been single-channel single-interface network testbeds.

We also evaluate the MCR [1, 2] routing metric scheme in our multi-channel testbed and propose modifications to the scheme to factor in the effects of cross channel interference and channel reuse on routes. Several routing metric schemes have been proposed in the past [11, 9, 8, 2]. The WCETT [8] routing metric scheme and MCR [2] routing metric scheme are the ones that are best applicable to our multi-channel network. None of these routing metric schemes account for cross-channel interference that may be experienced by nodes in a route. Further, they do not differentiate between channel reuse on hops close together in a route and channel reuse on hops spatially far apart. We observe that these two factors can make a difference in throughput achievable on a route. We propose changes to the MCR metric to account for cross-channel interference and channel reuse and set up experiments to observe UDP throughput performance of routes chosen by the new routing metric scheme as compared to the old MCR metric scheme.

# Chapter 3

# Background

This chapter describes the multi-channel multi-interface wireless mesh network testbed developed in our research lab. The mesh network runs the hybrid multi-channel protocol [1, 2] implemented on top of the Net-X system architecture [4, 5, 6].

Decreasing cost of wireless hardware devices today presents an opportunity to equip wireless nodes with multiple wireless radio interfaces. The presence of orthogonal channels in the spectrum defined by the wireless technology standards (such as IEEE 802.11) enable the simultaneous usage of multiple interfaces at a node.

The idea of using multiple wireless radio interfaces at a node raises new challenges and leaves several questions to be answered. The commercially available 802.11 wireless radio interfaces at one time can communicate on one channel. Assuming a distributed system without any central controller, the nodes using multiple interfaces and multiple channels in a wireless network need to synchronize amongst themselves and come up with a consensus for channel usage amongst them. Primarily, the wireless nodes need to come up with a channel assignment schedule which would provide information about how to reach a particular node. The schedule will specify at what time would a node be listening on what channel(s). A good channel assignment schedule is important for better performance of the system. A good channel assignment coupled with an intelligent routing protocol would provide for good quality, channel diverse routes in the network and increase the possibility of having more concurrent transmissions in the network. The hybrid multichannel protocol proposed by Kyasanur and Vaidya [1, 2] defines channel assignment and routing for multi-channel multi-interface node wireless network.

The network stack in today's systems is divided into separate layers with little inter-layer knowledge and interaction. For example, the network layer (OSI model layer 3) responsible for routing and addressing does not know the details of the underlying link layer (OSI model layer 2) which is responsible for link level connections. The network layer thus has no notion of channels existing at the 802.11 link layer. For implementation of multi-channel network protocols one would require network layer support for routing and other functionalities. The Net-X system architecture provides a framework designed to support multi-channel multi-interface protocols. A prototype of the Net-X architecture has been developed in our lab for the Linux Operating System (kernel 2.4.26). We have implemented the hybrid multi-channel protocol mentioned earlier to run on top of the Net-X prototype for our testbed implementation.

We now provide details about the hybrid multichannel protocol, the Net-X system architecture and its implementation in our wireless network testbed.

## 3.1 Hybrid Multichannel Protocol

Hybrid multichannel protocol, proposed by Kyasanur and Vaidya [1, 2], combines interface management together with an on-demand routing protocol. The protocol assumes that each node has at least two wireless interfaces, one of which can supports frequent channel switching. From now on we would refer to this hybrid multichannel protocol as HMCP. The basic functions of the protocol are as follows:

1. *Interface Management* : This involves assigning channels to the different interfaces at a node and exchanging this channel information between nodes such that nodes may be able to communicate with each other.

2. *Routing* : The protocol proposes an on-demand routing protocol which uses the multi-channel routing metric (MCR). The MCR metric is a modification of the channel-aware routing metric WCETT proposed by Draves *et. al* [8]. The MCR metric apart from preferring a channel diverse route also takes into account the channel switching costs involved at a node.

### 3.1.1 Channel Assignment

HMCP proposes a hybrid strategy for interface channel assignment. It assumes each node in the network has some $m$ interfaces ( $m \geq 2$) which is less than the total number of channels usable at a node. Some $1 \leq f < m$ of those interfaces are assigned channels for long intervals of time. These interfaces are the 'fixed' interfaces, the channels assigned to these interfaces are the fixed channels for the node. The channel assignment for fixed interfaces at a node is based on balancing the fixed channel usage in the two-hop neighborhood of the node. The remaining $m - f$ interfaces are dynamically assigned channels from the remaining channels in the spectrum. These are the switchable interfaces. Any communication with a node can only be done on one of the fixed channels for the node since these are the only channels one can be sure that the node will be listening on. Thus if node $S$ wants to send a message to node $D$, first it checks if it shares a fixed channel with node $D$. If so, node $S$ may initiate the communication with node $D$ by transmitting on the interface on the fixed channel shared between $S$ and $D$. If the two nodes do not share a fixed channel, node $S$ would be required to tune one of its switchable interfaces to one of the fixed channel of $D$ and then initiate the communication by transmitting on this interface. When neighbor $D$ has multiple fixed channels, we can potentially add intelligent algorithms for node $S$ to decide which fixed channel to use for communication with $D$.

Kyasanur and Vaidya's hybrid multichannel protocol (HMCP) simplifies the coordination required between nodes to know in advance the channel switching schedule for the nodes. One only needs to know the fixed channels for a node in order to communicate with the node. The hybrid channel assignment strategy also has the advantage of a dynamic channel assignment strategy such that any node can access any channel in the spectrum at any point of time. Also, the protocol does not loose out on the simplicity provided by a fixed channel assignment strategies and does not require complex coordination protocols required for exchanging channel switching schedules. One of the issues with this channel assignment strategy is that broadcast implementation adds overheads. A broadcast message is transmitted on all the channels usable at a node. This is done so that all the nodes in the vicinity of the transmitting node may be able to hear the broadcast transmission irrespective of their fixed channels. Thus, when a node has to send a broadcast message, it transmits a copy of the broadcast message on each of its fixed channels using the fixed interfaces. Also, it tunes its switchable interfaces one by one to all the remaining channels in the spectrum and transmits a copy of the broadcast message in each of the switchable channel. This implementation adds delay in transmission of the broadcast message. The routing protocol proposed by HMCP employs broadcast messages for route discovery. We discuss the effects of delay in broadcast transmission on routing and other aspects of the system in Chapter 5.

We have a 24 node testbed where each node is equipped with two wireless interfaces. One of the interfaces for each node is the fixed channel interface while the other interface is the switchable interface. Thus, there is one fixed channel for each node in our testbed.

### 3.1.2 Hello Message Mechanism

The nodes exchange their fixed channel information by broadcasting hello messages. Each node in its hello message sends its own fixed channel information and also the fixed channel information of all the nodes whose hello messages it can hear. These hello messages are how nodes come to know of their neighbors and the fixed channel usage in their two-hop neighborhood. This information also helps a node figure out if its current fixed channel assignment is good or whether it should change its fixed channel in order to balance channel usage in its two-hop neighborhood.

Each node broadcasts a hello message every $H$ (by default set to 5 for all nodes) seconds. These broadcast hello messages also help in link quality estimation. Each node maintains a 64 bit bitmap for each of its neighboring node, keeping tab of the successfully received hello messages in last $64 \times H$ seconds. This helps in estimating transmission success probability of backward link from the neighbor to the node. Similarly the estimate for the transmission success probability for forward link, from node to the neighbor, exists with the neighbor. The forward link transmission success probability is received from the neighbor who sends this information in each of its hello message. With the help of forward and backward transmission success probabilities of a link, a node can have an overall estimate of the quality of the link. The link quality estimation is required for routing purposes.

### 3.1.3 Routing Protocol

HMCP proposes the usage of MCR routing metric for choosing routes between nodes. The MCR metric is a modification of the WCETT [8] routing metric with the addition of channel switching cost in the metric cost calculation. The routing metric is implemented in the testbed on top of an AODV [10] style on-demand reactive routing protocol. When a source node $S$ has to start communication with a destination node $D$ (i.e., when the network layer at $S$ detects a packet for $D$ for which no route exists), the routing process is initiated. Node $S$ broadcasts a RREQ message with $D$ as the destination. All the nodes that receive the RREQ, forward (broadcast) the RREQ after appending their own routing details on the RREQ message. These details include their IP address, fixed channel information, channel switching cost information and ETT (Expected Transmission Time for a 1500 byte packet) on the last hop traversed by the RREQ. This process continues till the RREQ finally reaches the destination node $D$. Each instance of the RREQ which reaches destination node $D$ discovers a possible path to reach from $S$ to $D$. With the help of the routing information included in the RREQ messages, the node $D$ evaluates the metric cost associated with each of the paths discovered. The destination node $D$ then chooses the path with the least routing metric cost and unicasts the RREP for this path to the previous hop in the path. The RREP includes the entire path information and is unicasted hop by hop to the source. Along each hop when a node receives the RREP it updates its routing tables to include the forward route towards destination $D$ and a backward route towards source $S$ after reading the next hop and previous hop information from the RREP. After a RREP reaches the source node $S$ a route to the destination $D$ exists and all the packets buffered for the destination can now be sent along the route.

### 3.1.4 User Space Daemon

The implementation of the HMCP for our testbed is done in the Linux user space as a user space daemon. The daemon interacts with the Kernel Multi-Channel Routing (KMCR) module and Channel Abstraction Layer (CAL) modules in the system and glues together the different components to make the whole system work. The KMCR and CAL modules are described in detail in the next section.

## 3.2 Net-X System Architecture

The hybrid multi-channel protocol (HMCP) described in the last section requires support from the operating system to work. As discussed earlier, the network stack in today's systems is divided into separate layers with little inter-layer knowledge and interaction. The network layer (OSI model layer 3) responsible for routing and addressing does not know the details of the underlying 802.11 link layer (OSI model layer 2) and frequency channels. The network layer only chooses the next hop neighbor to forward a packet in order for it to reach its destination and the network interface to transmit the packet to the next-hop neighbor. The network layer thus instructs the link layer to transmit the packet on a particular interface to the specified next-hop neighbor. When a wireless node is equipped with multiple interfaces and is capable of transmitting packets over multiple channels, the information passed by network layer as described above may not be enough to reach the next-hop neighbor. The link layer at such a node must also know about which channel to transmit the packet on so as to reach the next-hop neighbor. In the specific case where the number of channels a node may use is equal to the number of interfaces it is equipped with, the choice of network interface to transmit a packet on may be enough for the link layer. In such a case each interface may be fixed on one particular channel over 'long periods' of time. The choice of interface to reach a particular neighbor thus also specifies the channel on which the neighbor would receive the packet. Otherwise if the intended next hop recipient of the packet has the capability to listen on all channels, the choice of channel at the sender to transmit the packet may not remain critical for communication.

In the general case where the number of channels a node may use are (far) greater than the number of interfaces at a node (more plausible case) the network stack architecture would not work in its current form. The link layer would require the intelligence of the channel to transmit each packet in order for it to be correctly received. Further the link layer may also be required to provide special services for multi-channel protocols, for example, a broadcast messages may need to be required to be transmitted on all the channels so that all the nodes in the vicinity of the transmitting node may be able to hear the message.

The Net-X project [6] aims to develop a system architecture, a framework, to seamlessly and efficiently utilize multi-channel, multi-interface capabilities in next generation wireless networks. The discussion above highlights important points and issues due to which current system architecture will fail to fully utilize the potential of multi-channel multi-interface capabilities in wireless nodes. Net-X provides for a generic framework over which different multi-channel, multi-interface protocols may be implemented.

We now provide details of our prototype implementation of the Net-X architecture for Linux operating system, kernel 2.4.26 . For more details about the implementation kindly refer to [4], [1] and [7].

### 3.2.1 Kernel Multi-Channel Routing Support

The Kernel Multi-Channel Routing (KMCR) module works at layer 3 (network layer) of the network stack. With the help of Linux net filter hooks [31] , the KMCR module is able to see the different packets passing through the Linux network stack. With this ability of inspecting packets in the network layer the KMCR module provides the following functionalities to any user level multi-channel protocol implementation:

1. *Initiate Route Discovery*: Whenever the KMCR module detects a packet with destination IP of a mesh node it checks for the existence of the route to the node. If no next hop information exists with the network layer, it sends a message to the user level protocol implementation to initiate a route discovery for the intended destination. The user level protocol implementation can later send a message to KMCR providing information about the success or failure of the route discovery for the destination. If the route discovery succeeded, the KMCR module can process the packets it had buffered for the destination, otherwise it drops all the packets buffered.

2. *Route Usage*: KMCR module keeps tab of routes currently in use and keeps refreshing timeout values for active routes. KMCR module updates route timers for all routes which it detects being active (packets traversing the forward/reverse path). This is useful for the user level protocol implementation which can ask the KMCR module if the routes it had set up earlier are still in use. If no packets have been forwarded/sent on the route for an extended period of time, the user process can then remove the route.

### 3.2.2 Channel Abstraction Layer

The Channel Abstraction Layer (CAL) is situated between layer 2 (link level layer) and layer 3 (network layer) of network stack. The CAL provides the functionality to specify the channel information for neighboring nodes in the mesh network. CAL layer is implemented as an extension of ‘Linux bonding module’ . All the wireless interfaces usable by the multi-channel protocol are ‘bonded’ (abstracted) as one virtual interface.

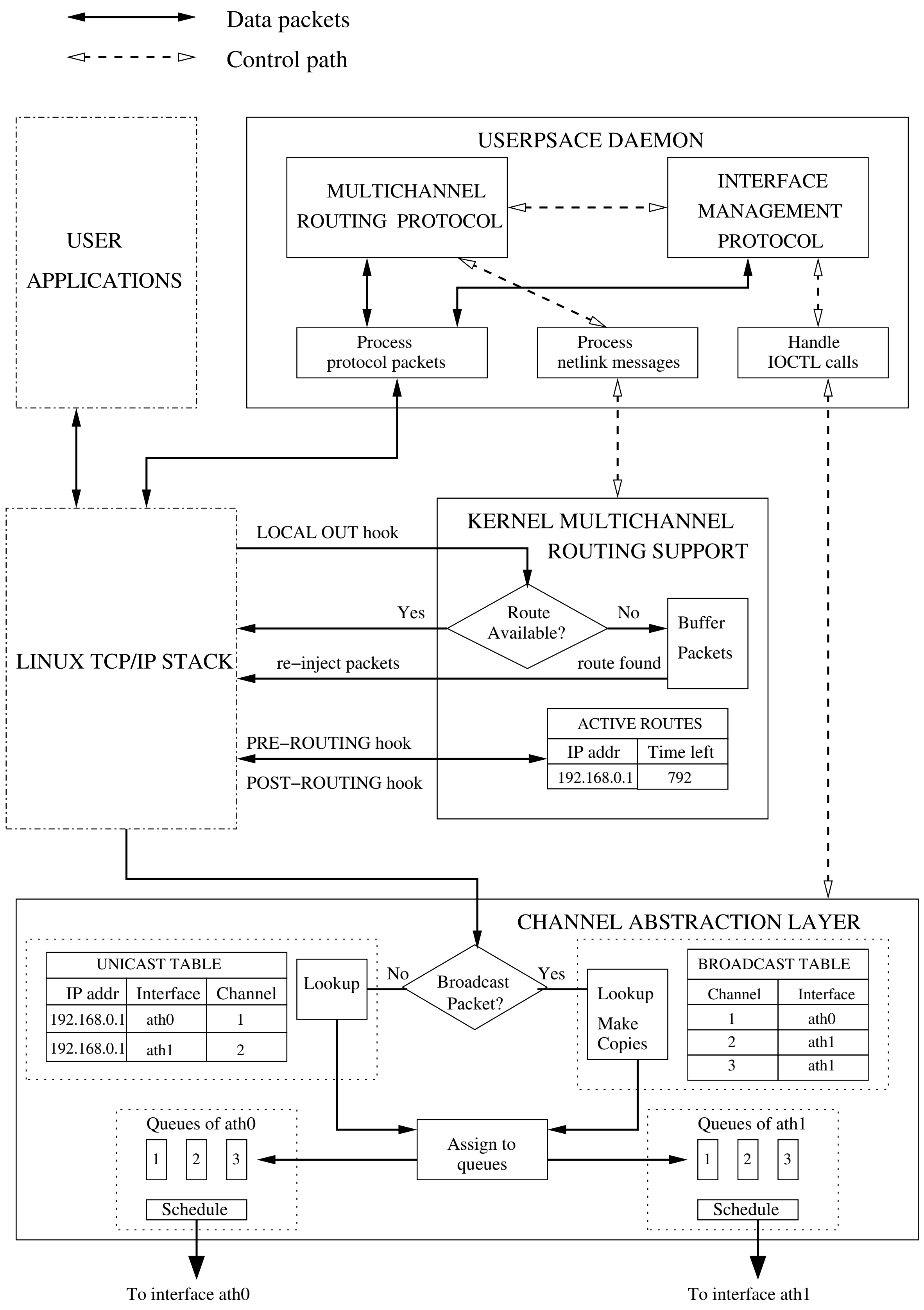
Network layer routes all packets to any node in the mesh network through this virtual interface. Thus, all mesh network packets to be transmitted by a node are handed over to the CAL. A multi-channel protocol implementation can specify to the CAL, the channel (or group of channels) to use when transmitting a packet to a neighboring node. Therefore, whenever a route is established by the multi-channel protocol it can instruct the CAL about what channel (maybe also the physical interface, power, data rate, etc.) to use for transmitting packets for a destination IP. This channel will be the channel on which the next hop node on the route to the destination will be listening. Thus, whenever the CAL layer receives a packet from the network layer to transmit on the virtual interface it checks the next hop channel information for the destination IP of the packet. If one of bonded interfaces with CAL is already tuned to the channel, the CAL can hand over the packet to the device driver responsible for transmitting on the interface. Otherwise the CAL needs to enqueue the packet till a switchable interface can be tuned to the channel required and then give the packet for transmission to the device driver of the switchable interface.

Also, the CAL can provide special treatment to layer 2 (link layer) broadcast messages. For our HMCP implementation the broadcast messages need to be transmitted on all the channels. The CAL thus provides a copy of the broadcast message to each of the fixed channel interfaces for transmission. Also it tunes the switchable interfaces one by one to the other remaining channels a node can access, to transmit the broadcast message on all the remaining channels.

Figure 3.1 shows a diagrammatic representation of the Net-X system architecture implementation in the testbed.

## 3.3 Wireless Mesh Network Testbed

We now describe the specific details about the setup of our wireless network testbed. The nodes in the testbed comprise of $Net4521$ boxes from Soekris [27]. Each node is equipped with two wireless radio interfaces (one connected through pcmcia interface and the other connected through mini-pci interface). The wireless cards are based on Atheros chipset [28], and support IEEE 802.11a/b/g protocols. A slightly modified madwifi device driver [30], [4] is used to access the wireless cards. The transmit power for the wireless cards is fixed at 18 dbm. They are configured to use 802.11a spectrum and protocol. The 802.11a defines 12 non overlapping channels, 4 each in the U-NII lower band (5.15 to 5.25 GHz), U-NII middle band (5.25 to 5.35 GHz), and U-NII upper band (5.725 to 5.825 GHz). Each of these channel is 20 MHz wide. The lower band channels are numbered 36, 40, 44, and 48. The middle band channels are numbered 52, 56, 60, 64; and the upper band channels are numbered 149, 153, 157, and 161. The cross-channel interference experiments



| ACTIVE ROUTES | |
|---|---|
| IP addr | Time left |
| 192.168.0.1 | 792 |

| UNICAST TABLE | | |
|---|---|---|
| IP addr | Interface | Channel |
| 192.168.0.1 | ath0 | 1 |
| 192.168.0.1 | ath1 | 2 |

| BROADCAST TABLE | |
|---|---|
| Channel | Interface |
| 1 | ath0 |
| 2 | ath1 |
| 3 | ath1 |

| | | |
|---|---|---|
| 1 | 2 | 3 |

Figure 3.1: A high level overview of the ‘HMCP Net-X testbed’ system architecture, borrowed from [5]

performed in [4] concluded that channels 36, 48, 64, 149 and 161 are usable for concurrent transmissions with tolerance required for certain amount of cross-channel interference. We use these five channels in our network. We will discuss cross-channel interference characteristics later in the thesis.

The two interfaces are made slaves of the bonding module and are visible as one virtual bonding interface to higher layers. Further, as reported in [4], switching a channel on these interfaces causes certain amount of delay which was measured to be 5 ms. Thus, every time a channel switch occurs on an interface there is a 5 ms overhead when no packets can be sent or received on the interface. We discuss the effects of the channel switching delay overheads in Chapter 5.

There are a total of 24 wireless nodes in our mesh network. The network is set up on the south section of fourth floor at the Coordinated Science Laboratory (CSL) building here in the University of Illinois at Urbana-Champaign. These nodes run a Linux 2.4.26 kernel with KMCR and CAL loaded as Linux modules. The hybrid multichannel protocol (HMCP) is implemented as a user space daemon. In the rest of the thesis we would use the term *'HMCP Net-X testbed'* to refer to our wireless network testbed running the HMCP protocol using five channels, and two wireless interfaces per node.

# Chapter 4

# System Implementation Issues

In this chapter we discuss certain implementation issues concerning our wireless mesh network and the hybrid multi-channel multi-interface protocol (HMCP). These are some interesting points which were not properly addressed during implementation of the first prototype of the system. These are important issues we believe would also occur in the implementation of other similar wireless networks.

## 4.1 Neighbor Discovery and Link Quality Estimation

Neighbor discovery and link quality estimation takes place with the help of the broadcast hello messages. Each node in its hello message sends its own fixed channel information and also the fixed channel information of all the nodes whose hello messages it can hear. These hello messages are how nodes come to know of their neighbors and the fixed channel usage in their two-hop neighborhood which is used in fixed channel assignment. A node $A$ also finds out if a link to a neighbor $N$ is "symmetric" , meaning the neighbor $N$ also considers the node $A$ one-hop neighbor, by looking at $N$'s hello message. If the one-hop neighbor list in $N$'s hello message contains information about $A$, it means node $N$ is able to successfully receive $A$'s hello messages. The link between $A$ and $N$ thus is 'symmetric' and is detected by both the nodes. Only symmetric links are usable with the 802.11 protocol which requires two way communication for a packet to be successfully transmitted.

Each node broadcasts a hello message every $H$ (by default set to 5 on the testbed) seconds. These broadcast hello messages also help in link quality estimation. Each node maintains a bitmap for each symmetric neighbor node keeping tab of the successfully received hello messages in last $64 \times H$ seconds. This helps in estimating transmission success probability of backward link from the neighbor to the node. The forward link transmission success probability is received from the neighbor who sends this information in each of its hello message. With the help of the forward and backward transmission success probabilities of a link, a node can have an overall estimate of the quality of the link. The link quality estimation is required for routing purposes. To provide accurate link quality estimation, the procedure above would be useful only if the size of the hello packets reflects the size of data packets that may be sent on the link. To provide an accurate estimate of link quality estimate, we set the size of hello packets to be 1470 bytes, the maximum data size currently supported for transmission on a link. Another point to note is that the forward and backward link quality estimates provided between a node $A$ and its neighbor $N$ are on two separate channels and separate interfaces. This is because the two nodes may have different fixed channels. In such a case, node $A$ transmits hello message on its switchable interface, which is received by $N$ on its fixed channel interface. The hello message received by $A$ is transmitted on $N$'s switchable interface on $A$'s fixed channel. Thus, the forward and backward transmission success rate provided by the link quality estimation mechanism is on different channels with different interfaces. During a unicast transmission from $A$ to $N$ the switchable interface from $A$ will transmit on fixed channel of $N$ and $N$'s fixed channel interface will send an 802.11 $ACK$ to be received on the switchable interface of $A$. The link quality estimation mechanism thus makes an assumption that transmission error rate between two nodes is not dependent on channel and interface (as long as transmission power on the interfaces is the same).

### 4.1.1 Prioritization of Hello Messages

The hello messages being sent by a node need to have priority over the data packets being sent by the node. This is necessary since the hello messages are management packets and the correct working of the network would depend on them. We prioritize the hello messages by appropriately setting the TOS field of the IP header of the hello packets. This provides them priority over normal data packets; they bypass any queues at the network layer, CAL layer or the device driver layer at the node to be transmitted before other packets at a node.

## 4.2 Routing Issues

We now list some of the issues we observed with the routing protocol implementation.

### 4.2.1 Route Discovery

The on-demand routing protocol implementation on the testbed is based on the AODV protocol. The routing protocol uses the MCR routing metric.

### Multiple RREPs

The AODV style on-demand routing protocol we have implemented is based on the underlying broadcast mechanism at a node. Broadcast is implemented by transmitting a copy of the broadcast packet on all channels. Each node in the network is equipped with two interfaces and uses 5 channels, thus broadcast involves channel switching on the switchable interface. Since channel switching delay is nontrivial (order of 5 ms), there is substantial variation in the time at which the broadcast message is transmitted on different channels. All this means that a RREQ message traversing different routes may reach the destination node at vastly different times. The destination node chooses from the routes discovered by the RREQ message. The destination can wait for all the different routes to be discovered by the RREQ message and then choose the best route and send the associated RREP. This may result in long route discovery latency (more on route discovery delay in the next chapter). Otherwise, the destination can reply with a RREP as and when it receives a route discovery message with a better route than the earlier discovered route. For our implementation we chose this second option, the destination replies with a RREP whenever it receives a RREQ (which is not stale) with a better route than the last chosen route. Thus, one can have multiple RREPs sent for a single route discovery attempt by the source. This requires the RREPs to have sequence numbers, such that any intermediate node or the source node does not accept a stale RREP (RREP with a sequence number earlier than the last RREP seen from the destination for the source). The RREP with the latest sequence number from the destination represents the last and the best route chosen by the destination based on the MCR metric.

### Suppressing RREQs

When a source node initiates route discovery for a destination node it broadcasts a RREQ message with destination IP information and its own unique sequence number. The source-destination pair along with the RREQ sequence number uniquely determines the source's route discovery attempt. All nodes that hear the RREQ message may further forward (broadcast) the message. Thus, for a single route discovery attempt by a source, several different RREQ messages may be spawned by the different intermediate nodes in the network in the attempt to reach the destination node. An intermediate node may receive multiple RREQ messages from different nodes discovering different paths. To keep the management message overheads low, the intermediate node may suppress (not act on) the RREQ message received later, if it estimates that the latest received RREQ would not lead to a better route compared to the RREQ forwarded earlier. For example, if the node detects presence of a routing loop in the path discovered so far by the RREQ message, the node should just drop the RREQ. Also, if the RREQ sequence number is less than the last seen sequence number for route discovery by the source to the destination, then the intermediate node should drop the RREQ. Otherwise, if the cost associated with the path discovered so far by the latest RREQ message is far more than the cost associated with the path of the RREQ forwarded earlier, the node may choose to drop the RREQ packet. Now, the MCR (multi-channel routing metric) is not isotonic, thus, it does not follow the greedy optimal approach. That is, an intermediate node $A$ can receive two RREQs (RREQ$_1$ and RREQ$_2$, with same source sequence number for route discovery to the same destination), having traversed two separate paths $PATH_1$ and $PATH_2$. The routing metric cost of $PATH_1$ ($COST_1$) can be less than cost of $PATH_2$ ($COST_2$) and yet it may that forwarding RREQ$_2$ by the node finally discovers a better path between the source and destination. For example, $PATH_1$, from source node $S$ to the intermediate node $A$, may consist of 2 hops, one each on channel 1 and channel 2. The other path, $PATH_2$, may consist of 2 hops, both on channel 1. Assuming the ETT (Expected Transmission Time) for a packet is same for all hops, the $COST_1$ for $PATH_1$ is less than $COST_2$ for $PATH_2$ because of channel reuse on consecutive hops in $PATH_2$. Let us say that the path from the intermediate node $A$ to destination $D$ consists of two hops both on channel 2. In this case the route $R_1$, from $S$ to $D$, consisting of the sub-route $PATH_1$, will have total of four hops with three hops on channel 2. The route $R_2$, consisting of $PATH_2$ will also have total of four hops with two hops on channel 1 and two hops on channel 2. Assuming the ETT is same for all hops, the MCR cost for $R_1$ will be higher than the cost of $R_2$ because of higher channel reuse. Even though the cost of the sub-route $PATH_1$ was less than the cost of the sub-route $PATH_2$, the cost of entire route consisting of $PATH_1$ was more than cost of route consisting of $PATH_2$. In the example above, if node $A$ forwards RREQ$_1$ and later receives RREQ$_2$, with cost of $PATH_2$ greater than cost of $PATH_1$, and decides to drop RREQ$_2$, the better route $R_2$ would not be discovered. Thus, we need to be careful while dropping RREQ packets. Currently we drop a RREQ$_i$, received after forwarding $i-1$ other RREQs with same source sequence number, for the same destination, if $COST_i > C_{suppress} \times COST_{best}$, where $C_{suppress} > 1$, $COST_i$ is the cost of the path discovered by RREQ$_i$ so far and $COST_{best}$ represents the lowest cost from all the $i-1$ RREQs forwarded earlier by the node. By default we set the value of $C_{suppress}$ to be 1.3. There is a trade off for setting the value of the constant $C_{suppress}$. If the constant is set too low, the overheads related to route discovery would be lower but one may not find the best route between a source and destination. Otherwise setting the constant too high may just increase the routing overheads without providing any benefits in regards to discovering better routes. While experimenting with different values for the constant we observed that the value 1.3 worked reasonably well for our testbed.

### 4.2.2 Separate Forward and Backward Route: Necessity or a Quirk?

When a forward route is established between a source and a destination, we currently automatically establish a reverse route between the destination and source consisting of the same nodes as the forward route. This is done keeping in mind that for most of the cases a reverse route would be required too (for example, with TCP connections and sometimes with UDP connections too.). There are some points to keep in mind with such an arrangement. First of all, according to MCR metric the best forward route between a source $S$ and destination $D$ may not be the best reverse route between $D$ and $S$. Furthermore there are other possibilities which may modify the reverse route between the destination and the source to comprise of totally different set of nodes. Let us say the source $S$ establishes a route to destination $D$ with $A$ as an intermediate node. Now let's say another node $B$ establishes a new route to $S$ which passes through $A$ but has a different next hop to reach $S$ compared to the reverse route between $D$ and $S$. Since there can only be one next hop node for route to $S$ at node $A$, the new route established by node $B$ takes precedence over the older established reverse route from $D$ to $S$ passing through $A$. In such a case the forward route between $S$ and $D$ would comprise of a different set of nodes as compared to the backward route between $D$ and $S$. Such a possibility may first seem unimportant but may have a significant impact on the throughput achieved over the route. We discuss this in more detail in Chapter 6.

### 4.2.3 Detection of Route Failure

We observed route failures mostly in two cases:

1. Link between two nodes in the route fails. In this case one of the nodes (or both) will stop receiving the other node's hello messages. Thus, the link will either become "asymmetric" or cease to exist, in either case both the nodes would come to know of the link failure. Whenever a node detects a link to be broken it checks if the unreachable neighbor served as a next hop node for some route. For each such route the node unicasts a RERR message to the source node of the route to inform the source of the route failure. On receiving a RERR message the source node can initiate a route discovery again.

2. When a route is set up, a timeout is associated with it. Each node in the route refreshes the route timeout whenever a packet on the route is forwarded. If no packet traverses the route for ROUTE_TIMEOUT duration the route is deemed stale and removed from the kernel routing table. Since this procedure is followed distributedly (all nodes timeout the route in an independent distributed manner) the routing information among the nodes may become inconsistent. Thus, route failure may occur when the routing information at an intermediate node in the route has timed out while the route

entry is still fresh at the source. A routing error mechanism is required to instruct the source of any such route failures. Detection of such route failures is implemented with the help of KMCR module. When the KMCR module receives a packet on one of the fixed channel interfaces to be forwarded to a remote destination, it checks if a route to the destination exists. If no next hop information for the destination node exists in the kernel routing table, the packet is dropped and the KMCR module sends a message to the user space daemon informing it about the source and destination of the packet dropped at the kernel network layer. Such an event means a route failure has occurred. The user space daemon is informed of the event, now the protocol implementation at the user space can handle it accordingly. In our current implementation when such a route failure is detected at a node, the node checks if a route to the source exists. If route to the source exists, it unicasts a RERR message to the source to inform about the route failure. If no route to the source exists at the node the node broadcasts the RERR message such that the previous hop on the route may receive the RERR and can then unicast it to the source. On receiving the RERR message the source node can initiate a route discovery again.

### 4.2.4 Route Refresh and Route Flapping

After a route has been setup between two nodes and a flow has been established one needs to regularly monitor the state of the chosen route. The quality of the established route may degrade over time or a new better route between the source and destination node may become available. To handle such situations we implement a route refresh mechanism for the testbed. A route discovery is initiated every RRFRESH_TIMEOUT (by default set to 30 seconds) interval for all active routes for which the source node broadcasts a 'RRFRESH' packet. The RRFRESH packets behave almost like RREQ packets except, to avoid route flapping or any frequent route changes, the costs of the new routes discovered by RRFRESH messages is multiplied by a constant $C_{rfresh}(> 1)$ before being compared to the cost of route from last route discovery cycle. Thus only if the new routes are significantly better than the current route being used would the new route be chosen and a RREP for the new route be generated by the destination. If the route discovered by the RRFRESH packet is same as the route currently in use, the latest cost of the route (being reported by the RRFRESH message) is compared with the route cost when last the route was chosen. If the latest cost is lower (better) than the last cost, we update the cost of the route currently being used as the new cost. Otherwise, if the latest cost for the route is higher than the cost of the route when it was last chosen, we update the cost value only if the latest cost is significantly higher than the previously stored cost for the route. That is only if $Latest\_Metric\_Cost > C'_{rfresh} \times Last\_Metric\_Cost$ , where $C'_{refresh} > 1$, do we update the routes metric cost. We do this to avoid frequent route changes or route flapping.

### 4.2.5 ICMP Redirects

We disable ICMP redirects [26] on the wireless interfaces on the nodes. ICMP redirects interfere with the routing set up on the network. The ICMP redirect messages are used by routers to notify hosts of availability of a better route to the destination nodes. If a mesh node (router) receives a packet from a host $H$ on interface $i$, to be forwarded to destination $D$; and if the node forwards the packet on the same interface $i$ on which it received the packet from the host, then the node may send an ICMP redirect message to host $H$ notifying the host that the destination $D$ may be reached directly from the host. If host $H$ accepts the ICMP redirect message, it would try to reach destination $D$ directly bypassing the router node. This may work well for Ethernet based networks, but for wireless networks (and that too multi-channel networks) the ICMP redirect messages could play havoc with the routing set up in the wireless networks. We, thus, disable sending and receiving of ICMP redirect messages on the wireless interfaces.

# Chapter 5

# Channel Switching Delay

The hybrid multi-channel protocol (HMCP) assumes that frequent channel switching on a wireless interface is feasible without significant overheads. To enable efficient channel switching on a 802.11 compliant wireless interface, a few modifications to the link layer is required. The details of the modification performed to the 802.11 link layer implementation in madwifi device driver can be found in [4]. With these modifications in place the channel switching delay for Atheros chipset based wireless card used in our network was reported to be 5 ms [4]. The delay of 5 ms is substantial enough to affect system performance. In this chapter we will discuss how switching delay affects the different aspects of the system.

## 5.1 Lowering Channel Switching Overheads

We would like to minimize the number of channel switches occurring at a wireless interface to keep the channel switching overheads low. When the number of interfaces at a node is less than the number of channels usable at a node, channel switching becomes necessary in order to use all channels. In our testbed we have two wireless interfaces at each node and five channels that may be used. One interface at each node is fixed on one channel while the other switchable interface can tune to the other four remaining channels as and when required. When the Channel Abstraction Layer (CAL) for a node receives a packet to transmit, it checks on which channel the packet needs to be transmitted. If the packet is to be transmitted on the fixed channel it is handed over to the device driver (madwifi [30]) handling the fixed channel interface for transmission. Otherwise if the packet needs to be transmitted on one of the other four remaining channels, the CAL needs to tune the switchable interface to the intended channel for the packet. CAL then has to make an important decision as to when does it tune the switchable interface to the intended channel for the packet. There is a trade off involved here, if the CAL keeps switching the interface to different channels as and when it receives packets to transmit for different channels, the switching overheads may become very high. This would be true when the node is involved in transmitting packets for multiple flows with different next hop channel specifications. Otherwise if the CAL, after buffering a packet for a channel, waits too long to tune the switchable interface to the channel, the high delay in transmitting packets on the particular channel could result in lower performance.

We adopt the policy to not switch channel on an interface for a constant amount of time after a switch has just been made on the interface. This constant represented by CHAN_MIN_TIME is by default set to 20 ms. We also try to bound the maximum delay a packet may experience at CAL in a channel queue. The details of the policy and its implementation follow below.

As proposed in [4] once we tune an interface on a channel we stay on the channel for at least CHAN_MIN_TIME (by default set to 20 ms). In our implementation whenever the bonding module (CAL) receives a frame to transmit on the switchable interface it checks for the channel the frame is to be transmitted on. Two cases may occur now

1. If the current channel of the switchable interface is the same as the destination channel for the frame, the frame is given to the device driver (madwifi [30]) for transmission. This is done provided there are no frames waiting to be transmitted on the switchable interface over other channels or the time spent on the current channel does not exceed the maximum time allowed on a channel (CHAN_MAX_TIME by default set to 60 ms). The CHAN_MAX_TIME bounds the time a frame would have to spend in a channel queue waiting to be transmitted on the switchable interface.

2. Otherwise if the current channel of the switchable interface is not same as the intended channel for the frame, the frame is buffered in a queue and a timer is started (provided a timer isn't already active). The timer runs for CHAN_MIN_TIME after which the channel switching on the switchable interface may occur and the frames waiting in the queue of other channels may be processed.

To lower channel switching overheads, a switchable interface should spend maximum possible time transmitting frames on different channels and minimize idle time and channel switching time. To keep channel switching overheads low we enforce the policy not to switch a channel on an interface before at least CHAN_MIN_TIME after the last switch on the interface. This is an optimistic approach hoping that we would get more packets to transmit for the current channel . This would be true in a system with high load. For a system lightly loaded on one channel and heavily loaded on the other channel this policy can backfire. We will come back to this point while discussing the TCP throughput for a flow in Chapter 6.

## 5.2 Round Trip Time

Round trip time (RTT) for a route between two nodes is affected by channel switching delay. The mesh network we have set up is a fairly dense node deployment, the propagation time between the nodes is negligible. The RTT for different routes is not dominated by propagation delay but the packet processing and transmission time at different hops on the route.

We measure the RTT for different multi-hop routes with the *ping* utility which employs the *ICMP echo* packets [26]. We compare the RTT observed for multi-hop routes in our HMCP Net-X testbed with RTT observed for multi-hop routes when a single channel is used in the testbed. For the latter case there would be no channel switching involved. Comparing the two cases would help us observe the effects of channel switching delay on RTT for multi-hop routes.

For a single-channel single-interface system we observe the RTT for different routes to be in the order of few milliseconds. For our HMCP Net-X testbed, channel switching delay dominates the RTT of a route.

We establish a single path between source and destination node for the forward and reverse route. What that means is that the reverse route from the destination to source consists of the same set of nodes as the forward route only in the reverse order. In such a scenario whenever the route is channel diverse (consecutive hops on the route are on different channel) channel switching delay at each intermediate hop on the route will dominate the RTT. For our hybrid multichannel protocol (HMCP) a channel diverse route implies that the consecutive nodes on the route listen on different fixed channels. For such a route, an intermediate node on the route forwards data towards destination on a different channel (fixed channel of next hop) and the reply towards source on a different channel (fixed channel of previous hop on the route). Currently we have only one switchable transmitting interface per node, thus, each intermediate node in a multi-hop channel diverse route will have to switch between transmitting data towards the destination and replies towards the source. As discussed earlier, every time an interface switches channel it stays on the channel for at least CHAN_MIN_TIME (currently set to 20 ms). The CHAN_MIN_TIME timer is started only when a frame for another channel is received. Thus, the RTT for a multi-hop path will increase by CHAN_MIN_TIME $\times 2$ for each hop added. This is because each intermediate node will wait CHAN_MIN_TIME before it tunes its switchable interface to the next hop neighbor's receiving channel for forwarding data towards destination. Then later the intermediate hop will again wait for CHAN_MIN_TIME to tune channel to previous hop's receiving channel for forwarding the reply sent by the destination to the source. Thus, each hop will add $2\times$ CHAN_MIN_TIME (= 40 ms) delay to RTT. Our experiment results for RTT support our observations above. Table 5.1 shows the RTT for single-channel case and multiple channel case. The minimum RTT observed for multi-hop routes in the HMCP Net-X testbed follows the trend we discussed above, average RTT $\approx$ CHAN_MIN_TIME $\times 2\times$ (NUMBER_OF_HOPS - 1), where NUMBER_OF_HOPS $> 1$.

The maximum RTT observed for multi-hop routes in our HMCP Net-X testbed is a little on the higher side. This is due to the hello message broadcasts taking place in the network. Each node broadcasts

| Hops on the Route | RTT in Milli seconds | | | | | |
|---|---|---|---|---|---|---|
| | Single-Channel Network | | | HMCP:Multi-Channel Network | | |
| | Min. RTT | Avg. RTT | Max. RTT | Min. RTT | Avg. RTT | Max. RTT |
| 1 hop | 2.1 | 2.1 | 2.8 | 2.1 | 15.9 | 68.5 |
| 2 hops | 3.2 | 3.3 | 3.5 | 45.8 | 66.7 | 289.2 |
| 3 hops | 4.6 | 4.6 | 4.8 | 59.1 | 105.2 | 280.9 |
| 4 hops | 5.9 | 6.0 | 6.8 | 105.4 | 147.6 | 482.1 |

Table 5.1: Round Trip Time for Multi-Channel Multi-Interface HMCP routes and Single-Channel routes as measured by 50 ping packets

a hello message to report its fixed-channel and neighbor information. These messages are sent every HELLO_TIME_INTERVAL (currently fixed to 5 seconds, for more info see Chapter 3). Thus every node every five seconds has a broadcast message to send. According to the hybrid multichannel protocol (HMCP) (Chapter 3) the broadcast message must be sent on all the channels. Currently each node uses five channels, for broadcasts the fixed interface transmits packets on fixed channel and the switchable interface transmits the packets for other four channels. Every HELLO_TIME_INTERVAL a node will require 3 channel switches on the switchable interface to send the broadcast hello packet. Thus if a data packet is received to be sent on some channel at the time the broadcast message is being sent, the data packet may have to wait for 3 channel switches before it can be transmitted. Thus a packet may observe a delay of $3 \times 20 = 60$ ms at a node if it is processed just after a broadcast packet is scheduled to be sent. This probably is the reason behind the maximum RTT time observed. The ping *ICMP echo* or the associated *ICMP reply* message may get delayed due to multiple hello message broadcasts that may occur at different nodes in the route. The ping packets are sent once every second and hello messages once every 5 seconds. Thus, on average every 5th ping packet got affected by hello message mechanism. We observe the RTT of other four ping packets remains close to the equation, RTT $\approx$ CHAN_MIN_TIME $\times 2 \times$ (NUMBER_OF_HOPS - 1), where NUMBER_OF_HOPS $> 1$.

The minimum RTT values observed are lower than the equation above. These minimum RTT values observed for the multi-hop routes are also due to the hello message mechanism. Since nodes in the route switch channels on their switchable interface to broadcast hello messages, it may so happen that the last channel tuned on the switchable interface at a node for hello broadcast could be the next hop channel for the route. In this case, when the node receives *ICMP echo* packet, after its hello broadcast is over, the node would not need to switch channels to forward the $ICMPecho$ packet to the next hop node in the route.

## 5.3 Delay in Broadcast

Broadcast mechanism is important for implementing various protocols. Many protocols (ARP, etc.) at different layers of networking stack assume an efficient broadcast mechanism implementation and rely on it for their correct working.

For the hybrid multi-channel protocol (HMCP) [1] employed in our testbed, a node must transmit a broadcast message over all channels. Each node has two interfaces, the fixed channel interface and the switchable interface. The fixed channel interface transmits packet on one of the channels (fixed channel) while the switchable interface handles transmission of packets on all other channels. We currently use five channels in our testbed. Upon receiving a broadcast packet, the bonding module (CAL) makes five copies of the packet, one each for the five channels. The interface on the fixed channel broadcasts the packet for fixed channel while the switchable interface transmits the other four packets one by one on each channel. Thus, the broadcast message gets transmitted at different times on different channels. This delaying of broadcast transmission on different channels at a node can add further complication to a protocol which relies on broadcast messages.

Let us try to quantify the delay a broadcast message may observe at a node. Interface on the fixed channel does not have to switch, thus, the broadcast packet for fixed channel does not deal with any switching delays. The broadcast packets for other channels would have to wait in the respective channel queues until the switchable interface switches to the respective channel. The switchable interface stays on a channel for at least CHAN_MIN_TIME (20 ms) and a maximum of CHAN_MAX_TIME (by default set to 60 ms) if there are other packets being transmitted. For lightly loaded nodes, when transmitting broadcast packets, there would not be packets to fill more than 20 ms worth of transmitting time for each channel. Thus, the maximum delay for a broadcast packet to be transmitted on a channel at a node would be CHAN_MIN_TIME $\times 3$ (= 60 ms). Four channels require three channel switches between them.

All broadcast based protocols (ARP, etc.) will experience this delay.

### 5.3.1 Routing Delay

We employ an AODV style reactive routing protocol [10, 2]. The routing protocol relies on the underlying broadcast mechanism at each node to discover routes. Since channel switching causes delay in broadcast transmission it also delays route discovery.

In our next experiment we observe the delay experienced in establishing a route between different pairs of nodes. As discussed in Section 4.2.1 we can have multiple routes discovered (chosen one at a time) between a pair of nodes for a single route discovery attempt by the source. As and when the destination receives a

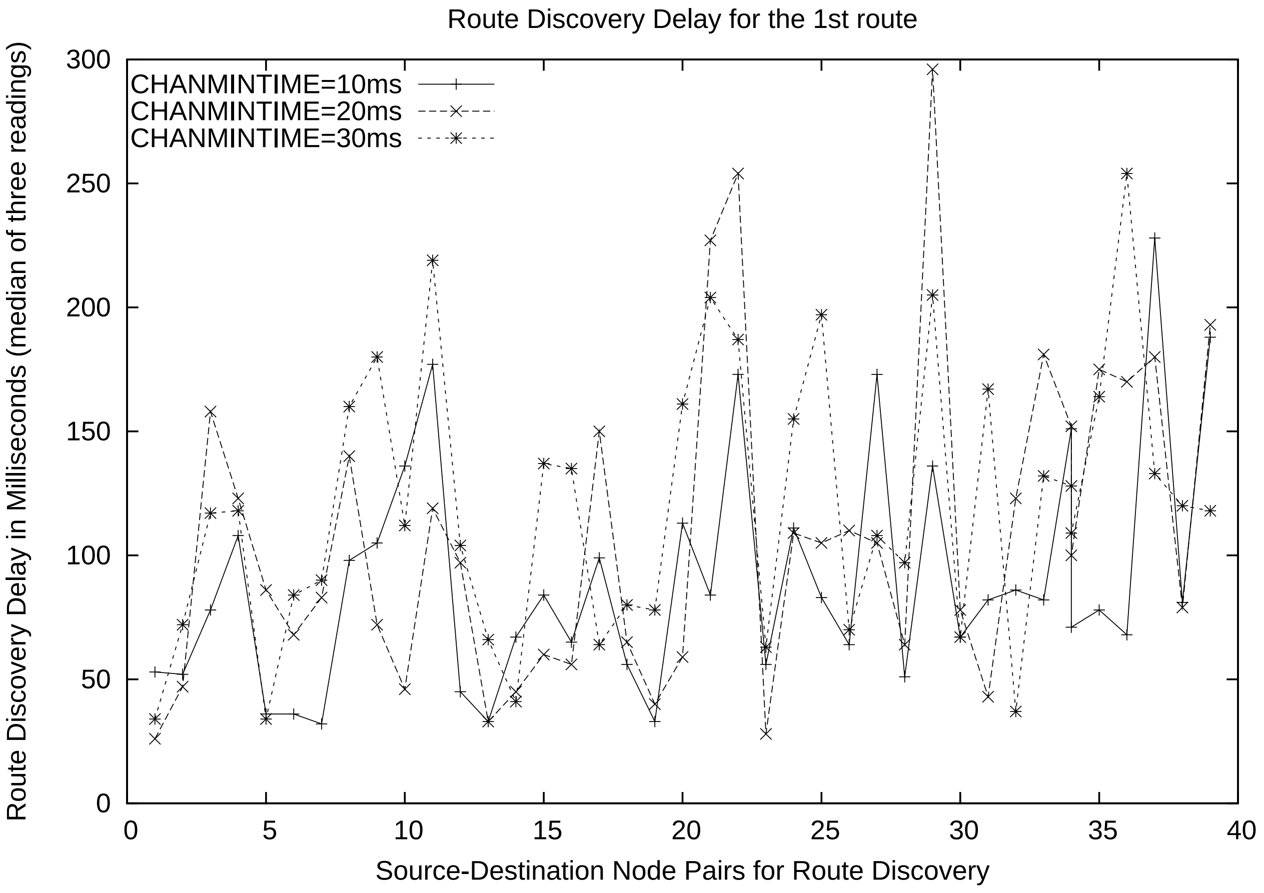


Figure 5.1: Delay in discovering a route between different pairs of nodes

RREQ with a better route discovered than the previous RREQ instance, a RREP is sent and the new route chosen. For our experiments, we report the delay in establishing the first route and the best route (last route chosen) between 39 different source-destination pairs of nodes. For each source-destination pair we repeat our experiment three times (three different route discovery attempts by the source). We report the median of the three route discovery delay readings between the source destination pair.

Figures 5.1 and 5.2 show the delay in setting up the first and best route between the 39 different source destination pairs. The routing delay for our HMCP Net-X network, we observe, can be as much as couple of hundred milliseconds. To reduce routing delay one may use a proactive routing protocol such as link state routing . But such protocols would involve other overheads in spreading routing updates in the network. The other way to lower the routing delay would be to lower CHAN_MIN_TIME. This would lower the delay observed in sending a broadcast message on different channels. Figures 5.1 and 5.2 show the delay observed in finding routes between different pairs of nodes for three different values of CHAN_MIN_TIME 10 ms, 20ms and 30ms.

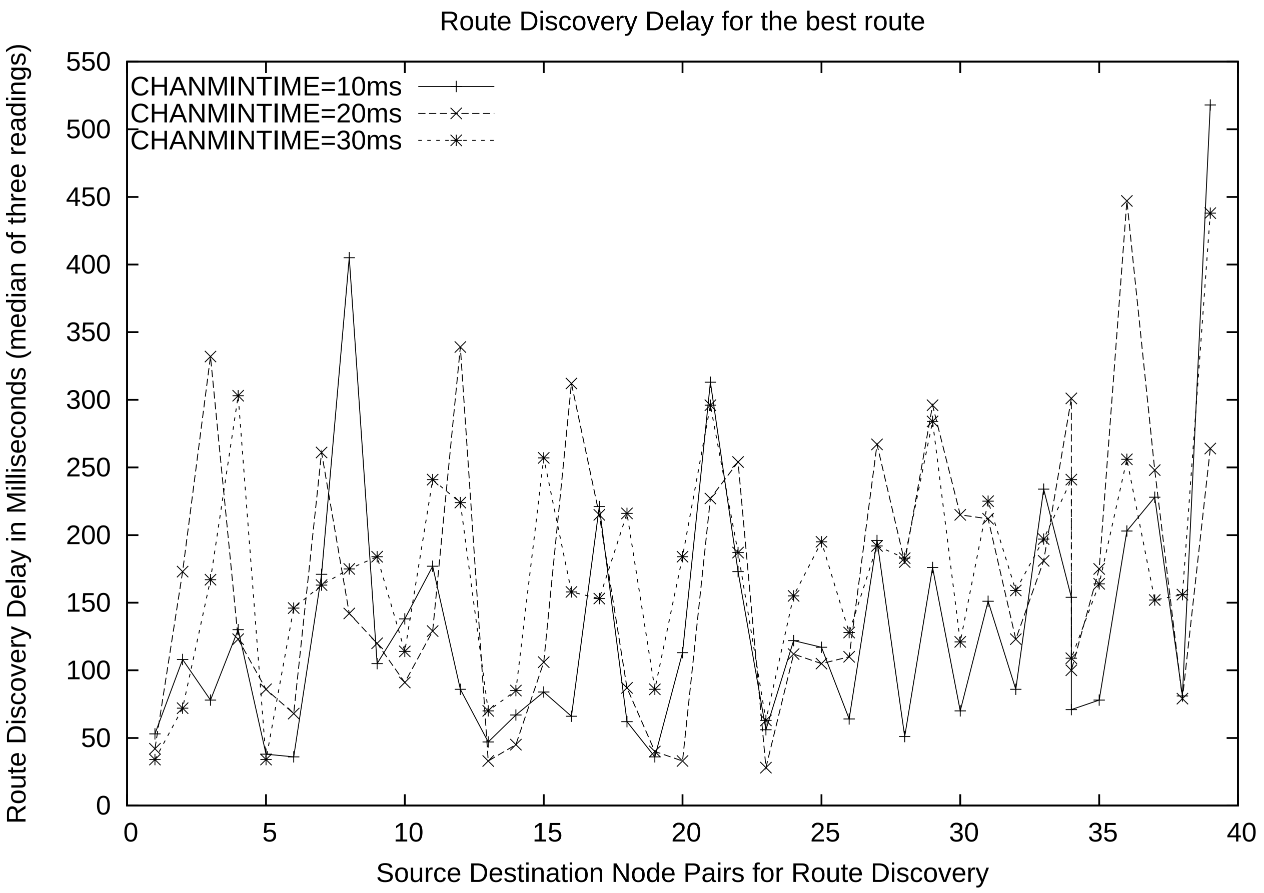


Figure 5.2: Delay in discovering the best route between different pairs of nodes

# Chapter 6

# Throughput and Performance Improvements

In this chapter we evaluate the performance improvements achievable in our HMCP Net-X wireless testbed as compared to using a single channel. We report observations for UDP and TCP traffic sent over multi-hop routes in the network. Routes in the network are established using the AODV style reactive routing protocol which employs the MCR routing metric, for more details please see Section 3.1.3.

We first report the performance improvements achievable for a single flow in the mesh network and then later set up experiments to observe throughput improvements for concurrent multiple flows. In the discussion that follows we refer to a flow in our multi-channel multi-interface wireless testbed as 'HMCP Net-X flow' and a flow in the network employing a single channel as 'single-channel flow'.

## 6.1 Single Flow Dynamics

As first step in evaluating performance improvements with HMCP and Net-X, we analyze the throughput obtained for a single data flow in our HMCP Net-X network. We try to find the best UDP and TCP throughput achievable for a multi-hop flow in the multi-channel mesh network and the factors which govern the results.

For our experiments we set up a chain topology of mesh nodes in the network and set up a single flow between the nodes. The source node for the flow is chosen at one end of the chain topology created. Each of the other nodes in the chain topology is chosen as the destination of the flow one by one in order to measure throughput as a function of hops in the route.

Measuring performance metrics in a wireless mesh network testbed can be non-trivial because of the different system and environment variables that can affect the results. Transmission power, data rate, interference and channel assignment are system level variables which influence performance. Noise and path loss are examples of environment variables which influence data rate achievable, error rate, etc.

Power and rate control are complementary to the hybrid multi-channel protocol (HMCP) being evaluated here. We statically fix the transmit power and data rate for our experiments. (For information on power and

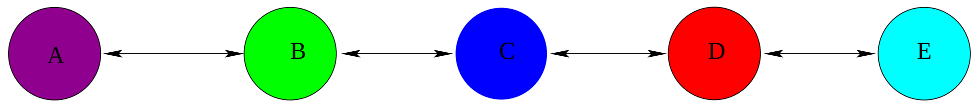


Figure 6.1: Five node chain topology. Different colors represent different fixed channel each node listens on.

rate control with Net-X see [7]). Only a single flow is established in the network so as to reduce interference from any other source.

For all the experiments done in this thesis, unless otherwise specified we fix the transmit power of wireless interfaces to 18 dbm and the data rate is set to be 6 Mbps (base rate for 802.11a). The nodes in the testbed are based on Intel 486 microprocessor with 100 MHz clock speed. We use the base rate of 6Mbps to avoid the microprocessor becoming the bottleneck in the experiments.

Further, we use 5 different channels in 802.11a spectrum namely channels 36, 48, 64, 149, 161. The fixed channel assignment of the nodes is accomplished through the two-hop channel balancing algorithm as discussed in [1].

### 6.1.1 Best Throughput Achievable

The chain topology set up consists of five nodes. A single flow set up amongst the nodes can thus have a maximum of four hops to traverse. Since we use five channels from the 802.11a spectrum, all four hops can be on a separate channel. This is important so that one may have non-interfering concurrent transmissions on all the hops.

Figure 6.1 represents the topology set up. The different colors of the nodes represent the different fixed channels each node listens on. The transmission on each of the four hops between the different nodes is on a different channel. Each node being equipped with two interfaces can thus listen and transmit concurrently.

The flow measurements are done with the traffic generator utility *iperf* [29]. The data packet size is set to 1470 bytes. We establish the flows for 50 seconds and here report the average of three experiments done one after the other. The routes are established with the AODV [10] style reactive routing protocol proposed in [1, 2]. The route establishment is done before the measurements are started so as not to include any routing delay while measuring the performance. The delay in routing is discussed in more detail in Section 5.3.1.

We wish to quantify the best achievable throughput for a flow. All hops in the routes set up are of very good quality with low error rate. Ensuring that all hops in a route are of very good quality is a job for the routing protocol. The routing protocol may not find a route with all good quality hops, or such a route may not exist between the source and the destination. Here we ensure that all the hops in the chain topology are of good quality and the route discovered by the routing protocol thus is also good. The results obtained here thus represent the best possible throughput achievable for multi-hop paths. There is always the possibility of a node in the chain topology detecting a ‘far away’ node as a one-hop neighbor in which case the chain topology might break. For example, node A in figure 6.1 may detect node C or node D as one-hop neighbor and the routing protocol may chose a route with this ‘far’ link as one of the hops. The hop from node A to C or A to D may not be good enough for best throughput in which case route with these long hops should be avoided. Routing protocol may or may not avoid such a route (more detailed discussion about the routing protocol follows in Chapter 7). The chain topology here was set up carefully such that all the hops between the consecutive nodes in the topology were of very good quality yet none of the longer hops between other nodes were good enough to be chosen by the routing protocol.

We measure the UDP and TCP throughput achieved for a single flow for the multi-hop routes established on the chain topology. We compare the results obtained with single-channel single-interface case. For the single-channel single-interface network, the routes are statically set up and we turn off all hybrid protocol hello messages. Thus, the single-channel case does not have any protocol overheads associated with it. We analyze the throughput results obtained for HMCP Net-X flow and list some interesting points and observations that came out of the experiments performed.

### 6.1.2 UDP Throughput

For single-channel single-interface network we observe the maximum UDP throughput achieved over a one-hop link to be 5.38 Mbps. There being no protocol overheads for the single-channel case, this throughput represent the maximum achievable UDP throughput between two nodes (with data rate fixed at 6Mbps). For multi-hop routes, we observe the throughput for single-channel case falls drastically due to self-interference from the flow.

For our HMCP Net-X testbed, we measure the maximum UDP throughput achieved for a one-hop link to be 5.21 Mbps. The difference between the one-hop UDP throughput for the single-channel case and HMCP case represents the protocol overheads with our implementation. As part of the hybrid multi-channel protocol (HMCP), each node sends a hello message every HELLO_TIME_INTERVAL (currently set to 5 seconds). The broadcast message needs to be transmitted on every channel being used in the system, for which the switchable transmitting interface needs to switch channels. Currently we use five channels in the 802.11a spectrum, the broadcast hello message (1500 bytes including IP header) eats up about 83 ms of time (more details about broadcast overheads may be found in Section 5.3). Thus, the overhead experienced should be about 83 ms every 5 seconds, or about 1.7%. The overheads calculated here are only for one node. Other

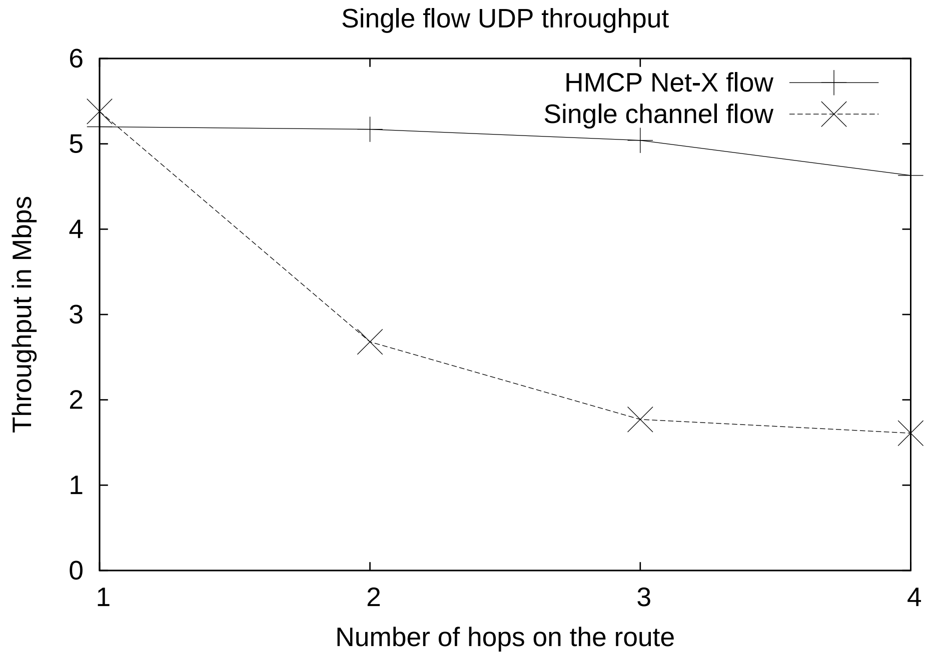


Figure 6.2: UDP Throughput achieved by a single-channel flow compared to a multi-channel multi-interface Net-X flow

| Hops on the Route | Throughput in Mbps | |
|---|---|---|
| | Single-Channel Flow | HMCP Net-X Flow |
| 1 | 5.38 | 5.21 |
| 2 | 2.68 | 5.17 |
| 3 | 1.77 | 5.04 |
| 4 | 1.61 | 4.63 |

Table 6.1: UDP Throughput Achieved for a single-channel flow and HMCP flow

nodes transmit their broadcast packet every five seconds and eat up transmission time, which also adds to the transmission overhead.

The overheads observed with the one-hop UDP throughput experiments here are $((5.38 - 5.21)/5.38) \times 100 = 3.15\%$,

For multi-hop paths we observe the real gains achieved by our HMCP Net-X network over the single-channel network. HMCP Net-X network allows the throughput achievable over a single hop to be sustained over multiple hops for a flow. With the use of orthogonal channels for different hops in a route and the use of multiple interfaces at each node, our network is able to maintain the single hop UDP throughput over multiple hops. The concurrent transmissions by nodes in the multi-hop route do not interfere with each other. The single-channel wireless network flow suffers from self-interference and thus can not sustain the throughput of a one-hop route over a multi-hop route. Figure 6.2 and table 6.1 show throughput measured for multi-hop routes using single channel in the network as compared to multi-hop routes in the HMCP Net-X testbed.

The overhead added by our system is measured to be around 3.5% for a one-hop flow. Also since our system implementation is able to sustain one-hop UDP throughput over multiple hops, we argue that the system does not add any other considerable overheads for multi-hop flows.

The results obtained above represent the upper-bound on the achievable throughput for a single flow(with data rate fixed at 6 Mbps) in our system. All the hops in the multi-hop route established above were of very good quality with very low error rate. Further all the transmissions on the hops were on orthogonal channels. There was no interference from any other nodes/flows except the nodes involved in the experiment. The throughput achieved for a flow in general case would be less than or equal to the results obtained above.

### 6.1.3 How to Attain Best Throughput for a UDP Flow?

The experiment above highlighted some interesting points. We observe that the throughput achievable for one-hop wireless flow can be sustained over multiple hops if the transmissions on the hops are on orthogonal channels. The concurrent transmissions over different channels with low error rate are thus sustainable. This remains true only when the links involved are of very good quality. We observe that a wireless mesh node transmitting on one interface while concurrently receiving on another interface must deal with cross-channel interference. This was also reported with the early experiments on the testbed in [4]

Since the transmitting antenna and receiving antenna are very close to each other (on the same node) even a small amount of cross-channel energy leakage on the transmitting antenna may lead to sufficient interference via the receiving antenna. We observe that if the signal strength on the receiving antenna is not of good quality to start with, the concurrent transmission on the transmitting antenna may cause high packet drop rate at the receiving antenna. Thus, if even one of the intermediate links on the multi-hop route is not of the best quality, the error rate on that link can severely affect the throughput for the flow.

The error rate experienced on an intermediate receiving link in a multi-hop route for a flow can be far worse than the estimation provided by link quality estimation mechanism. This would be true if the link quality estimation mechanism does not take into account cross-channel interference that may be experienced at a node. Thus, it is important to choose routes with good quality links to get the best throughput results.

Cross-channel interference may be improved with better hardware. Still, since the receiving and transmitting antennas are physically very close to each other (on the same node), some cross-channel interference may remain. Another way to tackle the problem would be to have the multichannel protocols be cross-channel interference-aware. For example, routing could be done so as to prefer routes where receiving channel at a node is far away in the frequency spectrum from the transmit channel at the node. Also it may be worthwhile to choose longer routes, with all the intermediate links to be of very good quality, than a shorter route with average link quality hops. We discuss these ideas in more detail in Chapter 7

### 6.1.4 TCP Throughput

TCP throughput for a connection is dependent on its round trip time, error rate and the connection bandwidth. As we observed for UDP traffic, if the individual hops in a route are of good quality, the HMCP Net-X network can maintain the data rate available on a single hop for the entire multi-hop route with low error rate. Thus one might expect the TCP throughput for a connection to be similar to the UDP throughput for the route. This unfortunately was not what we observed. We measured the one-hop TCP throughput to be similar to the UDP throughput, but for multi-hop routes the TCP throughput was far less than UDP throughput. The HMCP Net-X network though still attained better performance than the single-channel case but the improvements for TCP were not as substantial as for UDP. Figure 6.3 shows the TCP throughput achieved by a HMCP Net-X flow as compared to a single-channel flow. The figure also shows the HMCP Net-X UDP throughput for comparison.

The results we observe here are similar to those reported earlier by Kyasanur in his thesis [1] by experimentation on the early prototype of the system. We observe the one-hop TCP throughput for HMCP Net-X flow (5.1 Mbps) to be better than the throughput of one-hop single-channel flow (4.8 Mbps). This is different from what we observed for UDP, where one-hop UDP throughput of HMCP Net-X flow was worse by about 3% from the single-channel case due to protocol overheads. With TCP, HMCP Net-X flow achieves better results due to the duplex nature of the connection between the two nodes. The source and destination nodes have two interfaces each, with different fixed channels, thus, they can talk to each other concurrently. For one-hop route, the TCP data and TCP ACKs are transmitted on different channels and do not compete for channel transmission time as they do for single-channel case. This helps the one-hop HMCP Net-X flow to outperform one-hop single-channel TCP flow.

From figure 6.3 we observe that the TCP throughput achieved by HMCP Net-X flow on a multi-hop route is a lot worse than its UDP throughput for the same route. The reason for lower TCP throughput is the two way nature of data flow for the TCP connection. While establishing a route from source to destination, our routing protocol implementation implicitly sets up the reverse route to consist of the same nodes as the forward route. Thus the TCP data packets and TCP ACKs are being sent along the same set of nodes. Now there are only two interfaces at each node and only one switchable transmitting interface, the TCP ACKs thus eat up some of the transmission time from the data packets on the switchable transmitting

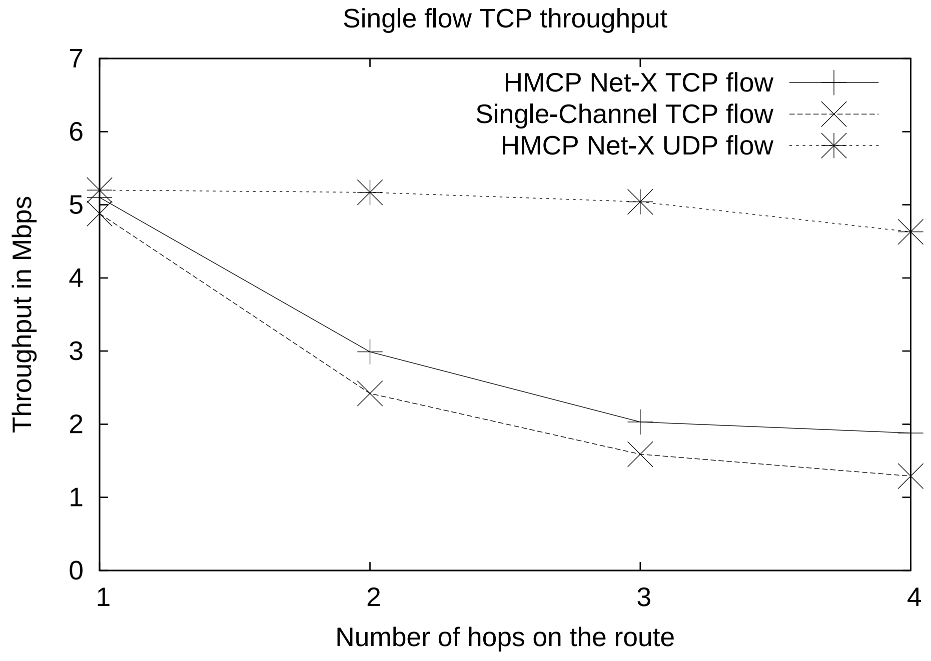


Figure 6.3: TCP Throughput achieved by a multi-channel multi-interface Net-X flow as compared to a single-channel flow

interface. But more importantly, for a channel diverse route, there needs to be frequent switching of channels on the switchable interface at each node for forwarding TCP data and TCP ACKs. For a channel diverse route, consecutive nodes in the route listen on different fixed channels. For such a route the transmitting interface for each intermediate node in the route has to switch channels between forwarding TCP data towards destination and forwarding TCP ACK towards the source. Channel switching delay is nontrivial and adds overhead. For The TCP rate control algorithm RTT of the route, specifically the delay in receiving of TCP ACKs determines the data rate.

The channel switching delay on the interface was measured to be 5 ms. Since the channel switching delay is substantial, one must avoid frequent channel switching. As discussed in Section 5.1 once we switch a channel we stay on the channel for at least CHAN_MIN_TIME (by default set to 20 ms). The policy decision not to switch a channel for at least CHAN_MIN_TIME after the last switch is an optimistic choice. We hope that once we switch to a channel we would get enough packets to transmit for the channel to fill much of the CHAN_MIN_TIME window. This would be true in a system with high load. For a lightly loaded system our policy can backfire. For TCP connections when an intermediate node in a route switches channel to forward an ACK towards the source it stays on that channel for at least 20 ms irrespective of the fact that there exists data to be sent on some other channel. During this time the TCP data would be buffered to be sent out later. TCP ACKs being small in size would take far less time to transmit than the data packets (each set

| Hops on the Route | Throughput in Mbps | | |
|---|---|---|---|
| | Single-Channel Flow | HMCP Multi-Channel Flow | HMCP:Multi-Channel Flow without channel switching overheads |
| 1 | 4.88 | 5.1 | 5.1 |
| 2 | 2.42 | 2.99 | 4.85 |
| 3 | 1.59 | 2.03 | 4.47 |
| 4 | 1.29 | 1.88 | 4.05 |

Table 6.2: TCP Throughput Achieved for a single-channel flow and HMCP flow

around 1500 bytes for our experiments). Forwarding of TCP ACKs towards the source by an intermediate node in the route takes little of the 20 ms window. The remaining time in the 20 ms window for the channel on the switchable interface is spent idle. The TCP data to be forwarded towards the destination is till then buffered in the other channel queue and waits for the switchable interface to be tuned to the fixed channel of next hop node.

The channel switching overheads and idle time while transmitting TCP ACKs degrades the TCP throughput as compared to UDP throughput where there are no ACKs and no channel switching for ACKs occurs.

### 6.1.5 Improving TCP Throughput

To improve the TCP performance one must improve on the channel switching overheads. One could choose less channel diverse routes such that channel switching may not be required. But this would increase self-interference of the flow and degrade performance, as seen for single-channel case. A better solution could be to choose the reverse route from the destination to the source of the TCP connection independent of the forward route. The best case would be to have none of the intermediate nodes in the forward route appear in the reverse route. Otherwise, if the intermediate nodes appear in the reverse route, the reverse route may be such that the intermediate nodes don't have to switch channels for transmitting data in forward and reverse route. We experiment with setting up independent forward and reverse routes such that none of the intermediate nodes of the two routes are common. Our twenty four node test bed is fairly dense and the routing protocol can find two such good quality routes between most of the node pairs. We observe a substantial improvements in the TCP throughput when the reverse route is chosen to be different from the forward route. The TCP throughput observed now is closer to the UDP throughput we saw earlier. Figure 6.4 and table 6.2 show the throughput observed for TCP flows in the HMCP Net-X network with separate forward and reverse routes as compared to the throughput achieved with channel switching overheads.

Choosing different forward and reverse route between two nodes has its own disadvantages. In sparse networks two good quality independent routes may not exist between two nodes. Further, for a reactive

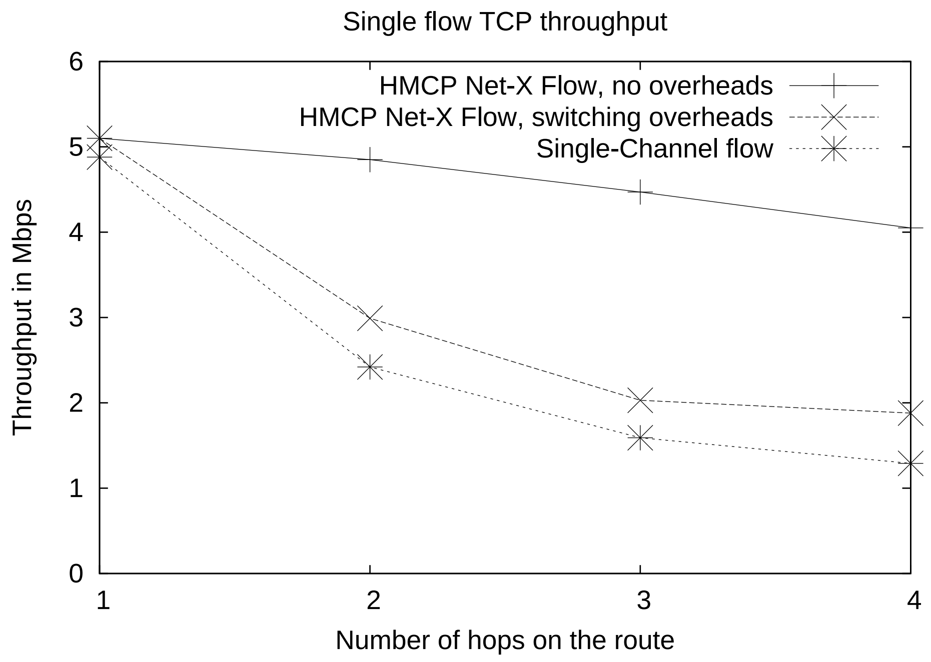


Figure 6.4: Throughput achieved by a TCP flow without channel switching overheads

routing protocol it will take double the time to establish two paths between a pair of nodes compared to a single path.

A better, cleaner solution to mask the channel switching delay at a node would be to add more hardware, an extra wireless interface, at each node. The third interface at the node can mask the channel switching delay. With the third interface added to our system we would have two switchable transmitting interfaces per node. Whenever channel switching is required and one transmitting interface is busy transmitting data on one channel, the other switchable interface can switch and transmit packet on the other channel. One would need more experimentation to see if two interfaces could transmit concurrently while the third interface receives data. If two concurrent transmissions originating from a node add too much cross-channel interference, one may decide to use only one switchable transmitting interface at a time. In this case we can use the two switchable interfaces as one virtual interface with negligible channel switching overheads. This virtual interface would be able to transmit on any two channels without any switching delay. We leave the implementation for adding the third interface on the nodes as future work.

Finally, the best solution for the channel switching delay problem would be to have specially designed hardware that has very little channel switching delay.

Currently we have not implemented any of the solutions for improving TCP throughput in the testbed. The TCP throughput is affected by channel switching delay and the gains achieved by enabling concurrent

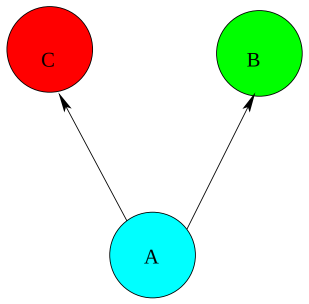


Figure 6.5: Set up of two one-hop UDP flows with one source node. Node A is the source of two UDP flows to B and C. All three nodes have different fixed channels.

transmissions with the use of multiple channels is offset by the channel switching overheads. Due to this reason, from now on we do not use TCP flows for our experiments on the testbed. We leave TCP flow experiments for future work when solutions to mask channel switching delay are implemented.

### 6.1.6 Channel Switching Delay and UDP Throughput

As we observed, channel switching delays substantially affected the TCP performance. In the next experiment we observe the effects of channel switching on UDP throughput. We set up two one-hop UDP flows originating from the same node as the source. The destination of the two flows are two different nodes listening on separate fixed channels. Figure 6.5 shows the source node A listening on channel 48 and the two destination nodes listening on channels 64 and 161 respectively. We observe that each of the flows achieves a throughput of about 2.25 Mbps, cumulatively the total throughput achieved by source node A is 4.5 Mbps. The one-hop UDP throughput for Net-X flow was measured to be 5.2 Mbps earlier. Thus we observe a 13% decrease in the achievable throughput. This figure of 13% can be explained as follows:

The channel switching on an interface causes a delay of 5 ms. To keep the channel switching overheads low one should spend as much time possible transmitting data in between two channel switches. The flip side of spending too much time transmitting data between two channel switches would be the delay experienced by packets waiting in transmit queues of other channels. To bound the delay a packet may experience in a channel queue at a node we enforce a bound on the maximum time the switchable interface can spend transmitting data on a channel when there are packets waiting in transmission queue of other channels. When CAL receive a packet to transmit on the switchable interface, it checks if the current channel of the switchable interface is same as the destination channel for the packet. If so, the packet is given to the device driver to be transmitted provided no packets are waiting for the interface in other channel queues or if the time spent on the current channel is less than CHAN_MAX_TIME. Otherwise, the packet is put in its channel queue and a timer bound is established on the transmitting interface. The switchable interface cannot stay on a channel for more than CHAN_MAX_TIME (currently set to 60 ms) if there are packets buffered to be sent on other channels.

In the two flow UDP experiment (figure 6.5), the fixed channels of destination nodes B and C are different and there exists only one switchable interface at the source node A. The switchable interface at A needs to switch channels regularly between transmitting data for the two flows. This is similar to the case of TCP flows (Section 6.1.4) where the intermediate nodes in multi-hop route required channel switching to forward TCP data and TCP ACKs. But unlike the TCP flow case, where the TCP ACKs were of small size and we spent idle time when switching channel for TCP ACKs, here we have enough data from the two UDP flows to fill the CHAN_MAX_TIME window (60 ms). The switching overheads thus remain lower than the TCP case where the throughput dropped significantly. Calculating the switching overheads, we switch after every 60 ms which wastes 5 ms of interface time. Thus, the overheads we should experience should be about 8% $((5/65) \times 100)$. In our experiments we observed a 13% drop with cumulative throughput from the two flows coming out to be 4.5 Mbps.

We may reduce the overheads by increasing CHAN_MAX_TIME, we observe the cumulative UDP throughput for the two flows increases to 4.72 Mbps when CHAN_MAX_TIME is increased to 100 ms. The observed overheads are about 9% $((4.72/5.2) \times 100)$ whereas theoretically the overheads should be around 5% $((5/105) \times 100)$. Increasing CHAN_MAX_TIME also increases the bound on the time a packet may spend in a channel queue waiting for the switchable interface to be available.

## 6.2 Throughput for Multiple Flows

We now look at throughput improvements in the HMCP Net-X testbed when multiple concurrent flows exist. We perform two types of multiple flow experiments. As part of the first experiment we observe how throughput differs when we have multiple one-hop flows concurrently transmitting data in our HMCP Net-X testbed as opposed to when a single channel is used in the network. For the second experiment we observe throughput variations when multiple multi-hop flows concurrently transmit data.

### 6.2.1 One-Hop Flows

As part of the first experiment we set up 10 one-hop routes in the network. Each one-hop route is of good quality and when used in isolation can support UDP data rate of more than 5 Mbps. Figure 6.6, from our Net-X visualization tool, shows the 10 one-hop routes set up in our HMCP Net-X testbed. We set up a UDP flow on each of the 10 routes to transmit data concurrently for 10 seconds at 4 Mbps. We conduct five

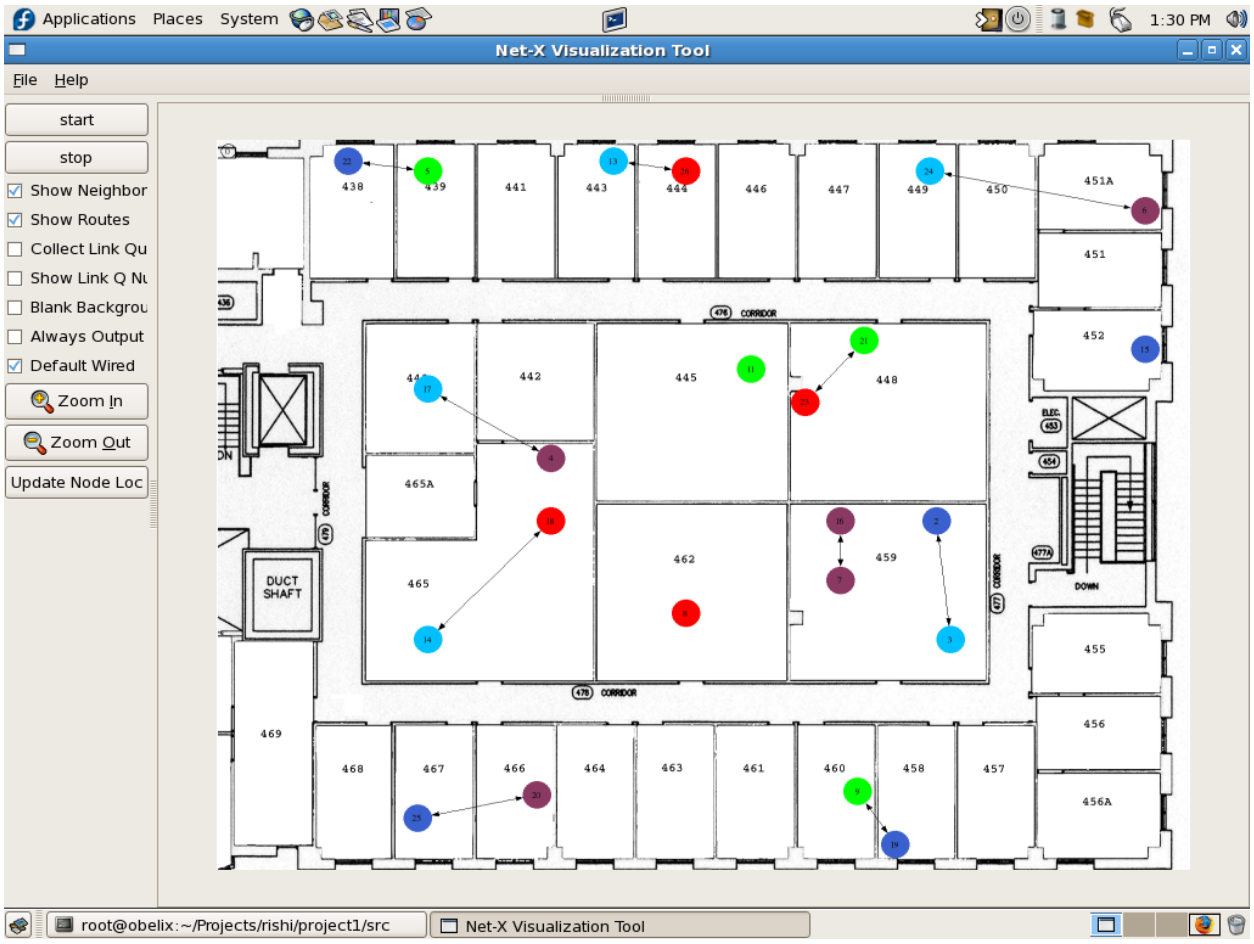


Figure 6.6: Route set up for 10 one-hop HMCP Net-X flows. Color of a node represents its fixed channel

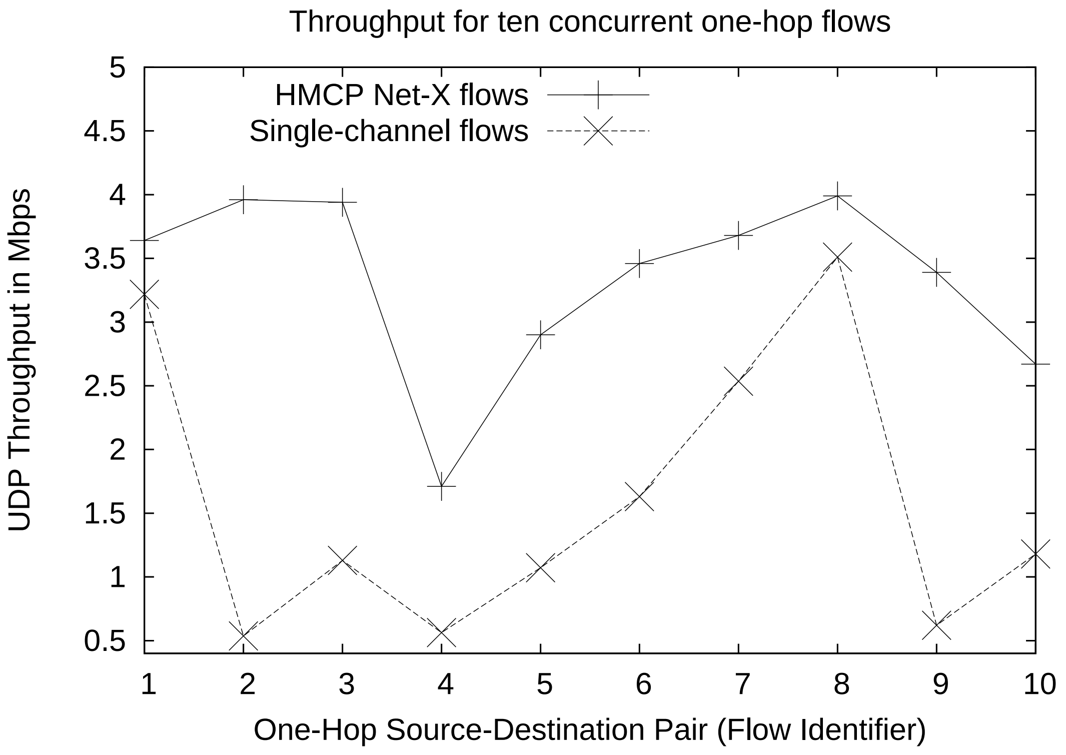


Figure 6.7: UDP throughput for 10 concurrent one-hop flows

such experiments and report the average UDP throughput observed for each of the flows. The whole setup is repeated using a single channel in the network. Table 6.3 and figure 6.7 compare the average throughput obtained for each of the HMCP flows and single-channel flows.

We observe that there is a significant difference in throughput achieved for HMCP one-hop flows and single-channel one-hop flows. The cumulative throughput achieved for the ten flows in HMCP Net-X network is more than twice the cumulative throughput achieved for the ten single-channel flows.

### 6.2.2 Multi-Hop Flows

For our next experiment we choose 20 sets of different node pairs in our testbed. Each set has different number of node pairs. Sets numbered 1 - 5 have two node pairs, sets numbered 6 - 11 have three node pairs, sets numbered 12 - 17 have four node pairs and sets numbered 18 - 20 have five node pairs. For each of these sets we will set up concurrent flows between each of its node pairs and measure the UDP throughput achieved. Each UDP flow will send data at 2.2 Mbps. Our experiment would measure the throughput achieved for each flow in the set. The routes between the node pairs can be multi-hop, we let the MCR routing metric choose the routes between the node pairs. After a flow is set up between a pair of nodes in

| Flow Identifier | Throughput in Mbps | |
|---|---|---|
| | Single-Channel Network | HMCP Net-X Testbed |
| 1 | 3.218 | 3.64 |
| 2 | 0.536 | 3.96 |
| 3 | 1.13 | 3.94 |
| 4 | 0.563 | 1.71 |
| 5 | 1.072 | 2.9 |
| 6 | 1.63 | 3.46 |
| 7 | 2.536 | 3.68 |
| 8 | 3.51 | 3.99 |
| 9 | 0.62 | 3.39 |
| 10 | 1.18 | 2.67 |
| Cumulative | 15.995 | 33.34 |

Table 6.3: UDP Throughput for 10 concurrent one-hop flows

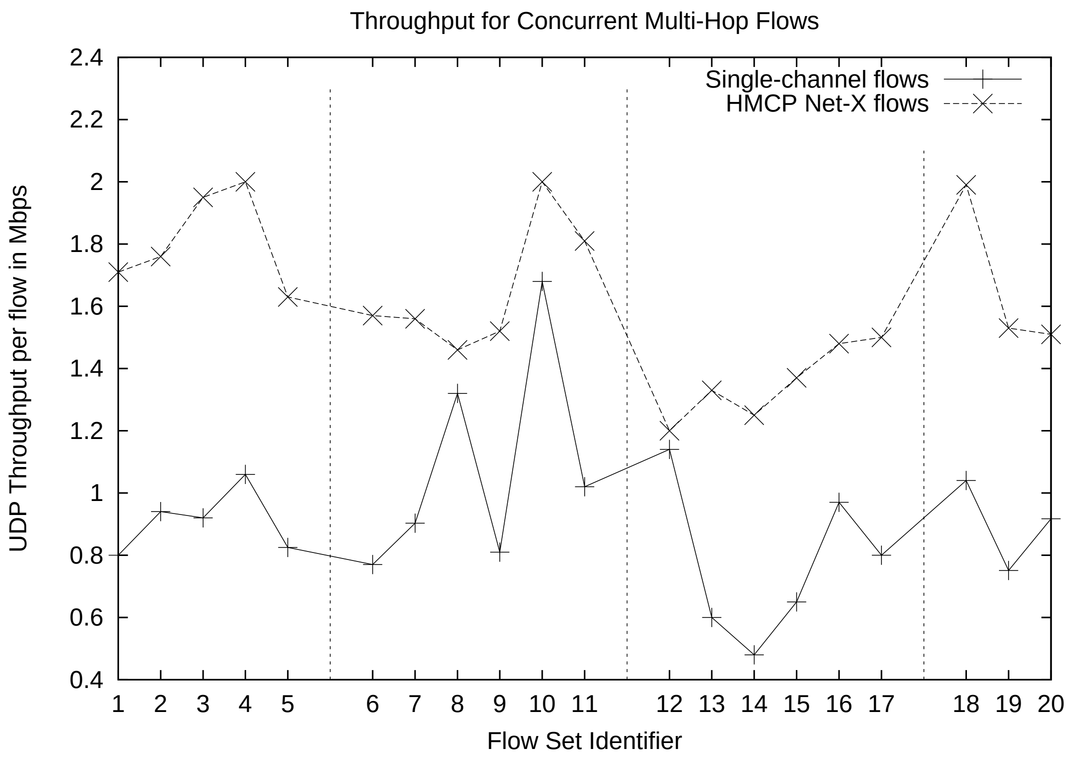


Figure 6.8: Each point in the graph represents UDP throughput per flow for a concurrent flow set. The flow sets with identifiers 1-5 had two flows, sets 6-11 had three flows, sets 12-17 had four flows and sets 18-20 had five flows.

| Flow Set Identifier | Number of Flows | Throughput in Mbps | | | |
|---|---|---|---|---|---|
| | | Single-Channel Flows | | HMCP Net-X Flows | |
| | | Per Flow | Cumulative | Per Flow | Cumulative |
| 1 | 2 | 0.8 | 1.6 | 1.71 | 3.42 |
| 2 | 2 | 0.94 | 1.88 | 1.76 | 3.52 |
| 3 | 2 | 0.92 | 1.84 | 1.95 | 3.9 |
| 4 | 2 | 1.06 | 2.12 | 2.0 | 4 |
| 5 | 2 | 0.825 | 1.65 | 1.63 | 3.26 |
| 6 | 3 | 0.77 | 2.31 | 1.57 | 4.71 |
| 7 | 3 | 0.903 | 2.709 | 1.56 | 4.68 |
| 8 | 3 | 1.32 | 3.96 | 1.46 | 4.38 |
| 9 | 3 | 0.81 | 2.43 | 1.52 | 4.56 |
| 10 | 3 | 1.68 | 5.04 | 2 | 6 |
| 11 | 3 | 1.02 | 3.06 | 1.81 | 5.43 |
| 12 | 4 | 1.14 | 4.56 | 1.2 | 4.8 |
| 13 | 4 | 0.6 | 2.4 | 1.33 | 5.32 |
| 14 | 4 | 0.48 | 1.92 | 1.25 | 5 |
| 15 | 4 | 0.65 | 2.6 | 1.37 | 5.48 |
| 16 | 4 | 0.97 | 3.88 | 1.48 | 5.92 |
| 17 | 4 | 0.8 | 3.2 | 1.5 | 6 |
| 18 | 5 | 1.04 | 5.2 | 1.99 | 9.95 |
| 19 | 5 | 0.751 | 3.755 | 1.53 | 7.65 |
| 20 | 5 | 0.917 | 4.585 | 1.51 | 7.55 |

Table 6.4: UDP Throughput for multi-hop concurrent flows

the set we give a gap of four seconds before setting up the flow for the next pair of nodes in the set. This is necessary since the MCR routing metric needs the channel switching statistics at different nodes to correctly calculate metric cost for different routes. Thus, one should not perform route discovery for all pairs of nodes in the set together. We start the UDP flows one by one for each node pair in the set. We give enough time in between starting of two flows such that the nodes who are involved in forwarding data for the first flow may be able evaluate their channel switching statistics from forwarding data for the flow. Finally, data flows will be established between each pair of nodes in a set. The experiment for a set finishes four seconds after the last flow for the set is established. Thus, for last four seconds in the experiment there would be concurrent data flow between each of the node pairs of the set. For each set we repeat our experiment three times and report the average throughput achieved per flow for each set from the three experiments.

Table 6.4 and figure 6.8 show the variation in throughput achieved per flow for the different sets in our HMCP Net-X testbed and when using single channel in the testbed. We observe that for most of the sets, the HMCP Net-X network achieves twice the throughput achieved when using single channel in the network. We do not observe performance gains of 500% when using five channels in the network as opposed to one. We list some possible reasons why the gains achieved were not as substantial as one might expect:

1. *Channel Reuse on Routes*: The routes set up for the different flows have the same channels used in them. We only use five channels in the HMCP Net-X network while we set up multiple multi-hop flows for our experiments. All the different links (hops) for the different flows can not be on separate channels. This limits the number of concurrent transmissions we can have for the flows in the network. Thus, even though links for each individual flow may be on separate channels, i.e., routes for each flow may be channel diverse, yet different routes would have links on the same channels. If the nodes forming these links are spatially close to each other, the throughput achievable for the flows with these links would be affected. Thus, when setting routes in the network, not only should one try to have the links for each of the route be on separate channels but the links on different routes which are spatially close should also be on separate channels. This would require the routing mechanism to have a global picture of the network and flows and then make routing decisions. We discuss this idea in more detail in Section 8.1.2.

2. *Fixed Data Rate*: We use UDP flows to measure throughput for the concurrent flows. We fix the data rate used for each flow. By fixing the data rate we bound the throughput obtainable for each flow. Thus, even if one can attain better throughput for a flow, our experiments above would not be able to detect it. Using TCP throughput for concurrent flows, with flexible data rate, may give different results (provided channel switching delay overheads are somehow masked).

3. *Cross-Channel Interference*: The different channels used in the HMCP Net-X testbed, in practice, are not entirely orthogonal. Transmission on one channel by a node produces some interference on other channels. The effects of cross-channel interference on flows increase with the number and proximity of concurrent transmissions taking place in the network.

The throughput gains achieved by the HMCP Net-X network for multiple flows is due to its ability to successfully sustain concurrent transmissions close together using different channels. The single-channel network suffers from self-interference when multiple flows are established. With single-channel usage, transmissions from different flows interfere with each other degrading the throughput achievable. The HMCP protocol with the use of an extra interface at each node is able to leverage channel diversity and in our experiemnts was able to, in most cases, almost double the UDP throughput for flows.

# Chapter 7

# Routing Metric

In the previous chapter we saw that channel diverse routes can help improve throughput as compared to using single-channel routes. Channel diverse routes thus should be preferred over routes where a same channel is reused on multiple hops. The WCETT routing metric [8], building on top of the ETX [9] routing metric, provided for preference of channel diverse routes. The MCR [2, 1] routing metric modified the WCETT metric and added the notion of channel switching cost. Apart from expected transmission time (ETT), channel diversity and channel switching costs we observed other factors which affected the throughput achievable for a route. While setting up throughput experiments in our testbed we observed that cross-channel interference can degrade flow throughput significantly for a route. We also observed that channel reuse on a route may not lower throughput compared to throughput for a channel diverse route provided the hops on the same channel are spatially far apart. In this chapter we discuss these observations in detail and propose certain changes to the MCR routing metric which take into account these observations. These modifications constitute the new routing metric, which for now, we call MMCR (Modified MCR). We set up experiments on our testbed to compare the difference in the throughput achieved on the routes chosen by the MCR metric and the new MMCR routing metric we propose.

## 7.1 Modifications to Multi-Channel Routing Metric

While setting up throughput experiments on the HMCP Net-X testbed, we observed that throughput for routes sometimes got affected by factors not taken into account by the multi-channel routing metrics in use (MCR, WCETT, etc.). We discuss these observations in detail and propose modifications to the MCR routing metric to take in consideration these factors that affect route throughput. We use words 'path' and 'route' interchangeably to represent route between two mesh nodes, and use 'link' and 'hop' interchangeably to represent link between two nodes in a route.

### 7.1.1 Cross Channel Interference

We had earlier pointed out in Section 6.1.3, that when a flow is established on a channel diverse route, the nodes in the route must deal with some cross-channel interference. For our multi-channel multi-interface testbed the intermediate nodes in a flow deal with self cross-channel interference the most. These are the nodes that concurrently receive and forward data on separate channels. Since the transmitting antenna and receiving antenna on the node are physically close to each other, even a small amount of cross-channel energy leakage on the transmitting antenna may lead to sufficient interference via the receiving antenna. We observe that if the signal strength on the receiving antenna is not of good quality to start with, the concurrent transmission on the transmitting antenna causes high packet drop rate at the receiving antenna. Further if the receiving channel and the transmitting channel for the node are close to each other (adjacent) in the frequency band spectrum the cross-channel interference observed is usually higher than when the channels are farther apart. Thus, if the intermediate receiving link on a multi-hop route is not of the best quality or otherwise if the receiving channel and transmitting channel for an intermediate node in the route are adjacent in the frequency band spectrum the transmission error rate on the receiving link for the node will increase. In other words, the error rate experienced on an intermediate receiving link in a multi-hop route for a flow could be far worse than the estimate provided by a link quality estimation mechanism. This would be true if the link quality estimation mechanism does not take into account cross-channel interference that may be experienced at a node.

Cross-channel interference may be lowered with better hardware. Still, since the receiving and transmitting antennas are physically very close to each other (on the same node), some cross-channel interference may remain.

Routing metrics such as ETX, WCETT, MCR require accurate link error/success rate estimates for calculating the ETT for a hop. These schemes assume a robust link quality estimation mechanism to help them choose a good route. The different link quality estimation mechanisms in use [18, 9, 8] do not take into account the degradation in wireless link's transmission success rate in the face of cross-channel interference. As discussed above, cross-channel interference can substantially degrade the quality of a wireless link. One requires an intelligent link quality estimation mechanism which takes into account the effects of concurrent transmissions and cross-channel interference. Such a link quality estimation mechanism would be more complex to construct; also it may add further overheads to the system (it may require concurrent transmissions to be set up in order to measure cross-channel interference effects).

Another way to get around the problem would be to make the routing metric cross-channel interference-aware. While choosing routes, we can artificially lower the estimated link quality of an intermediate hop on a route if we consider it to be highly susceptible to cross-channel interference. The question then is how to figure out which links in a route are susceptible to cross-channel interference and then how much should the link quality estimates for such links be degraded. To answer these questions we apply certain heuristics. In our observations we found that the transmission success rate for an intermediate link in a path degraded in two cases:

1. *Adjacent channels*: If the channel on which an intermediate node in a path listens is 'adjacent' (close) in the frequency band spectrum to the channel on which the node concurrently transmits then the receiving link observed higher cross-channel interference. If the transmission success rate for the receiving link (as determined by the link quality estimation) is very good then the link may still be able to handle the interference, otherwise the throughput achievable on the link will degrade. We model this case in our new routing metric MMCR as follows: if the transmission success probability '$p$' for a receiving link is greater than a constant $P_{consec}(< 1)$ (by default set to 0.9) we do not change the link quality estimates for the link. Otherwise if the receiving channel and the transmitting channel for an intermediate node in a route are 'adjacent' on the frequency band spectrum and the success rate for the receiving link $p$ is less than $P_{consec}$, we lower the link quality estimate for the receiving link as $p_{new} = p \times p$. Again this is heuristically done, we lower the link quality as square of the old success probability such that probability degrades, proportional to how poor it is to begin with. If the original success probability is poor the new success probability for the link would be worse. We use five 802.11a channels in our testbed namely channels 36, 48, 64, 149, 161. For our experiments with the new routing metric we consider the following channel combinations to constitute adjacent channel pairs: 36-48, 48-64 and 149-161.

2. *Poor receiving link*: If the quality of the receiving link for the intermediate node is poor to begin with, then concurrent transmission on any channel by the node will further degrade the success rate on the receiving link. That is, even a small amount of cross-channel interference from the transmission would be enough to make the receiving link with average link quality worse. To model this case whenever the transmission success probability $p$ for the receiving link is less than a constant $P_{bad}$ (by default set to $0.7 < P_{consec}$) we lower the success probability $p$ of the receiving link as $p_{new} = p \times p$.

   If both of the above cases apply for a link, we lower the estimated link quality once and not twice. That is, if an intermediate receiving link in a path is of poor quality ($p < P_{bad}$) and also the next hop on the path is on adjacent channel, we lower the transmission success probability for the link as $p_{new} = p \times p$ and this is done once and not twice.

The new routing metric MMCR lowers the estimated quality of certain links in a route which are considered susceptible to cross-channel interference. The lowering of transmission success probability of such links in a path is done while calculating the routing metric cost for the path. This lowering of success probability for a link in a path increases the ETT for transmission of a packet on the link, which in turn increases the cost associated with the path. The values of the probability constants $P_{consec}, P_{bad}$ defined above are specific to our network and were heuristically determined after observations and experiments on the testbed. In general, for good performance, the values for these constants may differ from network to network and will depend on factors such as the transmission power, hardware deployed, data rate, etc. For $P_{consec} = 0$ and $P_{bad} = 0$, MMCR will behave similar to MCR as far as cross-channel interference is concerned.

### 7.1.2 Channel Reuse

The MCR metric combines two components in calculating the metric cost for a path. The first part adds the ETT for all the hops in the path and any switching costs for the hops. The second component adds the ETT costs for the hops on the bottleneck channel in the path. The MCR metric is weighted combination of these two components. The second component of the metric represents the channel diversity of the path and will be small for channel diverse routes. This second component for a path is calculated as the maximum of sum of ETT of all hops on the same channel. The MCR and WCETT metrics do not differentiate between channel reuse on hops that are spatially close versus channel reuse on hops that are farther apart. Thus, a path which has channel reuse on consecutive hops and another path which has channel reuse with a gap of three hops are put in the same category in terms of channel diversity. This we observe is a little pessimistic. Concurrent transmission may be possible on hops spatially far apart even when using the same channel. The 802.11 MAC CSMA/CA protocols would most probably not allow for concurrent transmission on consecutive or alternate hops on a path using the same channel. For links with a gap of two hops between them in a path, one may or may not be able to have concurrent transmissions. For links which have a gap of three or more hops between them in a path, the possibility of having concurrent transmissions with low error rate improves. In any case, treating channel reuse on consecutive or alternate hops in a path same as channel reuse after a gap of three or four hops in most cases will not be justifiable. Then again things would depend a lot on the transmission power, data rate used, different environment variables (path loss, noise, obstructions) in whether one is able to sustain concurrent transmissions on links spatially separated using the same channel. Still, we believe one should have a provision in the routing metric to be able to tweak the metric associated with paths depending on the network characteristics. We relax channel reuse constraints in our new routing metric. We do not impose any metric cost penalty for paths which have channel reuse on hops separated by $CHAN\_REUSE\_CONST$ ($>= 3$, by default set to 3) links or more between them. For paths with channel reuse on links separated by less than $MIN\_HOP$ ($> 1$) hops, the metric cost penalty remains the same as in MCR. The path where the same channel is used on links separated by $h$ ($MIN\_HOP \leq h < CHAN\_REUSE\_CONST$) hops between them we impose only fraction of the penalty cost imposed for channel reuse on consecutive hops or alternate hops in the path. The routing metric calculation for MMCR route is given by :

$$MMCR = (1 - \beta) \times \sum_{i=1}^{n} (ETT_i + SC(c_i)) + \beta \times \max_{1 \leq j \leq K} X_j$$

where $n$ is the total number of hops on the route, $c_i$ is the channel switching cost for the transmitting node on the $i^{th}$ hop, $X_j$ is defined as the sum of ETT of links in the $j^{th}$ channel reuse hop list and $K$ is the total number of channel reuse hop lists in the path. The links in a channel reuse hop list are assumed to interfere with each other's transmission. Links in a path not part of the same channel reuse hop list are considered non-interfering. We choose the maximum of all $X_j$'s to represent the bottleneck portion for the route.

To define channel reuse hop list for a path more formally: it is as an ordered set of links in the path which use the same channel for transmission. A link, $link_{suc}$, is successor of a link, $link_{pred}$, in a channel reuse hop list for path $P$, if and only if:

1. $link_{pred}$ and $link_{suc}$ are on same channel.

2. $link_{pred}$ comes before $link_{suc}$ in path $P$ (when going from source to destination) and the two links are separated by less than $MIN\_HOP$ ($1 < MIN\_HOP < CHAN\_REUSE\_CONST$) links in path $P$. There is one exception to this rule, the last link in the channel reuse hop list may have $h$ ($MIN\_HOP \leq h < CHAN\_REUSE\_CONST$) hops between itself and its predecessor in the list. That is, if $link_{suc}$ is the last link in the channel reuse hop list then $link_{pred}$ and $link_{suc}$ may be separated by $h$ hops in the path, where $MIN\_HOP \leq h < CHAN\_REUSE\_CONST$. For such a case we add only a fraction of the ETT ($ETT_{last\_link} \times \alpha_h$ , where $\alpha_h < 1$) of the last link while calculating the bottleneck ETT sum for the channel reuse list.

The idea here is to divide a path into different channel reuse hop lists. All links in a channel reuse hop list would be considered interfering. Also, we try and differentiate between channel reuse occurring close by on a path and channel reuse occurring on links farther apart. Whenever there is channel reuse on links in the path where the two links have $CHAN\_REUSE\_CONST$ or more hops in between, the two links are considered non-interfering and are part of different channel reuse hop lists. If the two links, using the

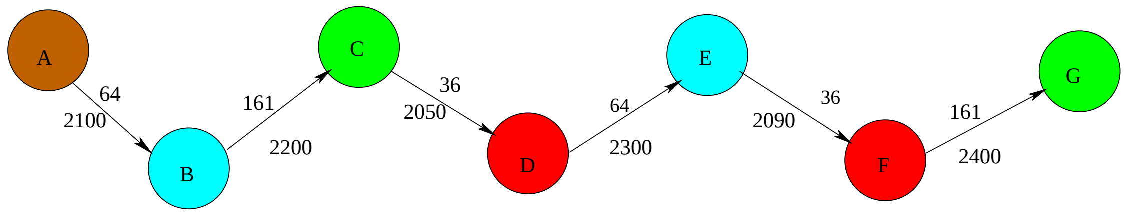


Figure 7.1: A six-hop path with channel reuse.

same channel, have less than $MIN\_HOP (> 1)$ links between them in the path, the two links are considered interfering and are considered part of the same channel reuse hop list. Border case is a link which has $h$ ($MIN\_HOP \leq h < CHAN\_REUSE\_CONST$) hops separating it from the last link on the path which used the same channel. Such a border case link is the last link for the channel reuse hop list continuing from its predecessor link, $h$ hops before it in the path. Also a border case link will be part of two channel reuse hop lists: as the last link of the list continuing from $h$ hops before, and the first link of the hop list starting at it. Apart from the border links all other links on a path are part of only one channel reuse hop list.

The sum of ETT (for a packet) for each link in a channel reuse hop list represents the bottleneck factor for that channel reuse list. If the last link of the list is a border case link, i.e., the last link in the list is separated from it's predecessor in the list by $h$ hops in the path, where $MIN\_HOP \leq h < CHAN\_REUSE\_CONST$, then we add only a fraction of the ETT (for the packet) for this last link in the bottleneck ETT sum for the list. The fraction is given as: $ETT_{last\_link} \times \alpha_h, \alpha_h < 1$. The bottleneck portion for the entire path is defined as the maximum of the ETT bottleneck sums for all the channel reuse hop lists. An important point to note is that the series $\alpha_h$ needs to be a non increasing series, with $\alpha_0 = \alpha_1 = 1$ and $\alpha_m = 0$, $\forall\ m \geq CHAN\_REUSE\_CONST$.

We now provide an example to show calculation of the new routing metric cost for a path in our testbed. We set $MIN\_HOP = 2$ and $CHAN\_REUSE\_CONST = 3$ with $\alpha_2$ to 0.5 as the default values for the constants. Figure 7.1 shows a six-hop path from node A to node G. For each of the links, the figure shows the transmission channel (36, 64 or 161) for the link and the ETT in microseconds for transmission of a 1500 byte frame on the link (with etx of 1 the ETT for transmitting 1500 byte frame with 6 Mbps bit rate $= 2000\ \mu s$). There are a total of 5 channel reuse hop lists for the path.

1. *List 1*: $(A \rightarrow B) \Rightarrow (D \rightarrow E)$ . Links A-B and D-E are separated by 2 hops (B-C, C-D) in the path and use channel 64. Since the links are separated by $MIN\_HOPS$ (=2), we add only fraction of ETT of link D-E for bottleneck calculation for this channel reuse hop list. Bottleneck ETT component for this list will be $X_1 = ETT_{A-B} + \alpha_2 \times ETT_{D-E} = 2100 + 0.5 \times 2300 = 3250\mu s$

2. *List 2*: $(B \rightarrow C)$. Though links B-C and F-G use the same channel but since they are separated

by $CHAN\_REUSE\_CONST$ (=3) hops in the path they are in different channel reuse hop list. $X_2 = ETT_{B-C} = 2200\mu s$

3. *List 3*: $(C \rightarrow D) \Rightarrow (E \rightarrow F)$. Links C-D and E-F use channel 36 and are separated by 1 hop ( $< MIN\_HOP$) in the path. They are part of the same channel reuse hop list. $X_3 = ETT_{C-D} + ETT_{E-F} = 2050 + 2090 = 4140\mu s$

4. *List 4*: $(D \rightarrow E)$ . Link D-E is a border case link and thus is a part of two channel reuse hop lists, *List 1* and *List 4*. $X_4 = ETT_{D-E} = 2300\mu s$

5. *List 5*: $(F \rightarrow G)$. $X_5 = ETT_{F-G} = 2400\mu s$

   Maximum ETT for all channel reuse hop list $= X_3 = 4140\mu s$

The MMCR routing metric cost for path $A \rightarrow B \rightarrow C \rightarrow D \rightarrow E \rightarrow F \rightarrow G$ , for $\beta = 0.5$, assuming all switching costs are 0, will be:

$$\begin{aligned} MMCR &= (1-\beta) \times \sum_{i=1}^{h}(ETT_i + SC(c_i)) + \beta \times \max_{1 \leq j \leq K} X_j \\ &= 0.5 \times (2100 + 2200 + 2050 + 2300 + 2090 + 2400) + 0.5 \times X_3 \\ &= 0.5 \times 13140 + 0.5 \times 4140 \\ MMCR &= 6570 + 2070 = 8640 \end{aligned}$$

The MCR, WCETT routing metrics added the ETT for all hops using the same channel on a path to represent the bottleneck component of the metric cost. Thus, both of these routing metric schemes defined a channel reuse hop list to consist of all links on the path using the same channel. In our new routing metric we do not consider links on a path which have $CHAN\_REUSE\_CONST$ or more hops in between them to be interfering. The value for $CHAN\_REUSE\_CONST$ is subjective, the number of hops between two links in a path to be considered safe enough for non-interfering transmissions can be changed depending on the network characteristics. Further, we believe the border cases where channel reuse occurs before $CHAN\_REUSE\_CONST$ hops but after a certain number of minimum hops ($MIN\_HOP$) should not be considered in the same category as channel reuse on consecutive or alternate hops in a path. For such cases the channel reuse hop list containing the two links in question will stop at this last link and only fraction of the ETT of this last link will be added to the bottleneck ETT sum for the channel reuse hop list during metric cost calculation.

Another interesting question to ask is whether having $CHAN\_REUSE\_CONST$ or more hops between two links on a path implies that the two links are spatially far apart ? The answer is no. Using a reactive routing protocol, a path discovered between a source and a destination can be rather circuitous such that the first hop and the hop at position $CHAN\_REUSE\_CONST + 2$ on the path are actually spatially close to each other. In this case the transmissions on these hops will interfere with each other and our strategy to consider these transmissions non-interfering would fail. The next question to consider then is how to deal with such cases.

The routing metric cost for a path with our new metric consists of two components, the first component adds the ETT (and any switching costs) of all the hops on the route. This component is a measure of the network resources being used by the path. We count on this component to come to our rescue when circuitous paths may escape the channel reuse penalty as described above. For fairly dense network in most cases whenever a circuitous path is discovered between a source and a destination, with high probability a short straight path will be discovered too. Our assumption is that the first component of the routing metric cost will give a preference to the short path instead of the circuitous path which may have escaped a channel reuse penalty. This assumption has been confirmed by the experiments we performed on our testbed. Most of the time when circuitous routes escaped channel reuse penalty, the shorter routes still had lower metric costs due to lesser number of hops. One can make a more intelligent guess as to whether two links are interfering or not if one knows about the neighbor information of the nodes forming the links. If the receiving node of one link can hear the hello packets being sent by the transmitting node of the other link, the two links can be considered as interfering. This neighbor information may or may not be available when the routing metric cost calculation for a path takes place. For proactive routing protocols such as link state routing all neighbor information may be available at the source for the flow itself. For reactive routing protocol implementations could be possible such that neighbor information is available at the node which decides on the interference properties of two links when calculating the routing cost for the path. Currently we have not implemented such intelligent guessing for link interference in our reactive routing protocol. We leave this for future work and currently statically decide to term two links with gap of $CHAN\_REUSE\_CONST$ (set to 3) or more hops between them in a path to be non-interfering.

## 7.2 Performance Analysis

We now test the performance of the new routing metric MMCR as compared to the performance of the MCR metric. We perform our experiments on the HMCP Net-X testbed we have set up. For the new routing metric MMCR we chose $CHAN\_REUSE\_CONST = 3$ , $MIN\_HOP = 2$ and $\alpha_2 = 0.5$. Different source and destination node pairs are chosen and routes are set up between them one at a time first using the MCR metric and then using our new routing metric. For each route set up we use UDP traffic between the nodes to figure out the throughput achievable for the route. We do not use TCP traffic to compare throughput due to the extra channel switching overheads involved with TCP flows (as discussed in Section 6.1.5). Both the old routing metric, MCR, and the new routing metric, MMCR, have the variable $\beta$ associated with them. $\beta$ plays a pivotal role in route determination as it indicates the importance given to the bottleneck portion of the route. We conduct our experiments with two different values of $\beta$, 0.5 and 0.7. Thus in total we compare the performance of four routing metric schemes. First two are the MCR metric schemes with $\beta$ equaling 0.5 and 0.7 and the next two are the MMCR routing metric schemes with $\beta$ values of 0.5 and 0.7. We call these metric schemes by the names $MCR_{B5}, MCR_{B7}, MMCR_{B5}$ and $MMCR_{B7}$ respectively.

For each pair of nodes in our experiment, first a route is established using one of the metric schemes. We then set up three UDP flows one at a time for five seconds each with the data rate of 2.5 Mbps, 3.5 Mbps and 4.4 Mbps respectively. Since UDP protocol does not perform rate control, to figure out the best throughput sustainable for a route we test the route with three different data rates. We choose the best throughput achieved from these three flow experiments. We employ an AODV style reactive routing protocol where RREQs are transmitted as broadcast messages. Due to the broadcast nature of RREQ message, the best route as dictated by a routing metric scheme may not get discovered. This is because the 802.11 broadcast mechanism is not reliable, the RREQ message may not be successfully delivered on the 'best route'. To counter this, the whole procedure, discovering a new route and testing the route with different data flow rates, is done twice. For each pair of nodes and routing metric we perform our throughput experiment twice (thus have two attempts at route discovery) to increase the possibility of the 'best route' as dictated by the routing metric to be discovered and also to give more reliability to our readings.

Also, for each route chosen by a routing metric scheme between a pair of nodes we note down the complete details for the choice. We note down the link details (ETT, channel specification) and the metric cost calculation performed for the route. We also note down the details of all the other routes discovered by the RREQ message but rejected due to higher routing metric costs associated with them. Thus for each routing metric scheme we have complete details of all the routes chosen and all the routes rejected between a pair of nodes with complete channel and link cost details. All these details would help us to compare and contrast the routes chosen by the MCR metric schemes with those chosen by our new MMCR routing metric scheme.

### 7.2.1 Results

We note that for most pairs of nodes the routes chosen by the MCR metric scheme and the MMCR metric schemes are the same. This is expected since the new routing metric scheme differs from MCR in specific cases where cross-channel interference or channel reuse come into picture. Sometimes one of the metric schemes might miss the 'best route' it should have chosen but did not since the route was not discovered owing to the unreliability of the RREQ messages. In such cases the throughput for the 'sub optimal' path chosen by the routing scheme is usually observed to be less than the throughput observed on the 'optimal path' which other routing schemes were able to chose. In our analysis we do not include such cases in which a routing scheme would have chosen the same route as the other routing scheme based on the ETT, channel characteristics of the links involved yet it could not since the 'optimal path' was not discovered. In our analysis we also do not include cases where the routing metric schemes choose different routes and reject routes chosen by the other routing scheme yet the throughput observed on the different routes is almost the same (5 % of each other). The most interesting cases in this category come from the $MMCR_{B7}$ metric case. Couple of times we observed that the $MMCR_{B7}$ routing scheme chose longer circuitous routes between pairs of nodes to avoid the channel reuse penalty while the other routing schemes chose the shorter routes. The short and the long routes both have channel reuse on the same links but the longer route escapes the channel reuse penalty in its routing metric cost calculation due to gap of three or more hops between the channel reuse links. In such cases almost all the time we observed that the throughput achievable by both the short routes and the longer routes between their respective pairs of nodes to be similar. Thus in our experiment the new routing metric scheme MMCR with higher $\beta$ value sometimes used longer routes, though without taking any visible throughput hits. Again such cases may be amended in the future by having a more intelligent routing protocol implementation.

The cases we include in our analysis are those where the routing metric schemes chose different routes between a pair of nodes, rejecting the route chosen by other routing schemes to be sub optimal, and further we observe substantial difference in the throughput for the different routes chosen. We report 13 such cases where we were clearly able to distinguish the difference between the routing schemes and observe difference in the throughput for the routes chosen. We observe that in most cases $MMCR_{B7}$ performs better than the other three routing schemes. $MMCR_{B5}$ performance can be termed as a little better than $MCR_{B7}$ and $MCR_{B5}$. There are a couple of exceptions where $MCR_{B7}$ and $MCR_{B5}$ outperform the MMCR schemes, but the performance difference in such cases is not substantial. Figure 7.2 shows the throughput differences for the 13 cases, also table 7.1 explains the reason why the routing schemes chose routes differently for each case. In table 7.1, we use the abbreviation CCI to mean Cross Channel Interference.

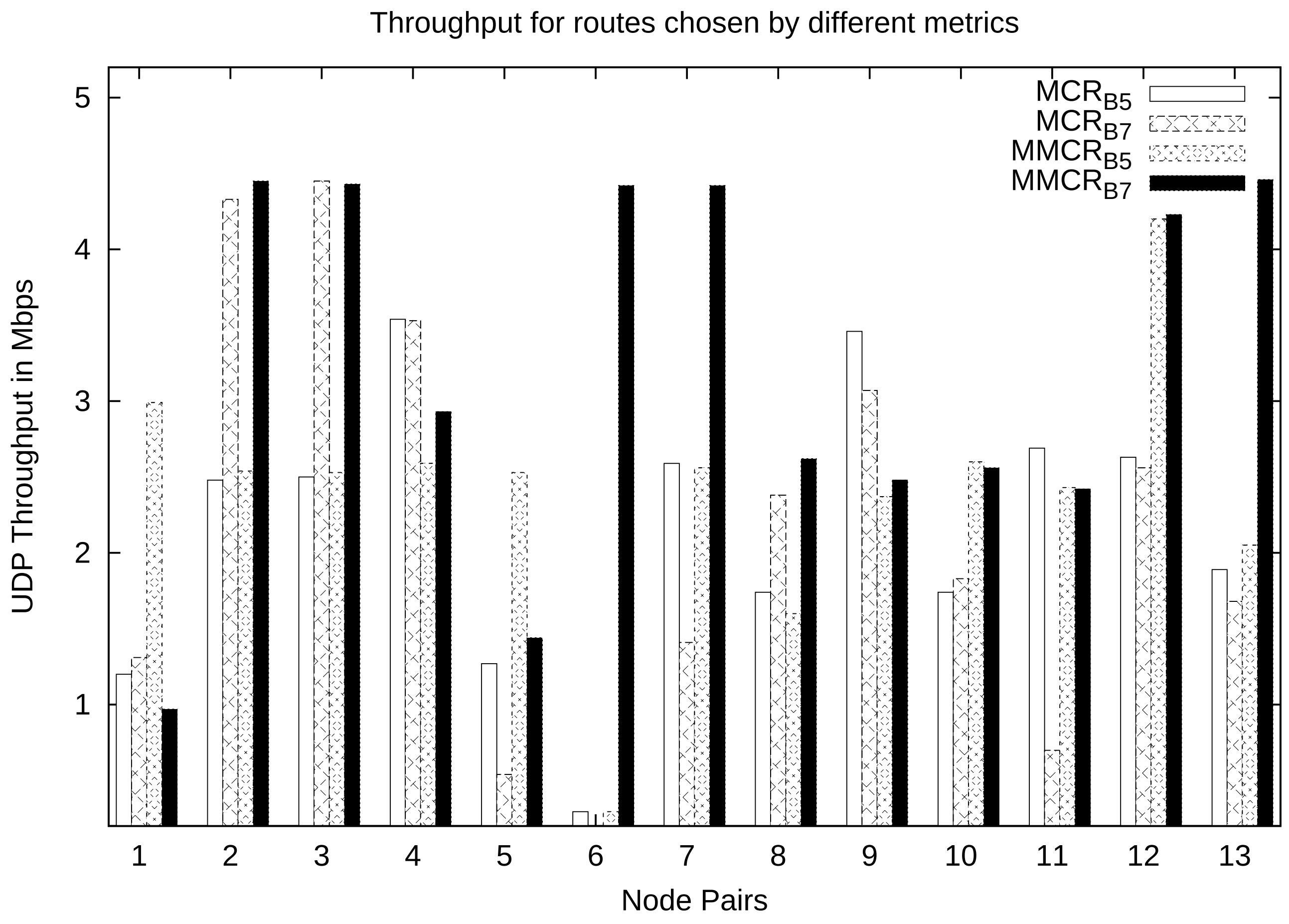


Figure 7.2: Throughput obtained for routes chosen by the different routing metric schemes. We only show results for node pairs for which there is significant difference in the throughput for the routes chosen by the different schemes.

| Node Pair | Achievable Throughput in Mbps | | | | Reason for choosing different route |
|---|---|---|---|---|---|
| | $MCR_{B5}$ | $MCR_{B7}$ | $MMCR_{B5}$ | $MMCR_{B7}$ | |
| 1 | 1.2 | 1.3 | 2.99 | 0.97 | CCI. $MMCR_{B7}$ chose a long circuitous route |
| 2 | 2.48 | 4.33 | 2.54 | 4.45 | $\beta = 0.7$ helps chose a 3 hop route with a lower bottleneck over a 2 hop path |
| 3 | 2.5 | 4.45 | 2.53 | 4.43 | $\beta = 0.7$ helps chose a 2 hop route with lower bottleneck over a 1 hop path |
| 4 | 3.54 | 3.53 | 2.59 | 2.93 | CCI susceptible route performed better than MMCR route |
| 5 | 1.27 | 0.54 | 2.53 | 1.44 | CCI. $MMCR_{B5}$ chose 2 hop path, $MMCR_{B7}$ chose 3 hop path with lower bottleneck |
| 6 | 0.294 | 0.2 | 0.294 | 4.42 | CCI |
| 7 | 2.59 | 1.41 | 2.56 | 4.42 | Channel Reuse |
| 8 | 1.74 | 2.38 | 1.6 | 2.62 | CCI |
| 9 | 3.46 | 3.07 | 2.37 | 2.48 | CCI susceptible 3 hop route performs better than MMCR chosen 4 hop route. |
| 10 | 1.74 | 1.83 | 2.6 | 2.56 | Channel reuse and CCI |
| 11 | 2.69 | 0.699 | 2.43 | 2.42 | $MMCR_{B7}$ chose 6 hop route to avoid channel reuse penalty |
| 12 | 2.63 | 2.56 | 4.2 | 4.23 | CCI |
| 13 | 1.89 | 1.68 | 2.05 | 4.46 | Channel reuse |

Table 7.1: Throughput performance difference for routes chosen by the different routing metric schemes. We use the abbreviation CCI to mean Cross Channel Interference

The MMCR metric schemes differ from the MCR metric schemes in how the bottleneck portion of the routing metric cost is calculated. Now the value of $\beta$ dictates as to how much effect this second component of the routing metric, the bottleneck portion, has on determining the route. We observe that when cross-channel interference susceptible links have been discovered by MMCR metric, or possible channel reuse opportunity is spotted, and thus, the bottleneck portion of metric cost of routes changes from those calculated by MCR, smaller value of $\beta$ may offset these differences by making a preference for the first component of the routing metric. This is the reason why $MMCR_{B5}$ a lot of times accepts the same route as MCR metric even when the bottleneck portion of the routing cost for the routes in question come out to be different in MMCR case as compared to the MCR case. The $MMCR_{B7}$ routing scheme with its higher value of $\beta$ gives more preference to the bottleneck portion of the route and thus is more sensitive to the cross-channel interference and channel reuse components of the MMCR metric. It therefore readily chooses different routes than those chosen by MCR metric when it finds cross-channel interference susceptible links or when it spots a case for channel reuse on links far apart in a path.

With increasing $\beta$ values the point to keep in mind is the chance that long, sometimes circuitous routes, may be preferred between a pair of nodes even though short paths between the nodes give the same or better performance. There is thus a trade off in choosing the value for $\beta$, it should not be too high and neither should it be too low. For our network and the MMCR metric setup, $\beta = 0.7$ works good.

# Chapter 8

# Conclusions and Future Work

We performed an evaluation of a multi-channel multi-interface wireless mesh network. The network employs the hybrid multi-channel protocol (HMCP) [1] implemented over the Net-X system architecture [4, 5]. We analyzed the working of the entire network and nodes forming the network. During the discussion we highlighted several important implementation issues. We also looked at the effects of channel switching delay on the network. The channel switching delay increases the route discovery latency and adversely affects TCP throughput for multi-hop routes. Results from throughput experiments on the HMCP Net-X network were very promising. We noted that the multi-channel network was able to sustain throughput observed for a one-hop UDP flow over multi-hop (channel diverse) routes. With data rate fixed at 6 Mbps, a UDP flow on a multi-hop route was able to attain throughput of around 5 Mbps. For single-channel network a multi-hop flow is affected with self-interference. We observed a UDP throughput of 1.77 Mbps for a three hop route when using a single channel. Similar throughput gains were observed when multiple concurrent flows are established in the network, the HMCP Net-X network was able to support twice the UDP throughput for the multiple flows compared to when a single channel was used in the network. Throughput gains observed for multi-hop TCP flows were not as substantial as for UDP flows. Channel switching delay in the HMCP Net-X network adversely affects TCP throughput on multi-hop routes due to which the gains observed when using multiple channels as compared to a single channel are not as significant for TCP flows.

While setting up the throughput experiments on the HMCP Net-X testbed, we observed that cross-channel interference and efficient channel reuse on routes can affect the throughput achievable for a route. Keeping these observations in mind we proposed certain changes to the MCR routing metric to come up with the new routing metric MMCR. We set up experiments in the testbed to compare and contrast the MMCR metric scheme with MCR metric scheme. We observed several different cases in which the UDP throughput for the route chosen by MMCR scheme was better than the UDP throughput for route chosen by the MCR scheme.

## 8.1 Future Work

We now discuss some ideas for future work in multi-channel networks.

### 8.1.1 Link Quality Estimation

Multi-channel multi-interface networks such as ours require intelligent link quality estimation mechanisms. Currently the link quality estimation mechanisms employed in our testbed and other past schemes [8, 9, 18] look at broadcast packet reception rate to estimate quality of links. Such a mechanism is not cross-channel interference-aware. It does not offer any insight in the degradation of link quality in the wake of cross-channel interference from concurrent transmissions. We try to offset this by having the routing metric be cross-channel interference-aware. A more intelligent link quality estimation mechanism is required which could give good estimate of link quality degradation when concurrent transmissions on different channels occur.

The cross-channel interference is experienced most at the receiving interface of a node which is concurrently transmitting on another channel on the other interface. We observed, for our network, the broadcast hello packets were not being successfully received at the receiving interface of a node when it continuously transmits on the second interface. Since there are no transmission retries with broadcast messages, as with unicast messages, the effects of cross-channel interference become easily visible with broadcast messages. Thus, when a flow is established in the network, the intermediate nodes (nodes which listen and transmit concurrently) in the route can start to miss hello packets being sent by their neighboring nodes including the next-hop node in the route. This can effect the link quality estimate of the link between the intermediate node and the next-hop node on the route and may deem the link as unusable even though the transmission on the link for the flow may be error free. The link quality estimation mechanism needs to be smart enough to know this can happen and should get the link quality estimates from the number of retries being done to transmit a packet to the next hop at the MAC layer and not by the broadcast hello message mechanism. Similar approach is proposed in [18].

### 8.1.2 Intelligent Routing

We now discuss some ideas for improvements in routing protocol in future multichannel wireless networks.

#### Dynamic Fixed Channel Assignment

In HMCP, the fixed channel assignment for a node is done keeping in mind the channel usage in the two-hop neighborhood of the node. This is done to balance fixed channel usage in different neighborhoods in the network. This fixed channel assignment is performed when the network boots up and the nodes discover their neighbors. While setting up flows in the network we noticed that sometimes one could not find good channel diverse paths between a source and destination pair. If one could possibly instruct some of the nodes to change their fixed channels, it may then be possible for a good channel diverse route to be set up between the pair of nodes. Thus, a new channel assignment mechanism is required that may dynamically assign 'fixed' channels to nodes based on the flow requirements in the network. The objective would be to have good channel diverse routes set up between the different source and destination pairs in the network and satisfy the throughput demands for the flows. Such a channel assignment mechanism would need to be closely tied to the routing mechanism in the network and may require global network information. Also the routing mechanism can avoid same channel being used on two hops (on different routes ) which are spatially close to each other (nodes forming the hops are spatially close to each other). That is, not only can the routing mechanism prefer channel diversity on a single route but can also help avoid situations where transmission on different links for different flows interfere with each other.

#### Load-Aware Routing

The routing metric schemes discussed ETT, WCETT, MCR, MMCR are not traffic load aware. This could be another direction that may be taken up for future work. One can have some notion of traffic load in the routing metric schemes. Routes having nodes who are already involved in transmission from different flows should be differentiated from routes with nodes which are not involved in any transmissions. This could help balance traffic flows across the network.

### 8.1.3 Mask Channel Switching Delay

We observed that channel switching delay adversely affected network performance. Channel switching delay for a switchable interface may be masked by using another switchable interface. This idea is discussed in more detail in Section 6.1.5.

# References

[1] Pradeep Kyasanur, “Multichannel Wireless Networks: Capacity and Protocols,” Ph.D. thesis, University of Illinois at Urbana-Champaign, August 2006

[2] Pradeep Kyasanur and Nitin H. Vaidya, “Routing and link-layer protocols for multi-channel multi-interface ad hoc wireless networks,” *Sigmobile Mobile Computing and Communications Review,* vol. 10, pp. 31 – 43, Jan. 2006.

[3] Pradeep Kyasanur and Nitin H. Vaidya, “Routing and interface assignment in multi-channel multi-interface wireless networks,” in *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC),* vol. 4, March 2005, pp. 2051 - 2056.

[4] Chandrakanth Chereddi, “System Architecture for Multichannel Multi-interface Wireless Networks,” M.S. thesis, University of Illinois at Urbana-Champaign, April 2006.

[5] Pradeep Kyasanur, Chandrakanth Chereddi, and Nitin H. Vaidya, “Net-X: System eXtensions for Supporting Multiple Channels, Multiple Interfaces, and Other Interface Capabilities,” Tech. Rep., University of Illinois at Urbana-Champaign, August 2006.

[6] “Net-X: A Wireless Networking Framework providing System eXtensions for Supporting Multiple Channels, Multiple Interfaces, and Other Interface Capabilities,” http://www.crhc.uiuc.edu/wireless/netx.html

[7] Nistha Tripathi, “Rate Control Framework for Net-X,” M.S. thesis, University of Illinois at Urbana-Champaign, August 2007.

[8] Richard Draves, Jitendra Padhye, and Brian Zill, “Routing in Multi-Radio, Multi-Hop Wireless Mesh Networks,” in *ACM Mobicom,* 2004.

[9] D. S. D. Couto, D. Aguayo, J. Bicket, and R. Morris. “A high-throughput path metric for multi-hop wireless routing.” *In Proceedings of ACM MobiCom,* San Diego, CA, Sept. 2003.

[10] C. Perkins, E. Belding-Royer, and S. Das. Ad-hoc on-demand distance vector routing. Internet Request for Comments 3561, July 2003.

[11] Charles E. Perkins and Pravin Bhagwat. Highly dynamic Destination-Sequenced Distance-Vector routing (DSDV) for mobile computers. In *Proceedings of ACM SIGCOMM Conference (SIGCOMM 94),* pages 234 - 244, August 1993.

[12] Piyush Gupta and P. R. Kumar, “The Capacity of Wireless Networks,” *IEEE Transactions on Information Theory,* vol. 46, no. 2, pp. 388 - 404, March 2000.

[13] Pradeep Kyasanur and Nitin H. Vaidya, “Capacity of multi-channel wireless networks: impact of number of channels and interfaces,” in *MobiCom 05: Proceedings of the 11th annual international conference on Mobile computing and networking,* New York, NY, USA, 2005, pp. 43 - 57, ACM Press.

[14] Ashish Raniwala and Tzi-cker Chiueh, “Architecture and algorithms for an IEEE 802.11-based multi-channel wireless mesh network,” in *INFOCOM 2005. Proceedings of the 24th Annual Joint Conference of the IEEE Computer and Communications Societies,* vol. 3, March 2005, pp. 2223 - 2234.

[15] Ranveer Chandra and Paramvir Bahl, Multinet: Connecting to multiple IEEE 802.11 networks using a single wireless card, in *INFOCOM 2004. Proceedings of the 23rd Annual Joint Conference of the IEEE Computer and Communications Societies,* vol. 2, Hong Kong, March 2004, pp. 882 - 893.

[16] H. Lundgren, D. Lundberg, J. Nielsen, E. Nordstrm, and C. Tschudin, “A large-scale testbed for reproducible ad hoc protocol evaluations,” in *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC),* vol. 1, Mar. 2002, pp. 412 - 418.

[17] Krishna N. Ramachandran, Elizabeth M. Belding, Kevin C. Almeroth, and Milind M. Buddhikot, “Interference-Aware Channel Assignment in Multi-Radio Wireless Mesh Networks,” in *INFOCOM,* Barcelona, Spain, Apr 2006.

[18] Kyu-Han Kim and Kang G. Shin, “On Accurate Measurement of Link Quality in Multi-hop Wireless Mesh Networks,” in *Proceedings of ACM MobiCom,* Los Angeles, California, USA, September 2006.

[19] J. Bicket, D. Aguayo, S. Biswas, and R. Morris, “Architecture and evaluation of an unplanned 802.11b mesh network,” in *MobiCom 2005: Proceedings of the 11th Annual International Conference on Mobile Computing and Networking,* 2005, pp. 31 - 42.

[20] D. Aguayo, J. Bicket, S. Biswas, G. Judd, and R. Morris. “Link-level measurements from an 802.11b mesh network”. *In Proceedings of ACM SigComm,* Portland, OR, Aug. 2004.

[21] MIT roofnet. http://www.pdos.lcs.mit.edu/roofnet.

[22] D. Raychaudhuri, I. Seska, S. G. M. Ott, K. Ramachandra, H. Krem, R. Siracus, H. Liu, and M. Singh, “Overview of the orbit radio grid testbed for evaluation of next-generation wireless network protocols,” in *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC),* vol. 3, March 2005, pp. 1664 - 1669.

[23] B. Raman and K. Chebrolu, “Design and evaluation of a new MAC protocol for longdistance 802.11 mesh networks,” in *MobiCom 2005: Proceedings of the 11th Annual International Conference on Mobile Computing and Networking,* 2005, pp. 156 - 169.

[24] K. Ramachandran, E. Belding-Royer, and K. Almeroth, “Overview of the orbit radio grid testbed for evaluation of next-generation wireless network protocols,” in *Proceedings of the First Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks(SECON),* Oct. 2004, pp. 601 - 609.

[25] IEEE Standard for Wireless LAN-Medium Access Control and Physical Layer Specification, P802.11, 1999.

[26] J. Postel. Internet Control Message Protocol. Internet Request for Comments 792, ISI, September 1981.

[27] “Net 4521 hardware from Soekris,” http://www.soekris.com/net4521.htm.

[28] “Atheros Inc,” http://www.atheros.com.

[29] Iperf version 2.0.2, May 2005, http://dast.nlanr.net/Projects/Iperf/.

[30] “Multiband Atheros Driver for WiFi (MADWiFi), BSD-branch CVS snaphot,” May 25th, 2005, http://madwifi.org.

[31] “Linux Netfilter,” http://www.netfilter.org/

[32] “Pebble linux,” http://www.nycwireless.net/pebble.