PERFORMANCE EVALUATION OF THE CACHE AND FORWARD LINK LAYER PROTOCOL IN MULTIHOP WIRELESS SUBNETWORKS

BY AYESHA BINT SALEEM

A thesis submitted to the

Graduate School - New Brunswick

Rutgers, The State University of New Jersey

in partial fulfillment of the requirements

for the degree of

Master of Science

Graduate Program in Electrical and Computer Engineering

Written under the direction of

Professor D. Raychaudhuri

and approved by

_____

_____

_____

_____

New Brunswick, New Jersey

October, 2008

ABSTRACT OF THE THESIS

PERFORMANCE EVALUATION OF THE CACHE AND FORWARD LINK LAYER PROTOCOL IN MULTIHOP WIRELESS SUBNETWORKS

By AYESHA BINT SALEEM

Thesis Director:

Dipankar Raychaudhuri

Cache aNd Forward (CNF) is a clean-slate protocol architecture for content delivery services in the future Internet. CNF is based on the concept of store-and-forward routers with large storage, providing for opportunistic delivery to occasionally disconnected mobile users and for in-network caching of content. This thesis presents the design and evaluation of a reliable link layer protocol for CNF in context of a multi-hop wireless access network scenario. The design aims to deliver files efficiently on a hop-by-hop basis as opposed to an end-to-end approach employed by TCP/IP. CNF leverages in-network storage to store an entire file before forwarding the file to the next node. If a next hop is not currently available or has low transmission quality, files may be stored at each CNF router along the path towards the destination. In addition to the link layer protocol, end-to-end transport in CNF involves packet scheduling and routing at each network node based on cross-layer observation of link speed, MAC congestion, path length, etc. An ns-2 simulation model has been developed for a multi-hop 802.11 access network with CNF routers and a reliable link layer protocol. Baseline results for first-come-first-serve (FCFS) scheduling and shortest path routing show significant capacity gains for CNF relative to TCP for a variety of network topologies and channel models. For example, for a 49 node network with a grid topology, maximum achievable throughput with CNF was found to be 12.5 Mbps as compared to 3.9 Mbps for TCP for a scenario in which the radio links experience short-term outages with a 5% duty cycle. Preliminary results showing the value of simple cross-layer scheduling in CNF are also given in the conclusion.

Acknowledgements

First and foremost, I would like to express my gratitude to my advisor, Dr Dipankar Raychaudhuri for his constant support, guidance and concern. Despite all his commitments and responsibilities he always gave me time and guided me whenever I went to talk to him.

I would also like to thank Professor Yangyong Zhang for her critique on my results and motivating me to find the basic answers first before moving on to more complex scenarios. I thank Dr Sanjoy Paul for his initial guidance to help me get started on my thesis.

I appreciate the help extended to my by many ex- and current WINLAB students during the course of my thesis. I thank Ms. Sumathi Gopal and Mr. Zhibin Wu for their help on getting me started with NS-2, Ms. Hongbo Liu for her help in debugging my NS-2 codes, and Mr. Kishore Ramachandran and Ms. Suli Zhao for their useful inputs. I thank my friends Ms. Tanuja Kumar and Mr. Sumit Satarkar for the discussions that I have had with them regarding my work.

I acknowledge the help that came from Mr. Ivan Seskar, Mr. James Sugrim and Mr. Gautam Bhanage regarding running simulations on unix. I also acknowledge the help from Mr. Ratul Guha in explaining to me his work on channel modeling.

Finally, I would like to thank my mother for being with me and supporting me during the last part of my thesis work.

Dedication

To my parents

Abbreviations

ACK	Acknowledgment
BER	Bit Error Rate
BS	Base Station
CDF	Cumulative Distribution Function
CLAP	Cross Layer Aware transport Protocol
CN	Continuous Noise
CNF	Cache aND Forwarding
CSMA/CA	Carrier Sense Multiple Access/ Collision Avoidance
DCF	Distributed Coordination Function
DIFS	DCF Inter Frame Spacing
DTN	Delay Tolerant Network
FCFS	First Come First Serve
FH	Fixed Host
FTP	File Transfer Protocol
GB	Giga Byte
IP	Internet Protocol
LL	Link Layer
MAC	Medium Access Control
Mbps	Mega bits per second
MH	Mobile Host
MSR	Mobile Support Router
NACK	Negative Acknowledgment
NN	No Noise
NS	Network Simulator
PDR	Packet Delivery Ratio
PER	Packet Error Rate
PN	Pulse Noise
RTT	Round Trip Time
SIFS	Short Inter Frame Spacing
SNR	Signal-to-Noise Ratio
TCP	Transmission Control Protocol
UDP	User Datagram Protocol

TABLE OF CONTENTS

TABLE OF FIGURES

1 INTRODUCTION

Driven by worldwide consumer acceptance of mobile/wireless telecommunication services, the number of wireless end-points connected to the Internet is fast overtaking the number of wired end-points [1]. However, the wireless links remain unpredictable in terms of connectivity, bandwidth and error rates. They suffer from rapid SNR fluctuations which are caused by fading, shadowing, additive noise and interference from other ongoing communications in the neighborhood. The broadcast nature of the wireless channel necessitates that co-located nodes share the radio channel using a medium access control (MAC) protocol. Hence, depending on the traffic load in the vicinity, the bandwidth available to a node also varies as a function of time. The current Internet transport protocol TCP, originally designed to operate over reliable and time-invarying wired links, performs inefficiently over these unreliable and time-varying wireless links. The problem exacerbates when TCP has to transfer data over a multi hop wireless link as there may not be an end-to-end path that is always available. Further, when multiple radio hops are used for access, end-to-end connection quality may be expected to degrade exponentially with the number of hops.

Cache aNd Forward is a novel architecture for the future Internet that aims to deliver content to a potentially large number of nodes, some or all of which could be mobile, intermittently connected to the network or connected to the network over multiple hops. In this thesis, we aim to design a reliable link layer protocol for the Cache aNd Forward (CNF) clean slate Internet architecture project. The design aims to overcome the shortcoming of TCP/IP architecture by optimizing transport on a hop-by-hop basis. It leverages cheap in-network storage to store an entire file before forwarding the file to the next node. If a next hop is not currently available, files are stored at the sender/intermediate nodes. An ns-2 simulation model has been developed for a multi-hop 802.11 access network with CNF routers and a reliable link layer protocol. The results obtained show significant capacity gains for CNF relative to TCP for a variety of network topologies and channel models.

1.1 Overview of TCP:

The current Internet architecture, based on TCP/IP, adopts and end-to-end approach for content delivery from one node to another. In this approach each data unit, (up to 1500 bytes), is transported to the destination individually. Reliability is provided by having the end node send acknowledgments to the sender. The acknowledgments are cumulative in that an acknowledgment of a particular sequence number also serves as an acknowledgment for all the previous sequence numbers. Hence separate ACKs are not necessary for each data unit. The transport layer at the destination is responsible for correct reassembly of the received data before passing it on to the application layer. TCP is a sliding window protocol that allows several packets to be transmitted before stopping to wait for an acknowledgment. All the unacknowledged data packets are maintained in a congestion window, the size of which determines how many unacknowledged packets the sender the can have in its queue at any time. Based on the acknowledgments received, the TCP congestion window size is changed dynamically with the aim of matching the size with the delay-bandwidth product of the end-to-end link. Congestion window size can be increased either in slow start mode or congestion avoidance mode. In slow start, size of congestion window increases by one packet with every data packet acknowledged, causing an exponential increase in size. In the congestion avoidance mode, congestion window size increases by one packet with every Round Trip Time (RTT) worth of packets acknowledged, causing a linear increase in size. TCP maintains a running estimate of the RTT and the standard deviation using the received acknowledgments, in order to calculate timeout values. Packet loss is detected if the TCP sender receives several duplicate acknowledgments or does not receive any acknowledgment during the timeout period. TCP responds to a packets loss by retransmitting the packet and decreasing its congestion window size. The window size is reduced to half if three or more duplicate acknowledgments are received after which TCP resumes operation in congestion avoidance mode. In case of a timeout, the congestion window size drops to one.

It can be seen that while acknowledgments were for meant for providing reliability, TCP also uses the arrival rate of the ACKs for its flow control algorithm. For TCP, loss of an ACK does not only mean loss of a packet but that the loss has occurred due to congestion in the network, which is typically the case in wired networks – hence the reduction in congestion window size. However, in wireless networks, packets losses are

more frequent due to channel conditions rather than congestion at nodes. In this environment, congestion window size is erroneously reduced causing further degradations in throughput. There have been many studies on the performance of TCP over wireless networks such as [2]. Several schemes have been proposed to deal with the effects of the wireless channel on TCP. Indirect-TCP [3] separates the flow and error control of the wireless part from the wired part. Snoop TCP [4] proposes a snoop module at the base station to monitor the packet losses on the wireless link and react accordingly. A cross layer scheme in [5] has been proposed to decouple the transport layer flow control from the error control. We will take more detailed look on theses schemes in section 1.4.

1.2 Problem Statement:

The end-to-end model for TCP requires continuous availability of an end-to-end path for the duration of the connection. In a wireless network, where link quality can change rapidly, it may be difficult to maintain an end-to-end link for entire duration of a large file transfer of the order of ~$100MB-1GB. Further an individual link in the end-to-end path operating at a slower speed would limit the achievable throughput and the links capable of operating at higher speeds would remain underutilized. Other than the underlying end-to-end model, TCP has it own limitations which inhibit its smooth operation over wireless networks. As explained above, flow control and error control are closely coupled in TCP. Any packet loss due to bad link quality in a wireless link is taken to be an indication of congestion, causing TCP to reduce its transmitting rate. TCP also requires continuous flows of acknowledgments from the destination to source to be able to keep sending data packets. In a wireless link, this reverse ACK traffic interferes with the forward bound data traffic further limiting the throughput of the network. This is called the TCP simultaneous send problem [6]

A clean-slate solution to combat the unavailability of an end-to-end path is to replace the end-to-end transport model by a hop-by-hop model. Of course, such a clean-slate network is still speculative and such solutions will not apply to existing TCP/IP networks, but it is worth pursuing this research objective to assess performance gains in the absence of backward compatibility constraints. In the proposed CNF transport model, the entire file is transferred reliably to the next hop if the link condition is good, regardless of the end-to-end link state. The next hop could keep the file in its storage until the next link in the chain becomes available. This

scheme is expected to utilize the network capacity more effectively averaging a lower file transfer delay as compared to TCP, while doing away with the requirement of end-to-end path availability. This approach also has the advantage of retransmission efficiency. Any packet that is lost only needs to be transferred over one link, in contrast to the end-to-end model, where only the source is capable of retransmission.

1.3 Our Goal and Approach:

In this thesis we aim to assess the gains that can be achieved by hop-by-hop approach of file transfer over end-to-end based approach of file transfer i.e. TCP style. Figure 1.1 qualitatively explains the difference between hop-by-hop approach and end-to-end approach.

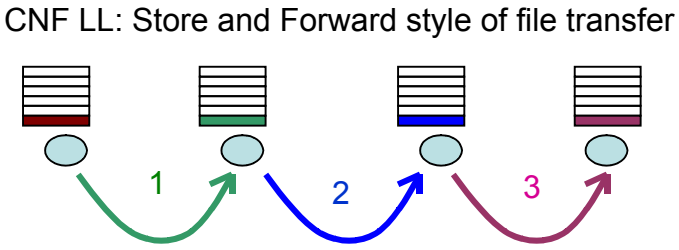

Figure 1-1(a) CNF Style of file transfer

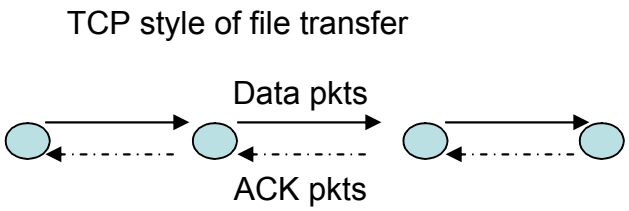

Figure 1-1(b) TCP style of file transfer

In the first case, each hop receives a file completely before starting to transmit it to the next hop. Obviously, these transfers in separated in time and there is no problem of interference from the same flow. The second case shows a TCP style of file transfer where the packets are delivered end-to-end in a streaming manner. Data packets not only suffer from interference with the data packets of the same flow being transferred by the next hop or next-to-next hop, but also from the ACK traffic coming from the receiver in the reverse direction.

We design a reliable link layer protocol for hop-by-hop delivery of files to their destination, and develop an ns-2 simulation model to evaluate the performance of our design vs. that of TCP under various channel and

traffic conditions. We investigate various MAC level optimizations that can be done to further improve the performance of our protocol. Further, we also investigate how link layer scheduling of various files, waiting in queue to be transmitted to their next hop, can be beneficial in view of the time varying nature of the channel.

1.4 Related Work:

To improve TCP performance over a link involving a single wireless hop [7] [3] proposed I-TCP which splits the TCP connection between a Fixed Host (FH) and Mobile Host (MH) into two separate connections; MH to Mobile Support Router (MSR) and MSR to (FH), with some buffering capability at the MSR. The goal is to integrate a single hop wireless link with the wired network by separating the error and flow control of the wireless part from that of the wired part, and by doing so achieve improvements in end-to-end throughput. A separate transport protocol runs on the MH that establishes connection with the MSR. The MSR establishes connection with the FH on behalf of the MH. When a MH moves to another cell, a hand off procedure takes place between the old MSR and the new MSR and remains transparent to the FH. Due to this split connection approach the TCP acknowledgments no longer remain end-to-end but are divided into two separate acknowledgments, one for the wired part and the other for the wireless part. However, [4] and [8] argue that this approach increases the software overhead and base station complexity.

S. Paul et. al used error correction schemes and retransmissions at the link layer to provide a reliable link to the transport layer and hide wireless channel imperfections from the transport layer [9]. However, if TCP runs an independent end-to-end retransmission protocol on top of the link layer retransmission scheme, the interaction of these two independent retransmission protocols can lead to degraded performance [2]. Authors of [8] also argue that when packets are lost and the link layer does not attempt in-order delivery, packets reach the TCP receiver out-of-order causing the receiver to send unnecessary duplicate ACKs to the TCP sender. The sender then goes into fast retransmission and recovery mode and can potentially cause degradations in throughput.

H. Balakrishnan et. al.[4] propose a snoop module at the base stations that caches unacknowledged TCP data packets from FH to MH and performs local retransmissions over the wireless link in case a packet loss is detected based on duplicate acknowledgments and timeouts. For packets flowing from MH to FH the module

detects missing packets at the base station and sends Negative ACKs for them to the MH which retransmits the lost packets. This approach aims to hide temporary situations of poor channel quality and disconnection from the FH.

S. Gopal proposes a new end-to-end based protocol, CLAP, in [5] that addresses the TCP self interference problem and decouples flow and error control. The flow control algorithm no longer depends on feedback packets (ACKs) from the receiver but rather utilizes supplemental cross layer information to estimate bandwidth and adapt its rate to bandwidth fluctuations. The CLAP sender periodically sends probe packets to the CLAP receiver to get an estimate of the end-to-end channel bandwidth available. This information is then used by the flow control algorithm to adapt the sending rate. The error control algorithm uses aggregate NACKs to minimize self interference and duplicate retransmissions.

Delay Tolerant Networks (DTNs) [10] [11] have been proposed as an overlay over existing heterogeneous networks. DTNs employ store-and-forward message (bundle) switching to overcome problems of intermittent connectivity, long delays and high bit error rates. It proposes a bundle layer between application layer and transport layer that is responsible for the store and forward operation of the DTN nodes. Underlying protocol layers remain the same as they are in current Internet.

M. Li et. al. propose a protocol in [12] that is a hop-by-hop transport-level replacement for TCP. In this protocol the object of a single acknowledgment is a block instead of a packet and the acknowledgment is sent at the transport level on a hop-by-hop basis as well as on an end-to-end basis. A file can be divided into many blocks. Each block consists of a number of packets. The transport layer sends out one block at a time. The underlying MAC layer is assumed to be 802.11e [13] which can send a 'txop' burst of packets in one go without waiting for channel access. Hence a block is sent out by the MAC layer in the form of a number of txops which in turn consist of a number of packets. Link layer acknowledgments for individual data packets or txops are disabled. Whenever the transport layer is done with the sending of one block of packets it sends a BSYN packet to the next hop which then sends a bitmap acknowledgment of all the packets in the block. The sending transport layer then retransmits the lost packets to the next hop. Each block is reliably transmitted to the next hop before moving onto the next block. The protocol also attempts to use the advantage of spatial pipelining by having

intermediate hops forward packets to the next hop as soon as it receives at txop worth of packets without waiting for the complete reception of the complete block. It also suggests that the intermediate nodes should cache the blocks to enable an efficient end-to-end retransmission if need arises. As an equivalent of TCP's congestion control mechanism, it proposes back pressure flow control where for each flow a node monitors the difference between the number of blocks received and the number of blocks reliably transmitted to the next hop. If that difference exceeds a certain threshold, the node stops sending block acknowledgments, BACKS, to the previous node. Further, the protocol also attempts to solve hidden node problems by having a receiver that receives BSYN packets from two or more nodes, send a BACK to only one at a time, thus asking only one sender at a time to transmit packets to it. Quality of service requirements for real time applications is delegated to the 802.11e MAC protocol. While this protocol covers a large number of aspects related with wireless file transmissions, there is still room for new ideas. Our protocol, reached independently, follows a similar procedure for reliable file delivery to the next hop. The difference is that we delegate the responsibility of entire file transfer to the next hop to the link layer. The link layer then does the division of files into blocks or batches, as we call it. Our novel contribution is to explore how the storage of files at the intermediate hops can be exploited to achieve schedule transmission of file in a manner that maximizes network capacity.

1.5 Thesis Organization:

The rest of the thesis is organized as follows. We explain the design of our link layer protocol in chapter 2. Chapter 3 details implementation of our protocol in NS-2, simulation scenarios and the results obtained in comparison with TCP. Chapter 4 introduces link layer scheduling and details the results obtained by implementing link layer scheduling in our protocol. Finally Chapter 5 summarizes the conclusions and future work.

2 LINK LAYER PROTOCOL DESIGN

Before we start to discuss our link layer protocol, we provide a brief review of the IEEE 802.11b MAC protocol which is used by our link layer protocol.

2.1 Overview of IEEE 802.11b MAC:

802.11 is a multiple access protocol where different sources contend to access a common channel. It employs Carrier Sense Multiple Access/ Collision Avoidance (CSMA/CA) technique. Each node begins with a default contention window size, say x, and randomly chooses a number from 0 to x-1. This number corresponds to the number of time slots that a node must sense the channel as idle before beginning to transmit. If the node senses the channel as busy during its wait time, the node pauses its timer, and continuously senses the channel to see when it gets free. After a transmission the channel must remain free for at least DIFS amount time before other nodes can resume their timers. When the timer of the node expires, it transmits the data packet. If the data packet was meant for a particular destination, the node expects an acknowledgment from the destination. When the destination receives the data packet, it waits for a SIFS duration after which it sends an ACK. However, if the data packet is sent as a broadcast packet, an ACK is not required. If the acknowledgment for the uni-cast packet does not arrive the sending MAC layer doubles it contention window size and retries sending the packet. It retries for a configurable number of times, doubling the contention window size each time, after which it drops the packet. There are other aspects like of 802.11 like DCF etc. but since they do not play a role in our simulation, they have not been discussed here. The reader can see detailed 802.11 specs in [14].

2.2 Overview of CNF LL protocol:

When a file to be transferred arrives at the link layer, it is stored in the link layer queue. Just before transmitting the files, the link layer assigns a temporarily unique identifier to the file and divides the file to be transferred into a number of batches, where each batch consists of a configurable number, pkts_in_one_batch, of

packets. The link layer then transfers each batch of packets reliably to the next hop, via the underlying MAC protocol, before moving onto the next batch. Reliably is ensured by having the receiving node transmit a NACK after the sending link layer has transmitted a complete batch of packets. This reliability is in addition to the partial reliability that the MAC layer provides. A receiving link layer receives the entire file correctly from the previous hop before handing it over to the routing layer to determine the next hop if applicable.

2.3 Detailed Protocol Design:

As mentioned above when a link receives a file to be transferred to another node from the routing layer, it puts the file in its send queue. Before starting the transmission of the file, the link layer assigns it a temporarily unique link level identifier and divides the file into a number of batches.

2.3.1 CONTROL Packet:

To start with, and after sending each batch of packets, the source link layer sends a CONTROL packet to the next hop, which has the following contents:

TYPE	Link Src Addr	Link Dst Addr	Link File_ID	Filesize	Pkts_in_one_batch	Batch Num	TimeStamp
5 bits	48 bits	48 bits	16 bits	30 bits	13 bits	16 bits	64bits

Figure 2-1 CONTROL packet format

The packets fields are explained below:

TYPE:

It defines one of the three packet types that the link layer protocol uses to transfer data. The three data types are CONTROL, NACK and DATA. Two other types of packets called MAC_SUCCESS and MAC_FAILURE are used for communication between the link layer and the MAC layer of the sending node. 5 bits were allocated to the TYPE field to make the header size an integral number of octets. This leaves room for more packet types to be defined in future. Here the type field would contain an entry of the type CONTROL.

File transfer over each link is identified by a 'Link Source Address', 'Link Destination Address' and a 'Link File Identifier'.

Link Source Address:

We use MAC address of the node transmitting a file as the Link Source Address

Link Destination Address:

We use MAC address of the node receiving a file as the Link Destination Address

Link File_ID:

Combined with link source address and link destination address, it gives a temporarily unique link level identifier of any file being transferred. For a particular Link source and destination address, Link File_id can take values from 0-255 before repeating itself. We impose a constraint on the sending link layer that it cannot have more than 255 files in the process of transmission to the same next hop at any time. This is done to avoid confusion between two different files with the same link level identifier at the receiving link layer.

File size:

It describes size in bytes of the files that is to be transferred. The maximum size supported by 30 bits is 1GB.

Packet in one Batch:

As explained earlier a file to be transmitted is divided into a number of batches. This field tells tell receiving node how many packets one batch o file will contain. This make the number of packets in one batch configurable at protocol level.

Batch Number:

The Batch number field is used to ask the receiver to send a NACK list, explained later, of all the packets in that batch. At the initiation of a connection the batch number field is set to zero and the receiver is required to send a NACK list for the first batch of packets (which are pkts_in_one_batch or lesser in number, depending on the file size). With 16 bits reserved for the batch number field, the maximum number of batches that a file can be divided into is 131071, which means the minimum number of pkts_in_one_batch supported by the protocol for a file of 1GB is 17. The maximum number pkts_in_one_batch, and hence the minimum number of batches, supported by the protocol is determined by the number of packets that can be acknowledged/negative-acknowledged simultaneously in the NACK list. For a NACK list size of 1000 bytes, the maximum number of packets that can be ACKed/NACKed is 8000. This is explained later on.

Time Stamp:

A node sending a control packet is required to time-stamp it. When the receiving node replies back with a NACK it copies the time-stamp from the CONTROL packet into the NACK packet. This is used at the sending node to match NACKs with CONTROL packets.

The size of the CONTROL packet is 30 bytes.

2.3.2 Response from MAC:

After sending the CONTROL packet, the sending node waits for a response from the MAC layer regarding its success in transmitting the packet. If the MAC is successful in transmitting the packet, which it knows by receiving an acknowledgment from the receiver, it sends a MAC_SUCCESS messages to the link layer. Otherwise, if all the MAC retries are exhausted and the transmitting MAC layer does not receive an acknowledgment from the receiving MAC, it sends a MAC_FAILURE message to the Link Layer. The 802.11 code was modified to send back a response to the link layer only for the CONTROL message.

If the Link Layer receives a MAC_SUCCESS message it starts a NACK_WAIT timer. If it receives a MAC_FAILURE message it immediately retransmits the CONTROL message.

2.3.3 NACK Wait Time:

If the underlying MAC protocol is IEEE 802.11, as has been used in our simulations, the duration of the NACK_WAIT timer is calculated using the following formula:

Time = (pkts_in_one_batch+1)[mean_channel_access_time + data_pkt_txmn_time + SIFS + ACK_txmn_time]*

The terms in the above formula are explained below:

pkts_in_one_batch:

This is the number of packets that can be sent in one batch without having to wait for a NACK from the receiver. The Link Layer sends all of the unacknowledged packets in one batch, at the same time to the MAC layer queue. The MAC layer then transmits the packets one-by-one. If we consider that when a node receives a CONTROL packet from another node, it might already be busy sending a file to some other node and might have

sent a pkts_in_one_batch number of packets to the MAC layer queue. In this case any NACK that is generated by the node would first have to wait for the transmission of all the packets already in the MAC queue, which can be up till pkts_in_one_batch. Thus any per packet channel access and transmission time that is calculated later in the formula would have to be multiplied by a factor of (pkts_in_one_batch+1), where '+1' refers to the NACK's own channel access plus transmission time. Transmission of NACK can also be given a high priority by putting them at the head of the queue whenever they are generated. We also try this option as a possible improvement to our protocol.

mean_channel_access_time:

For the sake of obtaining a value for the timer, we use a simplistic method of determining the mean_channel_access_time. MAC 802.11 typically starts with an initial contention window of 32 and a transmission slot is randomly chosen from 0-32. Thus, in the first transmission the mean value of the transmission slot chosen will be 16. Assuming that there are no retransmissions and making an extremely simplifying assumption that there are no transmissions from other nodes during the contention window wait period, the mean channel access time becomes DIFS+16*slot_time. More sophisticated ways of estimating channel access time can be used as has been described in [15].

data_pkt_txmn_time:

The data packet transmission time is to be taken from the perspective of the receiver. The receiver might have data already in its MAC queue to be sent to some other node at some rate. Assuming that the MAC is not operating in auto rate mode, we take the transmission rate of the other nodes to be the same as our own to calculate transmission time of a data packet.

ACK transmission time:

is the transmission time of MAC level acknowledgments at the basic rate which is 1 Mbps for the 802.11b channels under consideration.

SIFS and DIFS:

SIFS and DIFS are Short Inter-frame Spacing and DCF Inter-frame Spacing, respectively, as defined the IEEE 802.11 standard [14].

2.3.4 NACK Packet:

Whenever a node receives a CONTROL packet, it has to reply back with a NACK packet. Using the link level identifier for the file, contained in the CONTROL packet, the receiving node first checks whether it is already receiving this file or has completely received the file before (i.e. the file exists in its records). If it is already in the process of receiving the file or has completely received that file, it creates a NACK corresponding to the batch_num contained in the CONTROL packet. If the file is new for the receiver it creates an entry for the file in its own memory and send back a NACK packet with the NACK for all the packets in batch_num set to one.

We shall now see how a node can differentiate between two different files received at different times with the same link level identifier. A sending node assigns a link level File id to a file just before it sends out the first CONTROL packet for the file. The receiving node records the time when it receives the first CONTROL packet for the file. The receiving node then checks

1) if a file with the same link source and destination address and file_id 255 exists in its records
2) And its receive start time is less than the receive start time of a file with file_id 0.

If both the conditions are true, the receiving nodes deletes entry having a link level identifier same as that contained in the recent CONTROL packet and creates a new entry according to the recently received CONTROL packet. If any condition is not true, a node generates a NACK based on the current entry present in its records. Note that this scheme also requires that a sending node cannot have more than 255 started but unfinished files for the same next hop it its send queue. If a node gets to have 255 unfinished files in its queue which have the same next hop, it must first complete the transmission of the oldest file before starting the transmission of any new file.

Now we shall see how a node generates a NACK. The packet format of a NACK packet is given below:

TYPE	Link Src Addr	Link Dst Addr	Link File_ID	Batch Num	TimeStamp	Padding	NACK List
5 bits	48 bits	48 bits	16 bits	16 bits	64 bits	3 bits	17-8000 bits

Figure 2-2 NACK packet format

In the above packet format, the TYPE field is used to identify a NACK packet. Fields 2-6 are copied from the CONTROL packet that has prompted the generation of a NACK packet. The NACK list contains a bit mapping of received status of the packets with in the batch identified by Batch_num. If the i^{th} packet in the batch has not been received the i^{th} bit in the NACK list is set to 1, else it is set to zero. The NACK packet generated for the first CONTROL packet received has all bits in the NACK list set to 1.

The NACK packet size is 25 + ⌈pkts_in_one_batch/8⌉ bytes

2.3.5 DATA Packet:

When the sending node receives a NACK in response to its control packet that has any or all of the bits set to one, it (re)transmits the DATA packets asked for (bits set to 1) in the NACK packet. If the NACK acknowledges the receipt of all packets with in a particular batch num, i.e. all bits are set to zero, the sender moves on to the next batch of packets.

The format of the DATA packet is described below:

TYPE	Link Src Addr	Link Dst Addr	Link File_ID	Seq Num	Padding	DATA
5 bits	48 bits	48 bits	16 bits	22 bits	5 bits	0-1000 bytes

Figure 2-3 DATA packet format

The TYPE field is set to DATA. The Sequence Number field contains the absolute sequence number of the DATA packet inside the file regardless of which batch it belongs to.

The size of a DATA packet is 18 bytes + payload size. It should be noted that if we limit the maximum DATA payload size in the DATA packet shown in figure 2.3, to 1000 bytes, we also require the NACK list in the NACK packet to not exceed 1000 bytes. Since a NACK covers an entire batch of packets, maximum number of packets in a batch can be 8000 as a NACK list can contain 8000 bits at the most.

2.3.6 Protocol Overhead Calculation:

Assuming that there are no retransmissions, the protocol overhead can be calculated using the following expression

*Overhead = (CONTROL packet size + NACK packet size + DATA packet header size * pkts_in_one_batch) / (pkts_in_one_batch* DATA payload size)*

For pkts_in_one_batch = 1 and DATA payload size = 1000 bytes, the overhead comes out to be 7.4%. For pkts_in_one_batch = 8000 and DATA payload size = 1000 bytes, the overhead comes out to be 1.8%. Thus to achieve a lower protocol overhead, it is desirable to keep the pkts_in_one_batch as high as possible.

2.3.7 Timing Diagram:

An example timing diagram of the proposed link layer protocol is shown below:

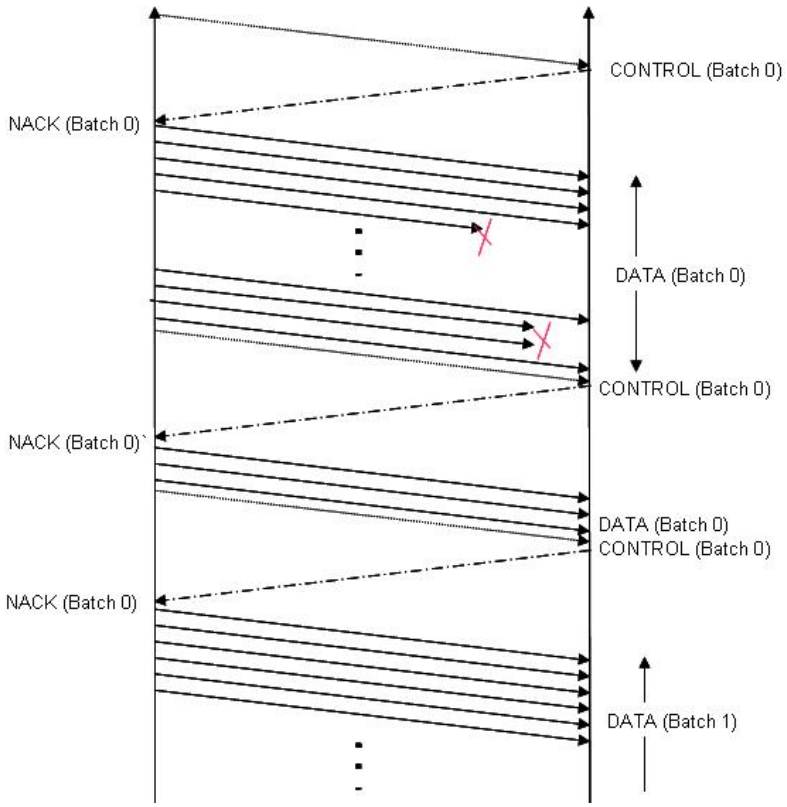

Figure 2-4 Example Timing Diagram

2.3.8 State Diagram:

The state Diagram at any node is shown below:

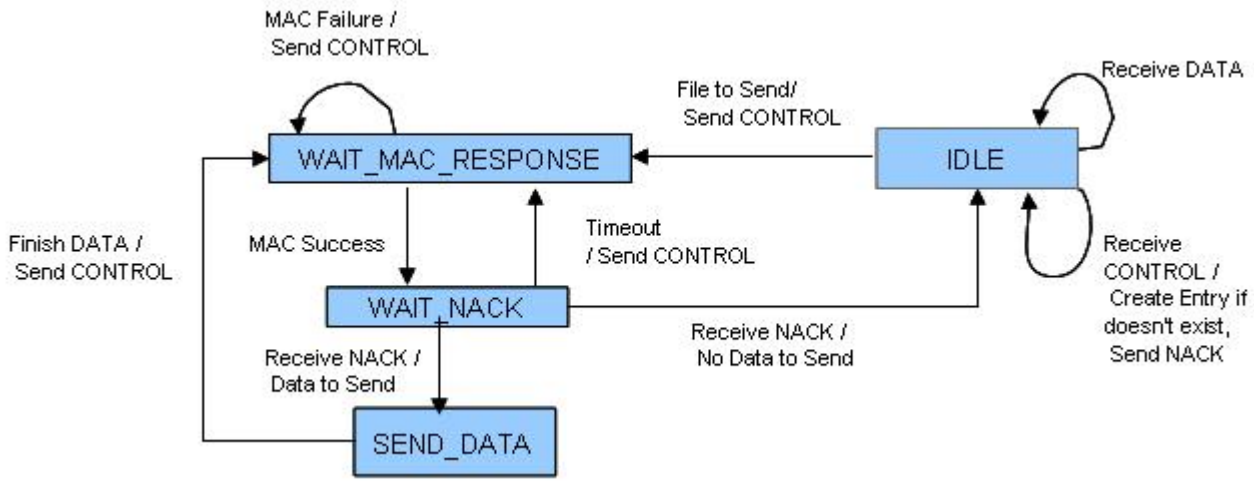

Figure 2-5 State Diagram

2.4 Possible Improvements in the Protocol:

With a baseline protocol designed, we also explore whether the following modifications would give any improvement in the performance of the protocol.

2.4.1 NACK prioritization:

In the version of the protocol described above, a NACK that is generated might have to wait for the transmission of an entire 'pkt_num_in_one_batch' of packets which are ahead of it in the Interface queue between the link layer and the MAC layer. We try prioritizing the NACK and put it ahead of the data packets, if any, already present in the interface queue. This would also change the NACK wait timer to:

mean_channel_access_time + NACK_pkt_txmn_time

2.4.2 Disabling MAC retransmissions and acknowledgments:

In the protocol described above, if the IEEE 802.11 MAC is used there are two levels reliability that are introduced. One level is at the MAC layer and the other at the link layer. When a link layer hands over a batch of data packets to the MAC layer to transmit, the MAC layer tries to reliably transmit each data packet, which is does by sending a data packet and waiting for an acknowledgment from the receiving MAC. If it does not receive an acknowledgment, it tries again. After trying for a limited number of times if it still does not get an acknowledgment, it gives up. A sub-layer above, sending link layer also asks the receiving link layer to send a cumulative NACK list for the batch of packets it just sent, using which it comes to which packets the receiving link layer was not able to receive and retransmits those packets. It seems that when we are anyways retransmitting at the link layer, we can probably disable retransmissions at the MAC layer. This would mean we also don't need the MAC level acknowledgment. As an experiment we try turning off the MAC level acknowledgments and retransmissions for the DATA packets and just keep the responsibility of reliability at the link layer. However, the MAC level acknowledgments and retransmissions if any for the CONTROL and NACK packets remain in place.

3 SIMULATION AND RESULTS

We developed the proposed CNF link layer protocol in Network Simulator-2 [16].

3.1 Implementation in NS-2

The CNF link layer at each node maintains two queues, Send queue and Receive queue. The receive queue is merely a storage space for incoming files from previous hops, until the file transfer is completed. Upon completion the file is handed over to the routing layer. The routing layer then checks if the node itself is the final destination of the file, in which case the file is handed over to the upper layer. If the node itself is not the final destination, the routing layer checks the next hop for the file, and transfers the file to the send queue of the CNF link layer . Before the receiving link layer hands over the file to the routing layer, a record for the file is created that contains the

1. receiving link source address
2. receiving link destination address,
3. receiving link file_id and,
4. File start time at the receiving link.

This record is kept so that in case the previous node did not receive the last NACK, the receiver can still send a zero NACK to the previous node. This record is deleted when a different file with the same link level identifier is added to the receive queue. This happens when the sending link layer has exhausted one round of file_ids and begins to re-use the same file_ids, starting from 0, for newer files. File reception does not require any state at the receiving link layer. A node can receive a file with its CNF link layer being in the IDLE state. Whenever the receiving link layer receives a CONTROL packet, it generates a corresponding NACK, and goes back to its previous state. When it receives a DATA packet, it simply marks the packet as received in its packet-to-receive array and maintains its state.

Whenever a node receives a file from its upper layer, it adds the file to its send queue. The send queue maintains files on a First-Come-First-Serve basis. It starts transmission of the file at its head by sending a CONTROL packet to the next hop. The rest of the file transmission follows as explained in chapter 2. When a

node finishes sending a file, it removes the file from its memory. Conceptually, the flow diagram is shown in figure 3.1

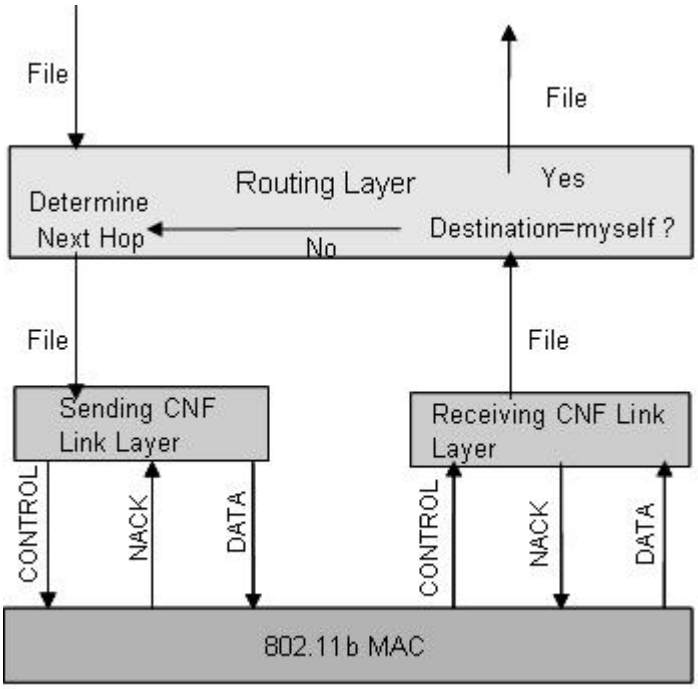

Figure 3-1 CNF LL Flow Diagram

For ease of implementation, the CNF Link layer protocol was developed as an extension to the routing layer. However, this does not affect the functionality of the protocol nor does it cause any effects that could not have been observed if the link layer were implemented in a separate manner from the routing layer.

3.2 Simulation Parameters

We created wireless nodes in NS-2 with different network topologies, traffic models and channel models to compare the performance of TCP protocol vs. CNF LL protocol under various scenarios. For the same network topology, traffic and channel model, NS-2 simulations were carried out to measure average file delay for two cases:

1. Wireless nodes with TCP running at the transport layer, IP at routing layer and LL and IEEE 802.11 at the link and MAC layer respectively.

2. Wireless nodes with UDP running at the transport layer, IP at the routing layer, CNF LL and modified IEEE 802.11 at the link and MAC layer respectively

The protocol stack for both the cases is shown below:

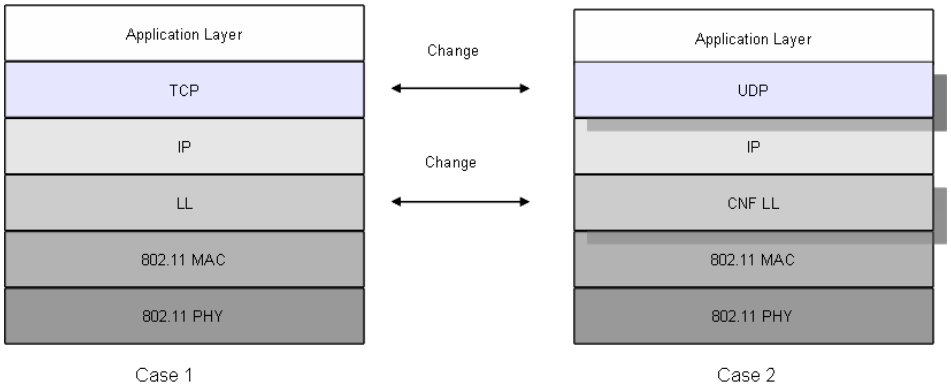

Figure 3-2 Protocol Stacks for TCP and CNF LL as used in the simulation

FTP was used at the application layer. Static routing was used at the IP layer. Static routing agent does not exist in ns-2 for wireless simulations so this was taken from [17] developed by Zhibin Wu. IEEE 802.11b was used as the MAC protocol. The data transmit rate was set to 11Mbps. The transmission range of a transmitter was 250m and the carrier sense/interference range was 550m as shown above. The MAC layer parameters that were used in the simulation are detailed in table 3.1

Parameter	Value
Minimum CW size	32
Maximum CW size	1024
Slot Time	20 us
SIFS	10 us
DIFS	50 us
max retries	7
Transmission rate	11 Mbps
Basic rate	1 Mbps
RTS/CTS	OFF

Table 3-1 MAC parameters in simulation

The propagation model used was two-ray-ground [18].

Our aim is to study and compare the performance of wireless networks running TCP and CNF LL protocols, under different conditions. For larger network topologies we also calculate the offered load that wireless networks running the above protocols are able to carry under various different conditions.

3.3 Simulation with a single flow in a linear topology

The first study that we did was the comparison of TCP and CNF LL protocols in a multihop scenario. We considered a linear multi-hop topology where 1 file of 10MB was supposed to be transferred to the destination. The number of hops between the source and the destination were varied to see the effects of multi-hop transmission. The TCP window size was kept very large so as not to inhibit TCP throughput. Also, buffers at the MAC layer were kept large to avoid packets drops due to buffer overflow.

Figure 3.3 shows the network topology.

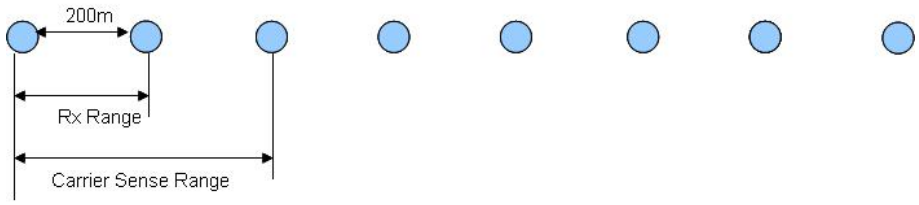

Figure 3-3 A multi-hop linear topology

We considered various Noise Environments and turning ON and OFF of MAC level acknowledgements. The following noise cases were considered while introducing noise

Noise Type	Total Cycle Time (sec)	Duty Cycle	PER during Noise	PER during No Noise
No Noise (NN)	2	0	NA	0%
Pulse Noise (PN)	2	0.5	15%	0%
Continuous Noise (CN)	2	1	15%	NA

Table 3-2 Noise specs in simulation

The simulation results for the Noise free environment are given in figure 3.4.

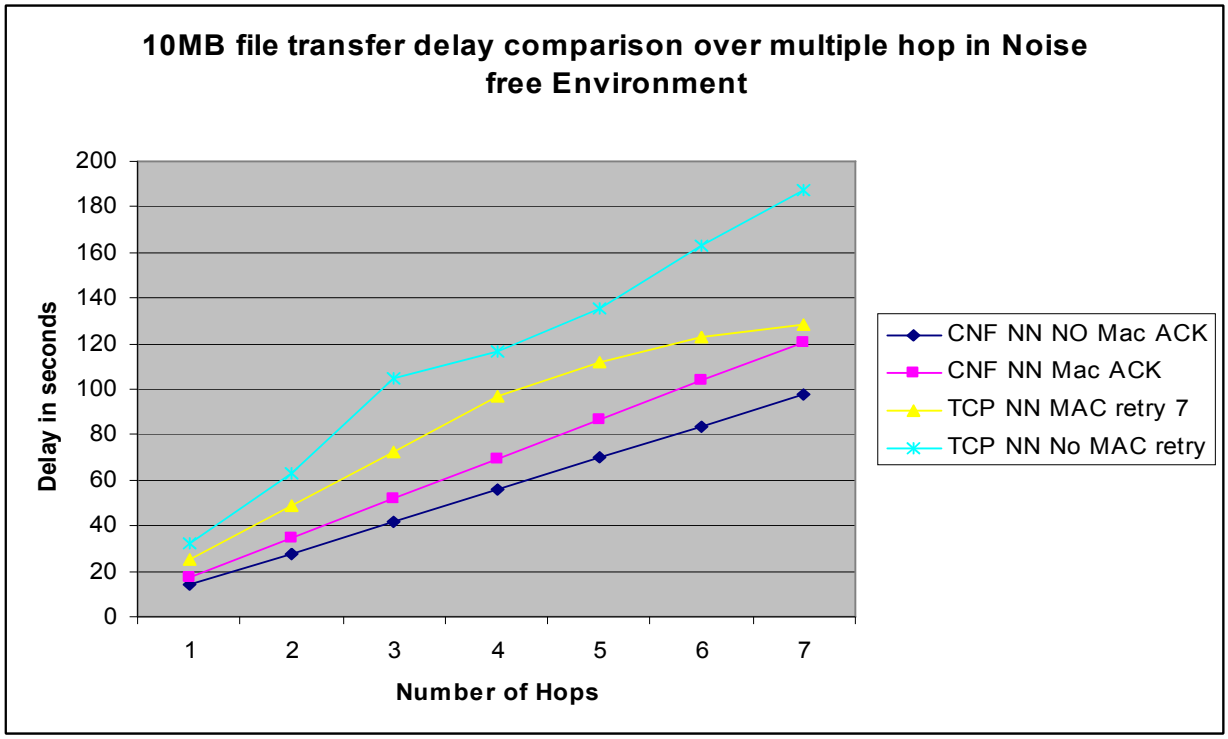

Figure 3-4 File delay comparison in a linear topology in noise free environment

In figure 3.4 we can see that for CNF, the delay without having MAC level acknowledgements and retransmissions is lower as compared to the case where we enable MAC level retransmissions. This is because the Link Layer has its own reliability mechanism that also works hop-by-hop. Therefore, another retransmission mechanism at the MAC layer only serves as an overhead. As an example, when the transmitting MAC does not receive an acknowledgment, it increases its contention window and retransmits the packet. However if the retransmission comes from CNF LL, the MAC layer treats it as a new packet and does not transmit it with increased contention window. The increase in contention window costs time. Moreover, time is also consumed by the SIFS wait time + ACK transmission after each packet. Therefore, disabling the MAC level acknowledgements actually causes lower file transfer delay as can be seen from the results.

In the case of TCP with MACK level retransmissions, we can see that as the number of hops increases, the end-to-end TCP file transfer delay begins to saturate. This is because TCP has the

advantage of spatial pipeline. While one data packet is being transferred from node 0 to node 1, another data packet can be simultaneously transferred from node 6 to node 7. This is an advantage that our hop-by-hop architecture does not have.

In the case of TCP without MAC level retransmission, we see that the delay becomes higher. This is because packets collisions occur at the physical layer when TCP is being used, even if there is only one flow active. The collisions are caused by interference

1. Between the forward bound data traffic and reverse bound ACK traffic
2. Between data packets being transmitted by two different spatially separated nodes.

With MAC level retransmissions these collisions are hidden from the transport layer. But if MAC level retransmissions are disabled, TCP timeouts begin to happen more frequently due to packet losses.

Figure 3.5 shows the results of simulation when carried out in a pulse noise environment.

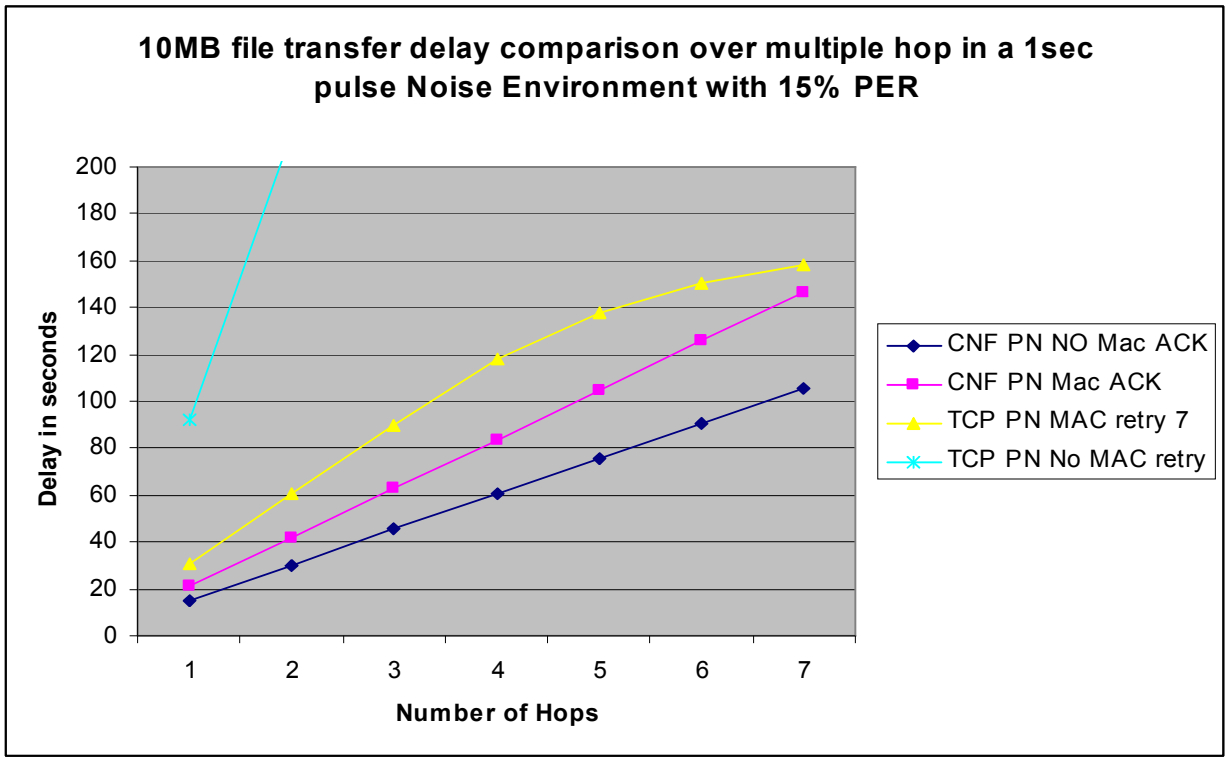

Figure 3-5 File delay comparison in a linear topology in pulsating noise environment

In this scenario, the same effects can be seen as described above. The difference is in case of TCP flow without MAC level retransmissions. Here due to noise packet losses become more frequent and the end-to-end file transfer delay shoots as the number of hops increase.

The simulations for the case of continuous noise are shown in figure 3.6

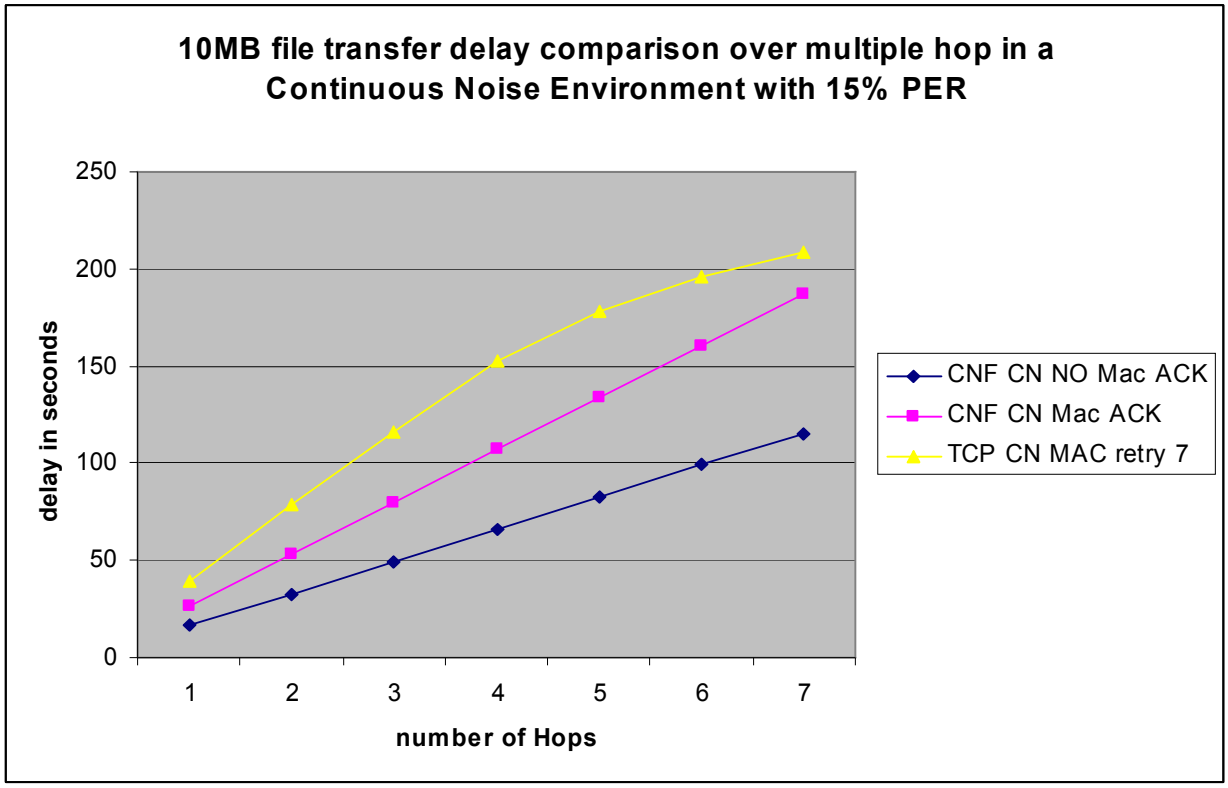

Figure 3-6 File delay comparison in a linear topology in continuous noise environment

Here the file transfer delay for TCP without MAC level retransmission is not shown, because even for a single hop, the delay, which comes to be about 2200 sec, goes beyond the range of the delay axis.

It can be seen that for all the above cases, CNF consistently performs better than TCP. More gains are showed by CNF when used is a multi-flow scenario.

3.4 Simulation with two flows

To study the queuing effects in TCP and CNF LL at a basic level we created two simple topologies shown in figure 3.7.

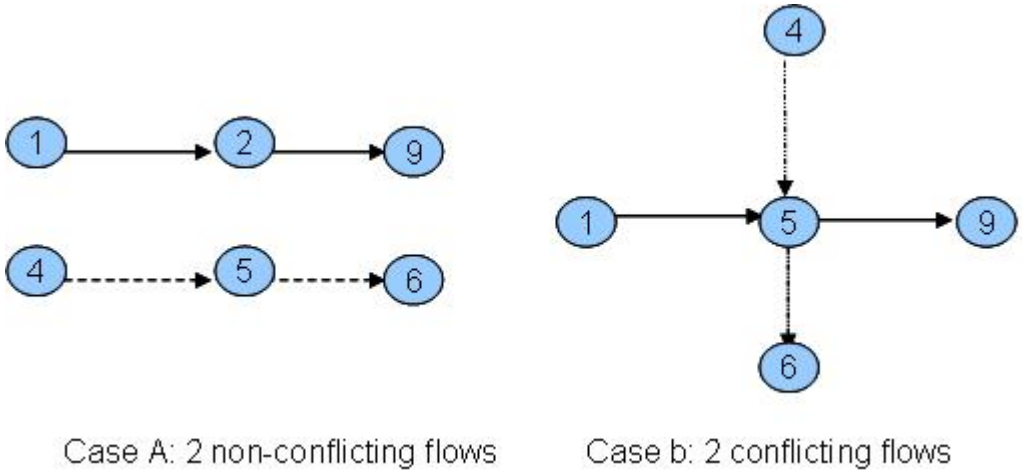

Figure 3-7 Case of two intersecting and non-intersecting flows

In these the above topologies a 50 MB file transfer request arrives at node 1 for destination 9, and at node 4 for destination 6 at the same time. However, in one case the flows intersect and in the other the flows run parallel. We aim to see how TCP and CNF behave in this topology and traffic pattern in various noise environments. Three noise environments were created:

1. Noise-free environment (NN)
2. Pulse noise with 50% duty cycle in a 2 second interval. PER=15% during noise (PN)
3. Continuous noise with 15% PER (CN)

We first consider case A the results for which are given in figures 3.8 and 3.9. In this case the both flows share equal bandwidth while using either CNF or TCP. Also, the delay for each flow in a bandwidth sharing environment is twice the delay for a single flow that has all the bandwidth to itself. It can be seen that delay wise CNF performs better than TCP.

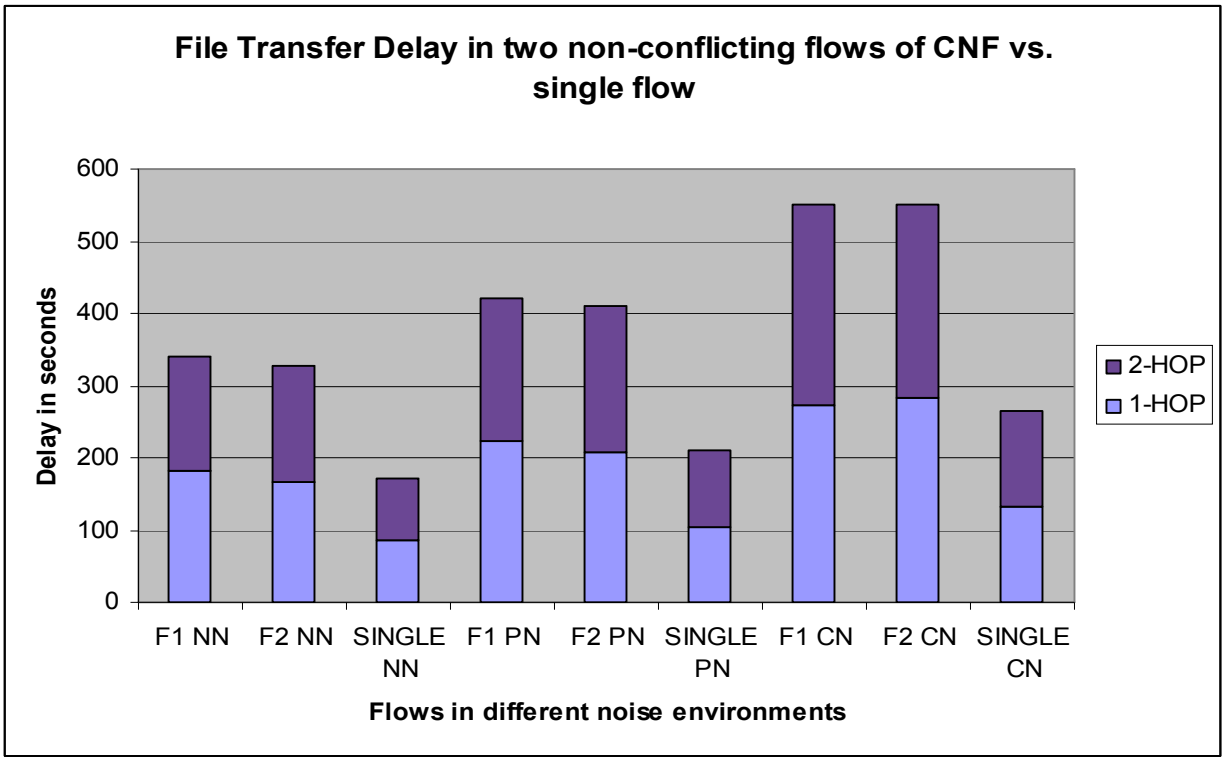

Figure 3-8 CNF Delay for two non-conflicting flows

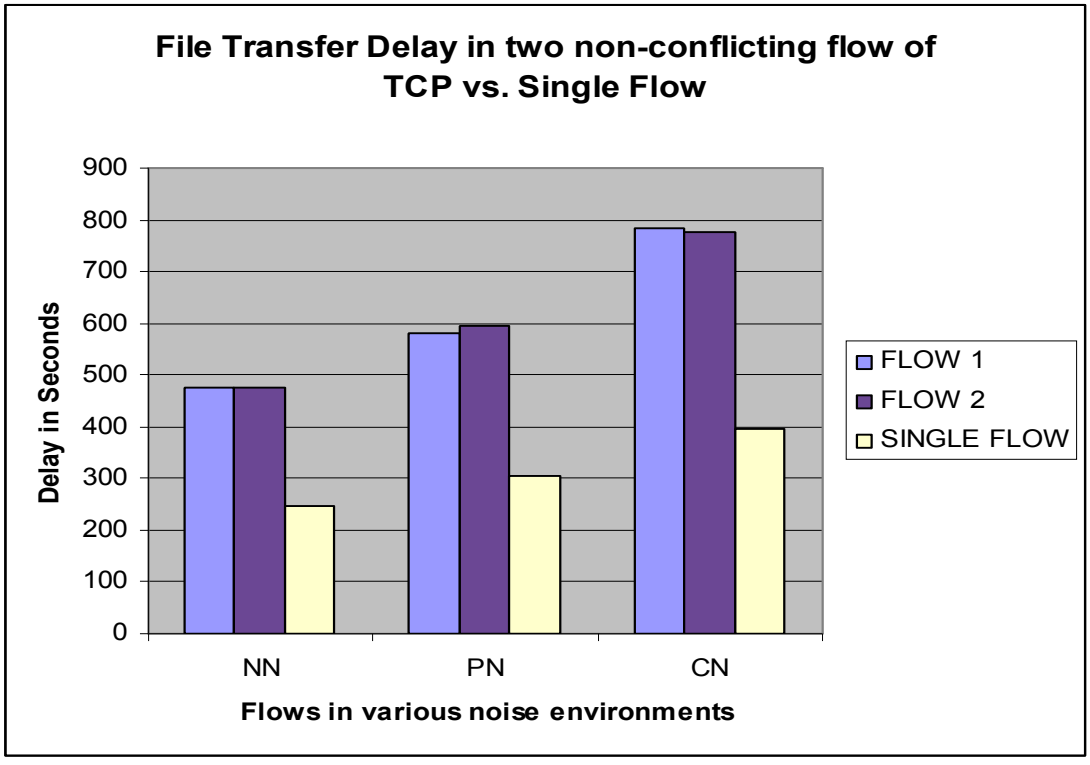

Figure 3-9 TCP Delay for two non-conflicting flows

Next we consider the case of conflicting flows which involves queuing, the results for which are given in figures 3.10 and 3.11. In the case of CNF, both flows share transmission to the first hop equally. But the flow to complete first in the first hop transmission gets an early 2nd hop transmission while the other file has to wait in queue. It should be noted however that the average delay in the conflicting flows becomes lesser than the average delay over non-conflicting flows because none of the flows face contention over the 2nd hop transmission.

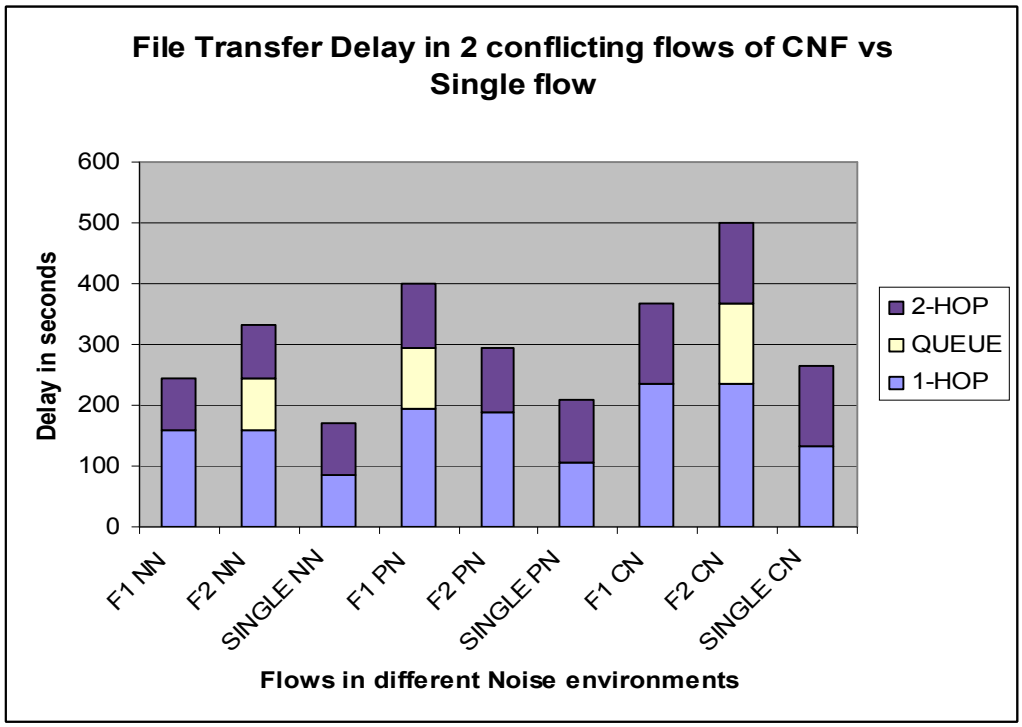

Figure 3-10 CNF Delay for two conflicting flows

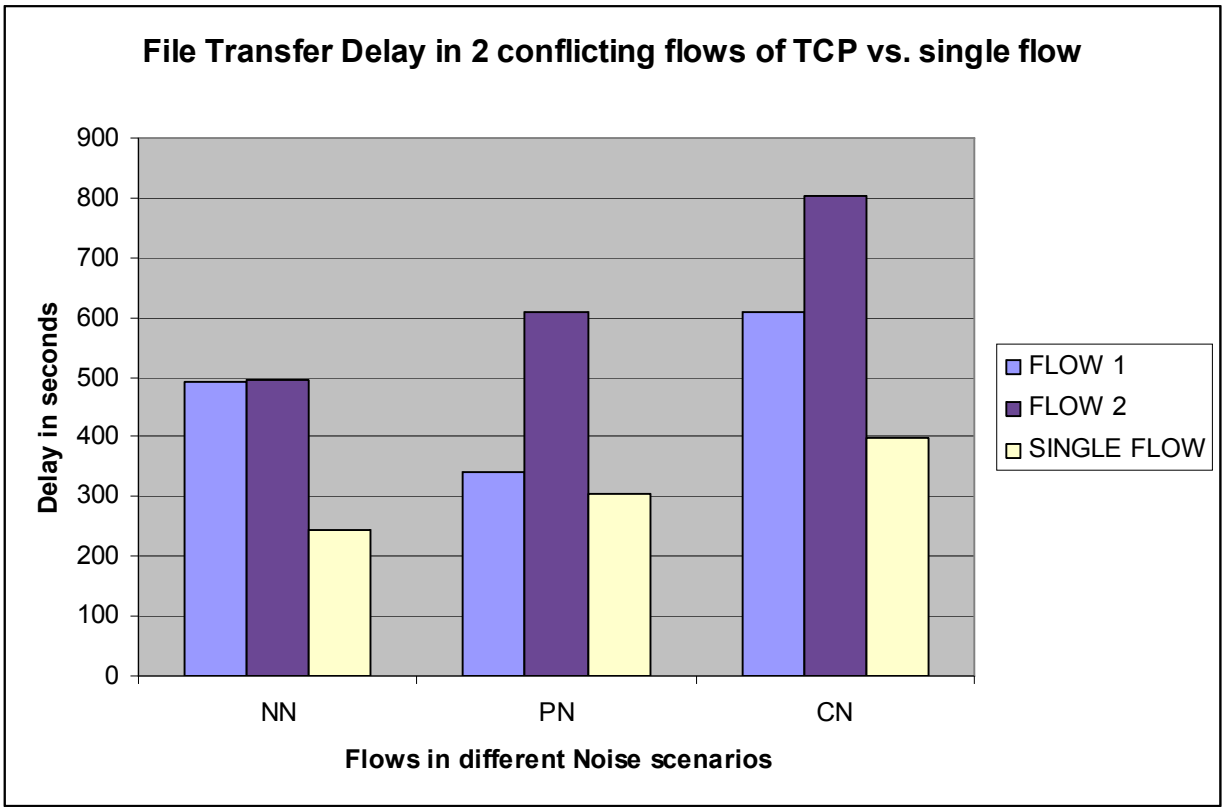

Figure 3-11 TCP Delay for two conflicting flows

While bandwidth sharing was equal for TCP in case of non-conflicting flows, we see that a bias is introduced in TCP when noise is injected. In a noise free environment both flows suffer from equal transmission delay which is about equal to the delays suffered by the flows in the noise free non-conflicting environment. With the introduction of noise TCP becomes more biased towards one flow than the other. Detailed observation of the packet level dynamics revealed that if a timeout occurs for one TCP flow due to a lost packet or delayed acknowledgment, which is a typical event, the other TCP flows takes the entire queue at the middle node. While the queue sizes were taken to be infinite, the amount of traffic of the other TCP flow going through the middle node doesn't let the first TCP flow recover from a timeout. Thus due to this bias the average delay in a noisy environment becomes lesser than the average delay in a noise free environment. Nevertheless the average file transfer delay of CNF remains lesser than TCP.

Figure 3.12 shows the min, max and average delay for 2 conflicting flows using either CNF or TCP with various duty cycles of pulse noise. The thickest lines show the delays for flows in a conflicting environment that

suffer from higher delay (due to queuing) for both TCP and CNF. The thinnest lines show the delays for flows in a conflicting environment that get to have a lower delay for both TCP and CNF. The medium thickness lines show the average of the two. The dotted lines show the TCP delay while the continuous lines show the CNF delays. It can be seen that even with the TCP queuing bias introduced; the CNF average delay remains lower than the TCP average delay in all the noise environments.

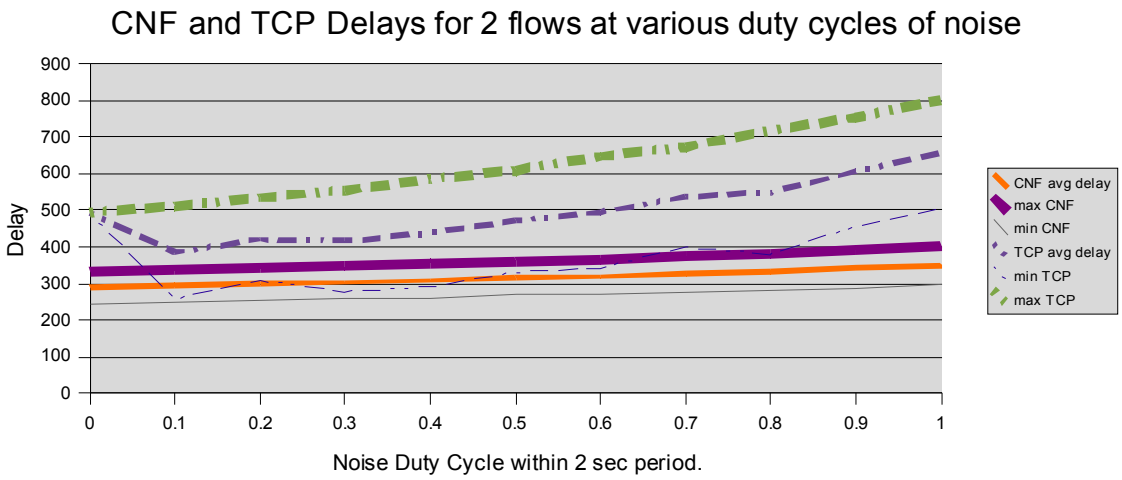

Figure 3-12 CNF and TCP file delay comparison for two conflicting flows

3.5 Simulation for a grid topology with continually arriving flows

For performance evaluation of the CNF protocols in wireless multi-hop environments, a 7x7 grid of 49 nodes was used, as shown in figure 3.13.

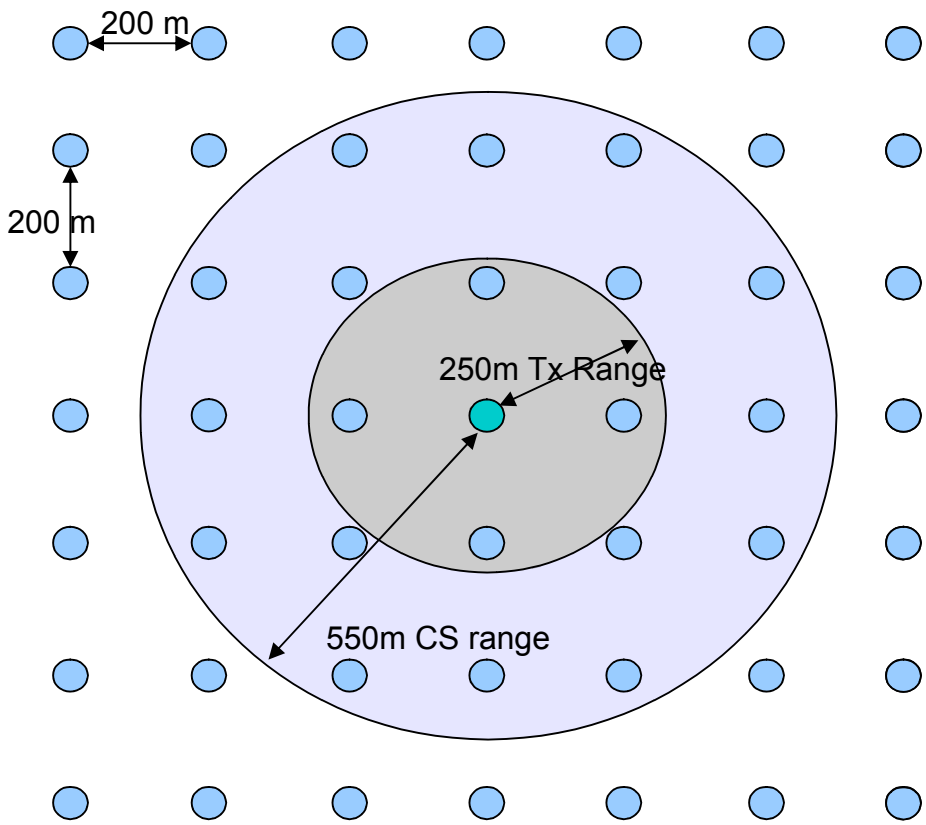

Figure 3-13 Grid Topology

3.5.1 Poisson Arrival Traffic Pattern:

Each node gets a request for a 10MB file transfer to a random destination in the grid at the rate of λ per hour, where λ is a Poisson variable. Incoming files (from the upper layers in case of the sending link layer and from the previous hops in case of the intermediate link layers) are queued and forwarded to the next hops on a first-come-first-serve basis. To allow some traffic buildup in the network before we start our measurements, we first let the first 450 files entering into the system (i.e. files coming from upper layers as a result of user generated requests and not just coming form previous hops) to pass through. Then we mark the 550 files transfer requests that come from the upper layers and note the time to completion for each of the marked file and take the average.

Since the source destination pair for each file is random, the number of hops that each file has to go through is also random. Considering all possible combinations of source destination pairs in the 49-node grid we find the average hop length to be 4.63.

With all these parameters the offered load of the network is calculated is

Offered Load = λ * *file size* *8* *number of nodes* * *average hop length* / 3600 sec

We then set an average file delay limit of 500 sec. By varying λ we change the offered load of the network and note the average file delay. The *capacity* of the network is then defined as the amount of offered load for which the average file delay remains below the specified delay limit.

Average file delays for varying networks loads were obtained for the following cases

1. TCP at the transport layer, IP at routing layer and LL and IEEE 802.11 at the link and MAC layer respectively.
2. UDP at the transport layer, IP at the routing layer, CNF LL and modified IEEE 802.11 at the link and MAC layer respectively.

3.5.2 Initial Results

The results obtained are shown in figure 3.14.

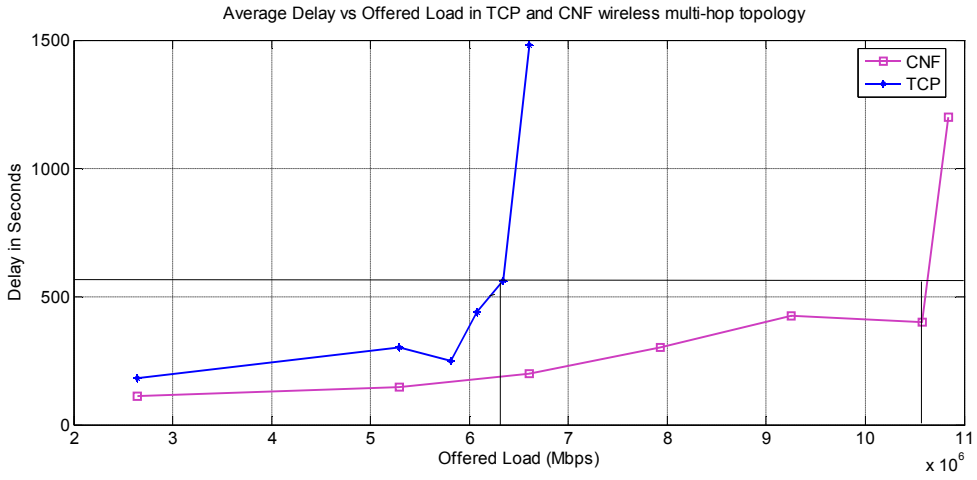

Figure 3-14 Performance graph of TCP vs. CNF LL in a Poisson Arrival traffic pattern

It can be seen that for all values of offered load, the CNF LL outperforms TCP. For the delay limit of 500 seconds, the capacity of the network running TCP is about 6.2 Mbps while the capacity of the network running CNF is about 10.6 Mbps. This gives the CNF LL a gain of about 70% over TCP in terms of network capacity.

Figures 3.15 and 3.16, show the CDF of the file transfer delay at various offered loads when using CNF and TCP protocols, respectively.

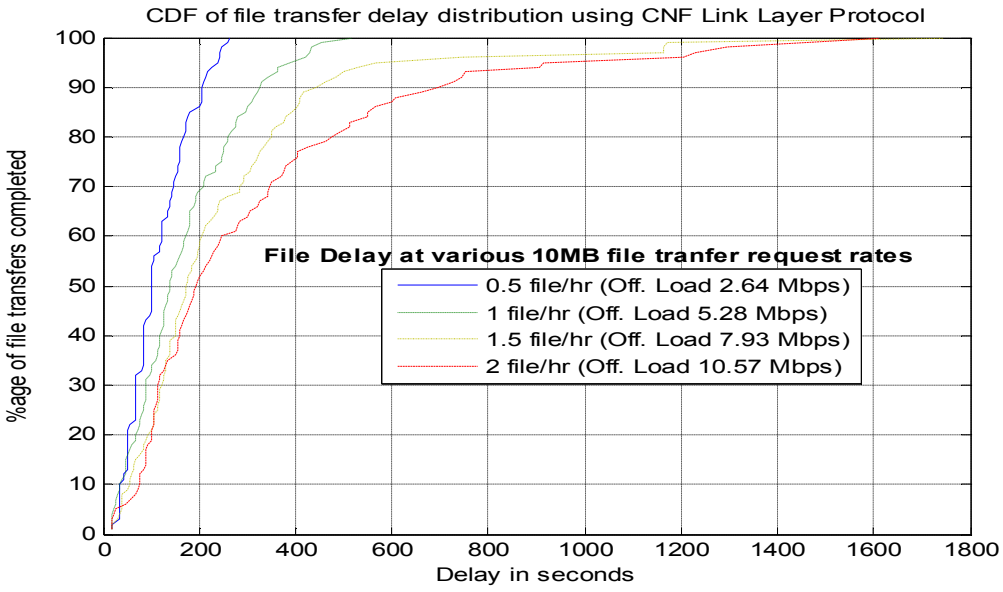

Figure 3-15 CDF of file transfer delay when using CNF LL

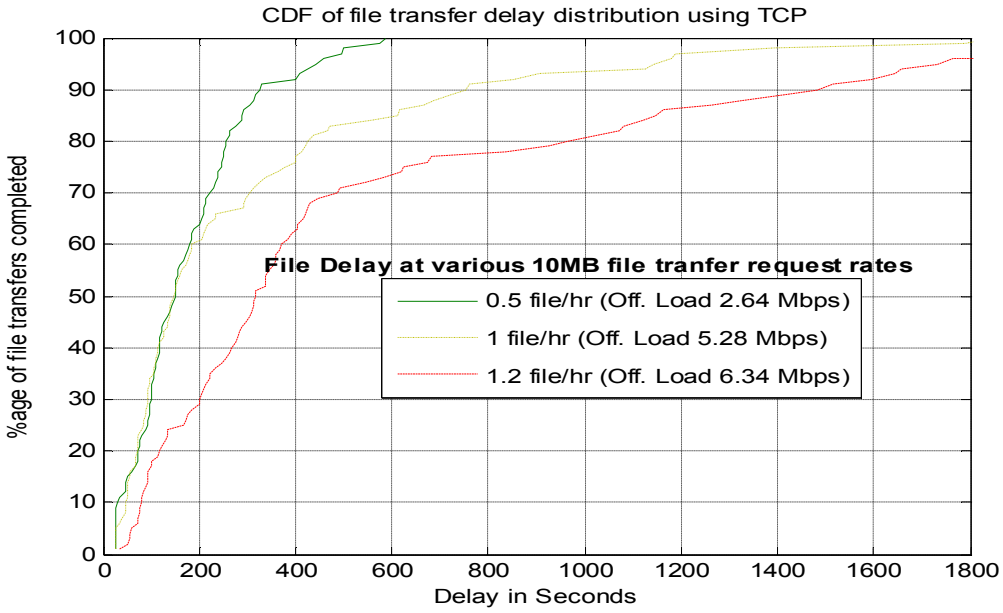

Figure 3-16 CDF of file transfer delay when using TCP

If we compare the performance of the two protocols at the offered load of 5.28 Mbps, we see that 60% of the files are completed around 200 seconds for both cases. However, the remaining 40% in CNF take another 350 seconds to complete while they take another 1600 seconds to complete in case of TCP. This is because, when in competition the TCP protocol shuts off for some flows that have greater Round Trip Times, which here means greater hop length. This TCP unfairness issue has been noted in many studies like [19]. Thus for the same offered load TCP not only suffers from higher average delay but also a higher variance as compared to CNF LL.

Next we tried to evaluate our protocol under different traffic and channel conditions.

3.5.3 Client-Server Model:

The next model that we tried was a Client-Server model. We introduced 7 servers in to the topology above so that file transfers would only start from one of the seven servers to any random destination. Thus our topology became a 7x8 node grid where only 7 nodes generate files, and rests just receive them. The requests for 10 MB file downloads were assumed to be coming from the 49 clients in a random manner, at the rate of λ per hour per client, where λ is a Poisson variable.

Figure 3.17 shows the client-server topology.

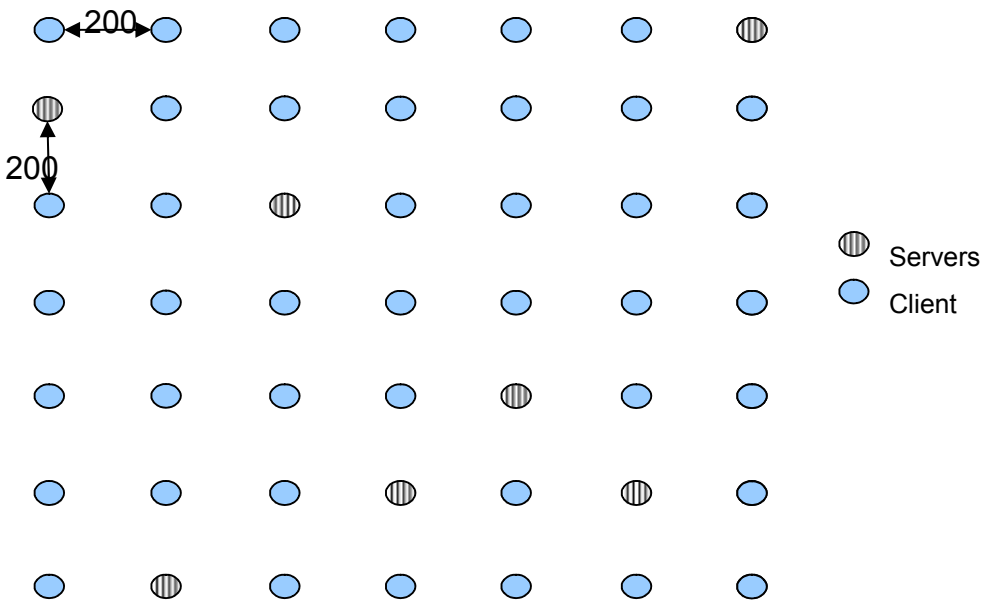

Figure 3-17 Simulation Topology with the introduction of 7 servers

With the average file delay limit kept at 500 seconds capacity of this network was again calculated for both TCP and CNF. It should be noted here, that since the source-destination pairs are now changed. With the source always set to one of the seven servers, and destination being a random node as before, the average hop length now changes with the new value being 4.83. It was found that capacity of the network when using TCP and CNF was close to their previous values of 6.2 and 10.6 Mbps, respectively. Hence our client-server model did not introduce any significant skew in the network capacities.

3.5.4 Bursty-Source Model:

After this, we tried another traffic model called bursty source model. In this model each source would generate file transfer requests in bursts, injecting a lot of traffic with in a short duration and then remaining silent for a longer period. This traffic pattern causes more congestion in the network. We however assume that the active periods of different sources are not synchronized i.e. different sources inject bursty traffic at different times in the network.

For the purpose of our simulation each node was assumed to have an ON period in which it would generate files that are to be transferred to other nodes and an OFF period where the source itself would not generate any traffic but rather act only as a relay of the current traffic in the network. The OFF and ON times of all nodes were desynchronized and every node woke up and went to sleep at a random time. When a node wakes up it remains in the ON state for an exponential duration with mean 1200 seconds. After that a node goes to sleep for an exponential duration with mean 4800 sec. Thus on average a source remained in the ON state 20% of the time and in the OFF state 80% of the time. In terms of the block diagram shown above the behavior of the nodes in the ON state and OFF state would look as shown in figure 3.18

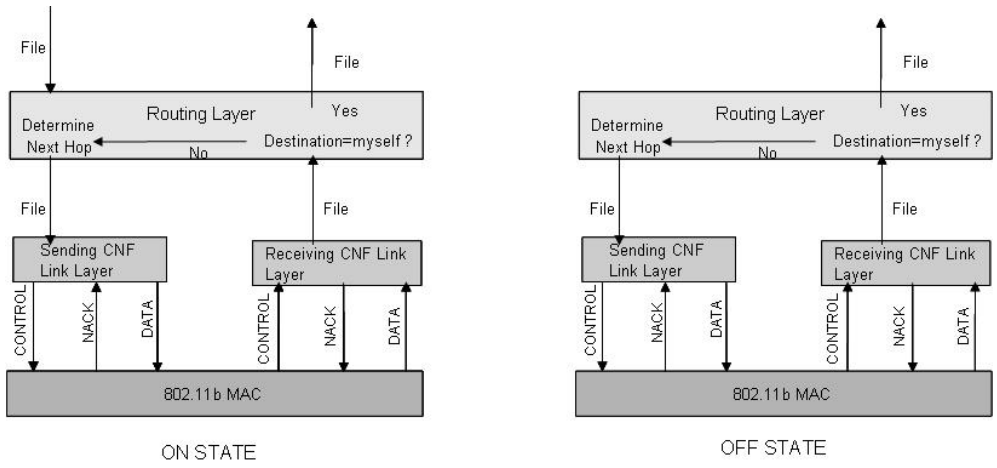

Figure 3-18 Flow Diagram of CNF LL in ON-OFF state

The topology of the first case of 49 nodes arranged in a 7x7 grid was used. Network capacities were calculated for both CNF and TCP and it was found the CNF LL capacity decreased by 13% and TCP capacity was decreased by 19% of its value as compared to an always ON Poisson arrival model of traffic. The yielded capacities for CNF and TCP are shown below:

	Capacity (Mbps)
TCP	5.2
CNF LL	9.2

It should be noted that these capacity values are the values of maximum offered load for which the average file delay for both TCP and CNF is ~ 500 seconds. Thus capacity of the network while using CNF LL remained 77% greater than the capacity of the network while using TCP in bursty source model.

3.5.5 Markovian Noise Channel Model:

Bursty errors were simulated at the packet level in NS-2 by introducing a 2-state Markovian Noise model (similar to Gilbert-Elliot model [20] but introduced at a packet level). For each link in the network the channel was divided in two states

1. Good State with 5% Packet Error Rate (PER)
2. Bad State with 90% PER

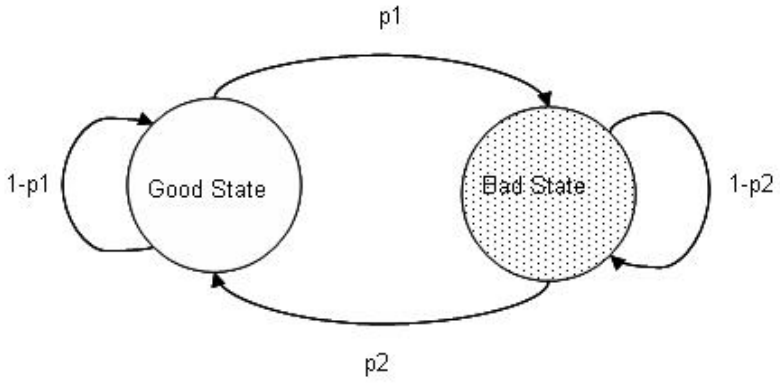

Figure 3-19 Two state Markov model for the channel

However, the links were considered to be symmetric i.e. link quality from node A to node B was equal to the link quality from node B to node A. The duty cycle of the noise, which depends on the probabilities p1 and p2, was varied to see the effect on channel capacity when using TCP or CNF LL. The resulting capacities while using TCP and CNF in the network are shown below.

	Capacity (Mbps) when using Markovian Noise with duty cycle		
	1.00%	5.00%	10.00%
TCP	5	3.7	0
CNF	8.5	8	7.4

It can be noted that

1. For 1% Markovian noise CNF gains were about 63% over TCP,
2. For 5% noise CNF gains were 110% over TCP and,
3. For 10% Markovian noise, TCP file transfer delay went beyond the average file transfer delay limit even for a very lightly loaded network, while CNF achieved capacity up to 7.4 Mbps.

3.5.6 Results

The simulated capacities of the network, when using TCP and store-and-forward based CNF link layer, are shown in figure 3.20 for various traffic and channel conditions. Note that all these results are for static shortest path routing used for both TCP and CNF. MAC layer acknowledgments were used for all packets.

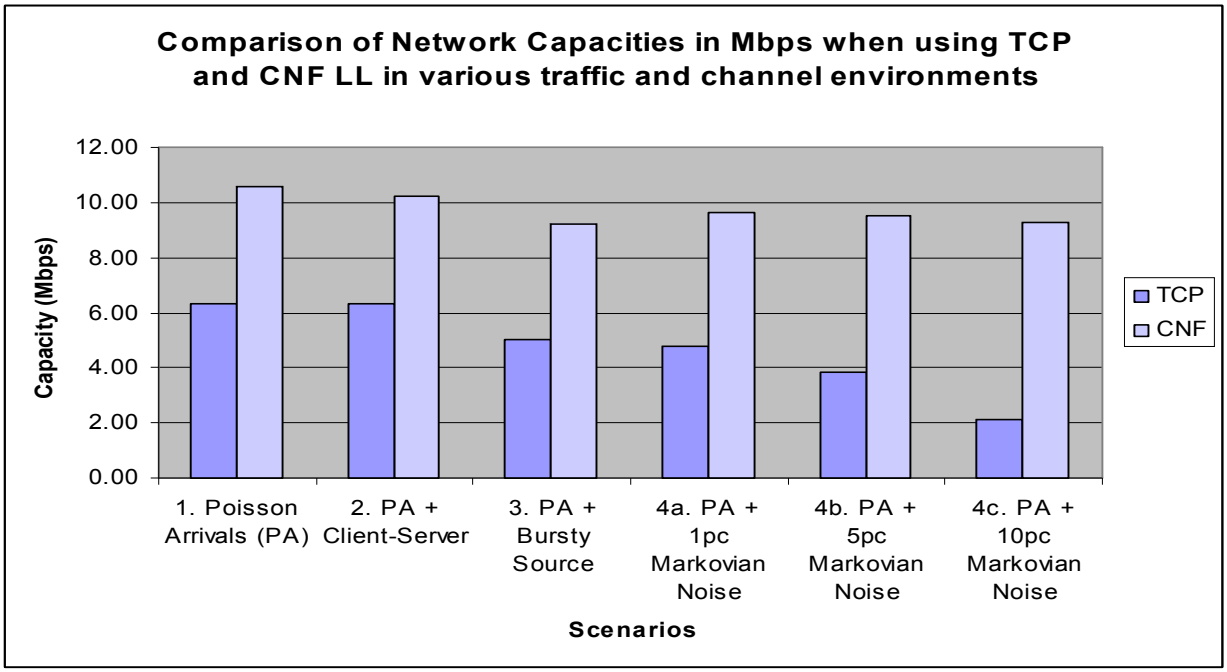

Figure 3-20 Network Capacities when using TCP and CNF LL

It can be seen from figure 3.20 that network capacity for both CNF and TCP is not significantly reduced with the introduction of client server model. However for the bursty source model the capacity gains of CNF

over TCP increase from 67% to 86%. With the introduction of 1% Markovian noise, the gains of CNF over TCP become 102%. As the noise duty cycle increases, TCP performance becomes worse. For 10% noise CNF achieves a gain of about 330% over TCP. Thus we can see that the 'killer' for TCP is correlated noise. CNF, however, faces some challenges due to congestion as can be seem from the bursty source case.

3.5.7 Memory Limitations:

Memory limitations were also introduced for the CNF LL case. A receiving node is required to allocate space for the file that it is receiving. If the node doesn't have enough space to accept any file it starts dropping file transfer requests. These requests could come from the application as well as the previous hops. It was found that for first-come-first-served basis, always-ON Poisson arrival traffic pattern, the capacity of the system is not materially affected if memory at each node is kept to be >200MB.

3.5.8 Simulation with Protocol Improvements:

Next we tried to implement some possible improvements, namely NACK prioritization and disabling of MAC acknowledgments for DATA packets, into the protocol. With only the NACK prioritization tested in the always-ON Poisson arrival scenario, the modification showed little gains. When the MAC acknowledgment skipping for DATA packets was introduced the gains became significant. This is because; when packets are lost MAC doubles the contention window size to gain access to the channel effectively doubling the channel access time. However, for the new packet MAC always starts with the minimum contention window size. Thus if the retransmissions are carried out by the Link Layer they are always treated as new packets by the MAC and hence channel access time remains the minimum for all DATA packets. Thus, the combination of the NACK prioritization and MAC acknowledgment skipping was again simulated in the afore-mentioned traffic and channel scenarios to see possible gains in the channel capacity. The results for various scenarios are shown in figure 3.21.

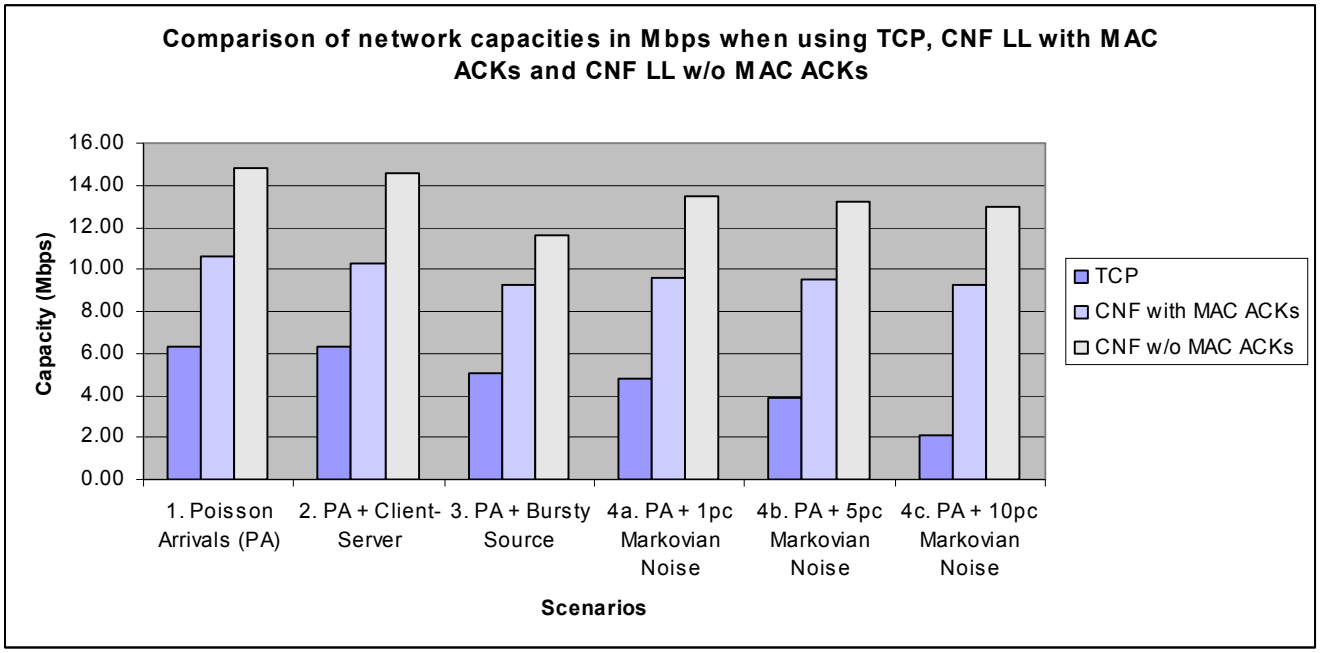

Figure 3-21 Comparison of Network capacities with CNF modification

As we can see, disabling the MAC level acknowledgments results in further gains for the CNF LL case. We cannot, however, disable MAC level acknowledgments for the TCP case as that would produce more problems for TCP as seen in the case of a single flow in linear topology. The behavior of CNF in different scenarios remains similar to the case with MAC level ACKs. It is somewhat noise-resilient but the worst case for CNF is the bursty source that, which leads to congestion. Overall all we can see that the gains of CNF over TCP are at about 130% for the noise free scenarios of Poisson Arrivals, Client-Server mode and bursty source model. However, with the introduction of noise the gains become 180% for 1% noise, 240% for 5% noise case and going up to 500% for 10% noise case.

4 INTRODUCING LINK LAYER SCHEDULING

In all the cases that we considered in the earlier chapter, the CNF sending link layer transmitted data files on a First-Come-First-Serve (FCFS) basis. However, in the cases where we introduced Markovian noise in to the channel, we can consider rescheduling of file at the link layer. To explain this concept let us consider that we have a node A that has two files at its sending link layer waiting to be transmitted. File 1 is at the head of the send queue and has a next hop B, and file 2 is next to the head and has a next hop C. If we transmit the files on a FCFS basis, file 1 will be transmitted first followed by file 2. But consider the scenario shown in figure 4.1, where at the current time, the channel from node A to node B is in bad state while the channel from node A to node C is in good state.

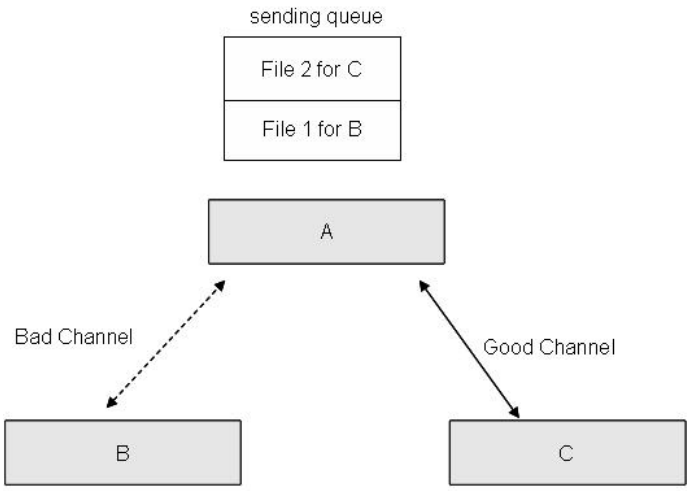

Figure 4-1 Opportunity for link layer scheduling

It would be better that if we first transmit file 2 to node C, where the link quality is good, and then transmit file 1 later when the channel improves or when there are no other files to be sent. In this way, not only we would be deferring the transmission to node B thus avoiding keeping the channel busy for no reason, we would also be making sure that we do not lose the good link to C while it is available. Furthermore, since we transmit the file in batches, it is not necessary to complete the transmission of one file before going to the other. Even if the channel becomes bad in the middle of the transmission, it is possible to switch to some other file that can be transferred over a better link.

4.1 Implementation in NS-2

Each node periodically broadcasts short packets so that other nodes can know of the link conditions. We have taken the channel to be symmetric. So the SNR at some node B when node A transmits is the same as SNR at node A when node B transmits.

The Link Layer maintains three queues now: the Receive queue, the Send queue and an Interim queue. The receive queue works in the same manner as explained before. When a node completes receiving a file it is handed over to the routing layer. The routing layer determines the next hop and passes it onto the LL. The Link layer then decides in which queue the file should be placed based on the next hop. If another file with the same next hop already exists in the send queue, the file is placed at the end of the interim queue. If there is not any file with the same next hop in the send queue then the file is moved to the end of the send queue. When a node finishes the transmission of a file it deletes the file from its send queue. It then looks in the interim queue whether any file with a new hop, than those it already has in its send queue, exists in the interim queue. If it does, the file is moved to the send of the send queue. The flow diagram for this process is shown in figure 4.2.

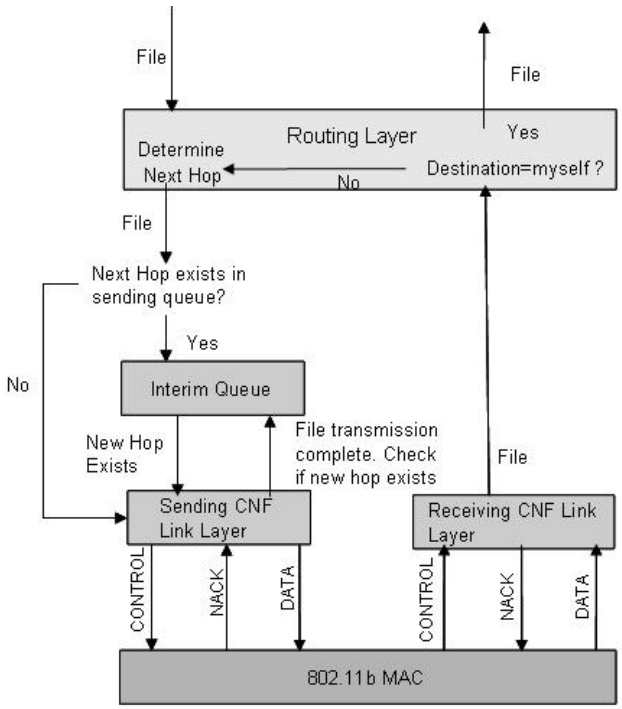

Figure 4-2 Flow diagram of CNF LL with scheduling

In this manner the send queue has files for different next hops and based on channel quality it can schedule transmissions between them. To start with, the sending link layer checks the channel quality with the next hops of each of the files that it has in its queue. It chooses the file that has currently the best channel quality with its next hop and starts transmission of one batch of packets. After the completion on one batch of packets, it again sorts the files as per the channel quality with their next hops and starts transmission of a batch of packets again.

4.2 Simulations with Link Scheduling (2-state Markov channel model)

We introduced two scheduling schemes for the 2-state Markov model.

1. Keep transmitting even if the best available link is in bad state (90% PER)
2. Halt transmission if the best available link is in bad state (90% PER)

We introduced this scheduling scheme in our 2-state Markov channel model introduced earlier to see what gains we can get.

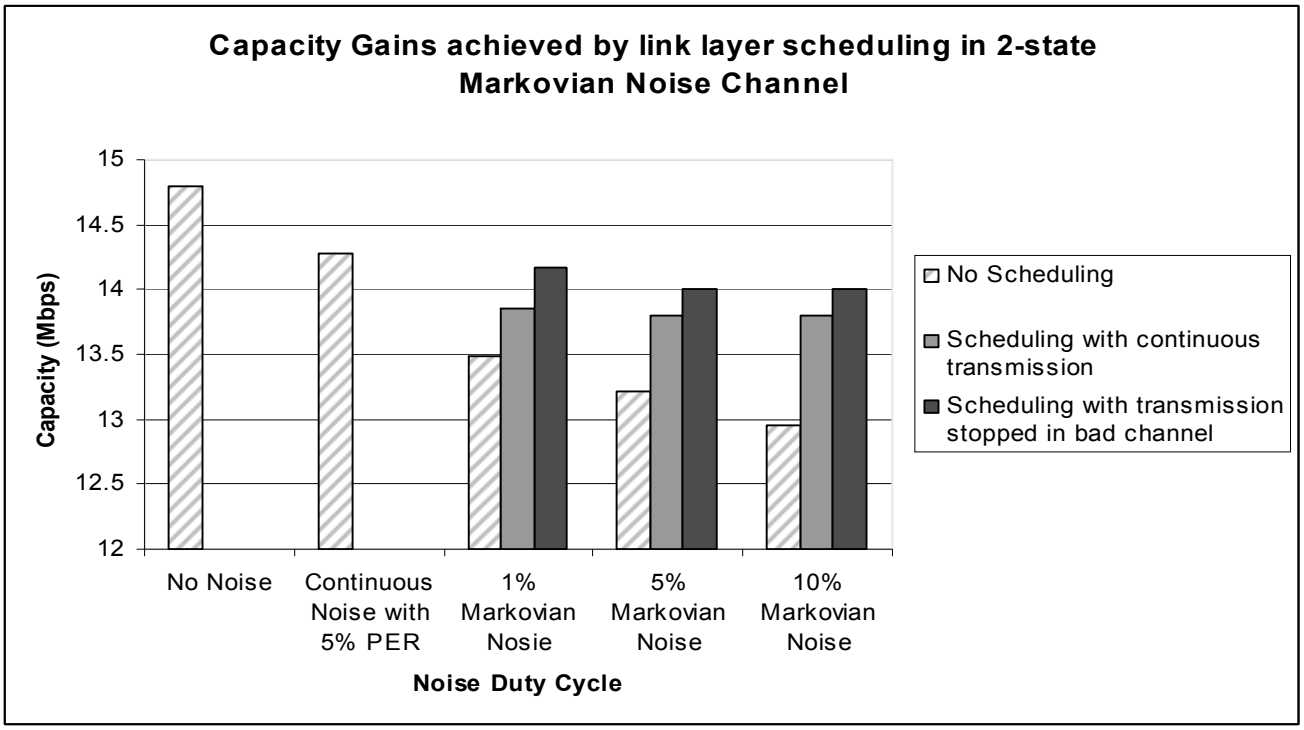

Figure 4-3 Capacity gains achieved by scheduling in2-state Markovian Noise Channel

It can be seen that more capacity is achieved if transmission is stopped during bad channel. For 1% noise the capacity improvement for scheduling-with-continuous-transmission over no scheduling is 2.7% going up to 4.4% for 5% noise and 6.5% for 10% noise. On the other hand, the capacity improvement for scheduling-with-transmission-stopped-in-bad-channel is 5% for 1% noise, 6% for 5% noise going up to 8% for 10% noise.

4.3 Introducing 8-state Markov Model:

The Markovian noise model that we introduced before was a 2-state Markov model where either the channel existed in good state with a 5% PER or bad state with a 90% PER. Further, the overall cycle time, with one epoch of good channel quality followed by one epoch of poor channel, spanned about 200 seconds on an average. We now try to improve the accuracy of the Markov modeling of the channel by introducing different SNR levels and making it more time-varying.

We refer to paper [21] in which authors collected actual 802.11 SNR traces in a variety of environments and tried to model it using a multi-state Markov chain. For our purposes, we take the intermediate environment model by the authors using an 8-state Markov chain.

Although the authors have considered transmission only at 11Mbps, we also consider that 802.11 PHY allows changing the transmission rate. The IEEE 802.11 standard allows changing the physical transmission rate in view of the observed channel quality. Since lower bit rate modulation schemes offer greater resilience to noise than higher bit rate modulation schemes, it might be better to switch to a lower transmission rate if the channel quality becomes worse. IEEE 802.11b supports rates of 1, 2, 5.5 and 11 Mbps. We did some calculation to find the SNR thresholds at which changing to a lower rate can lead to better throughput. It was found that for SNR>10dB it is transmitting at 11 Mbps lead to the highest throughput. Since according to [21] for an intermediate channel, the SNR remains greater than 10 dB, auto rate adaptation would not be a good option. So for intermediate channels we decided to keep the physical transmission rate at 11Mbps. The SNR threshold calculation is explained in Appendix A.

Going back to the discussion of the 8-state Markov model of the channel, each state comes with its own PER. The authors have provided an example average packet delivery ratio vector for the intermediate state, which is [0.0000 0.6785 0.7772 0.8535 0.9083 0.9453 0.9689 1.0000]. Using the following SNR vs. BER graph

given in [22] for 802.11b, we were able to reverse engineer an estimate of SNR level for each packet delivery rate.

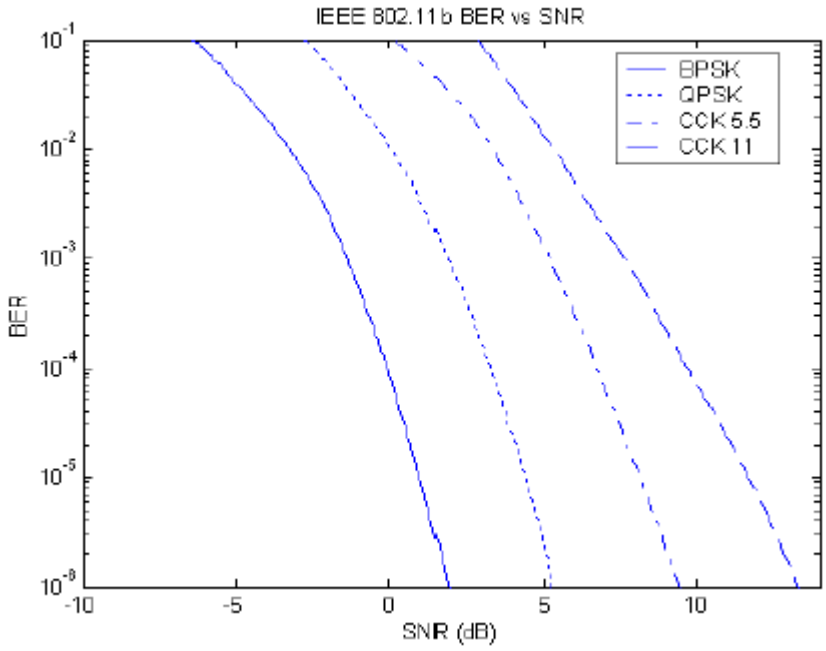

Figure 4-4 SNR vs. BER for 802.11b for various transmission rates [22]

Our DATA packets have a length of 1000 bytes. When all the headers are added, the length becomes 1100 bytes or 8800 bits. Using the above vector for packet delivery rate (PDR) we calculate the corresponding BER sing the following formula

PDR = (1- BER) ^8800

=> BER = 1 – (PDR) ^ (1/8800)

In this way we were able to get a mapping of SNR vs. PER for each of the 8 states. The mapping is shown below:

State	PER	SNR
0	1	<10
1	0.32	[10-10.88)
2	0.22	[10.88-11.48)
3	0.15	[11.48-11.88)
4	0.09	[11.88-12.14)
5	0.05	[12.14-12.3)
6	0.03	[12.3-13)
7	0	>13

Although the authors of [21] suggested the average memory of a good channel to be of the order of 20 packets where each packet was assumed to last 0.73ms, in order to be able to reflect channel variations typical of 802.11 environments [23][ref to the PER vs time figure showing ~100's of ms per mode], we considered channel memories which last up to an average of 5 seconds. In the model that we implemented in NS-2 the channel remains in each state for an exponential duration with mean 5 seconds, before moving on to the next or previous state. Given that the channel is leaving a state, it has equal probability of going into any adjacent state. Transitions to non-neighboring states are not allowed, which means that the channel does not behave in an absolutely random way.

4.4 Simulations with Link scheduling (8-state Markov model):

First we carried out simulations with the above channel model without introducing any link layer scheduling. Then to introduce link layer scheduling, we divided the link states into four levels to be informed to the link layer. The division of states into levels is given below:

Level	State	PER	SNR
0	0	1	<10
1	1	0.3215	[10-10.88)
	2	0.2228	[10.88-11.48)
2	3	0.1465	[11.48-11.88)
	4	0.0917	[11.88-12.14)
3	5	0.0547	[12.14-12.3)
	6	0.0311	[12.3-13)
	7	0	>13

The link layer does the scheduling based on these four levels. Then we assigned an individual level to each state and allowed the link layer to schedule file based on eight levels. The resulting network capacities with no scheduling, 4-level scheduling and 8 level scheduling are shown in figure 4.5

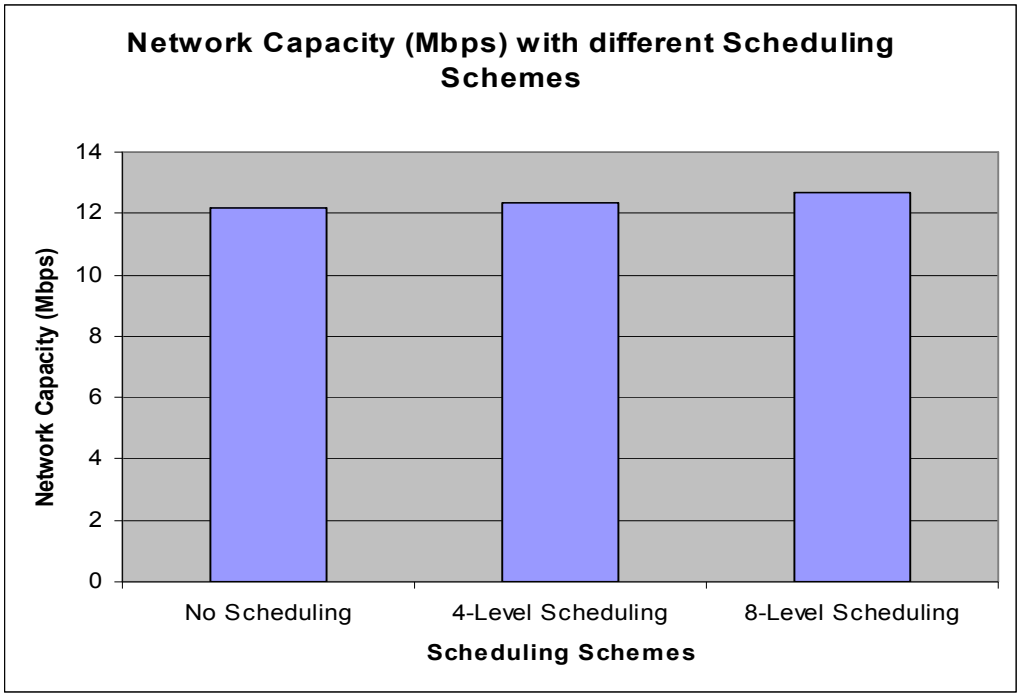

Figure 4-5 Network Capacity with different scheduling schemes

As can be seen the 4-level scheduling yielded 1.7% gains while 8-level scheduling yielded 4% gains.

5 CONCLUSIONS AND FUTURE WORK

5.1 Conclusions:

In this work, we proposed and evaluated the cache-and-forward (CNF) link layer protocol aimed at reliably transferring file on a hop-by-hop basis. We used UDP at the transport layer and static routing at the IP layer. It was observed via simulations that for every network load in a variety of different scenarios, our link layer protocol gives significant gains over TCP in terms of average file transfer delay and consequently network capacity. In the noise free scenarios with different traffic patterns, capacities provided by CNF were more than twice the network capacities provided by TCP. With the introduction of Markovian noise, the capacity gains reached up to 200-400% depending on the noise duty cycle. With the introduction of link scheduling, however, modest incremental improvements could be seen.

5.2 Future Work:

Link Scheduling can be given another dimension by having nodes co-ordinate their transmissions. The wireless channel is inherently a broadcast medium where all nodes contend for access. Simultaneous transmission cannot take place with in a certain geographical area. It might be possible to achieve some scheduling gains if node that has low SNR links with its neighbors halts its transmission for another node in its neighborhood that has higher SNR link(s) available to it.

We have used static shortest-path routing in the simulations. For dynamic routing, it remains to be seen how we can utilize the concept of in-network storage to define a routing metric. It might be appropriate to keep the metric as the minimum number of hops and travel along the path till the point where the link quality is good and then wait in storage till the time when the next link becomes available. It might also be efficient to take into consideration congestion at each node and take the path accordingly. Comparisons can be made to postal system when defining routing of files. Quality of service can be introduced by taking a faster route for some files that require urgent delivery and keeping in storage the files that do not have any time constraints.

6 References:

[1] S. Paul, R. Yates, D. Raychaudhuri and J. Kurose, “The cache-and-forward network architecture for efficient mobile content delivery services in the future Internet”, Innovations in NGN: Future Network and Services, First ITU-T Kaleidoscope Academic Conference, May 2008

[2] A. DeSimone, M.C. Chuah, and O.C. Yue, "Throughput performance of transport-layer protocol over wireless LANs," in Proceedings of IEEE GLOBECOM'93, Dec. 1993

[3] A. V. Bakre and B. R. Badrinath. “I-TCP: indirect TCP for mobile hosts”, Proceedings - International Conference on Distributed Computing Systems, 1995.

[4] H. Balakrishnan, S. Seshan, and R. H. Katz., “Improving reliable transport and handoff performance in cellular wireless networks”, ACM Wireless Networks, 1995.

[5] S. Gopal., “CLAP: A Cross Layer Aware transport Protocol”, PhD thesis, Rutgers University Dept. of Electrical and Computer Engineering, 2007.

[6] S. Gopal, S. Paul, and D. Raychaudhuri, “Investigation of the TCP simultaneous send problem in 802.11 wireless local area networks”, IEEE International Conference on Communications, 2005.

[7] B. R. Badrinath, A. V. Bakre, T. Imielinski, and R. Maramtz. “Handling mobile clients: A case for indirect interaction”, In Workshop on Workstation Operating Systems, 1993.

[8] H. Balakrishnan, V.N. Padmanabhan, S. Seshan and R.H. Katz, “A comparison of mechanisms for improving TCP performance over wireless links” in Proceedings of ACM SIGCOMM ’96, Stanford, CA, August 1996.

[9] S. Paul and E. Ayanoglu et al., ”An asymmetric protocol for digital cellular communications”, In INFOCOM , 1995.

[10] K. Fall. “A delay tolerant network architecture for challenged internets”, 2003.

[11] F. Warthman. “Delay-tolerant networks (DTNs): A tutorial”, www.ipnsig.org/reports/DTN Tutorial11.pdf, 2003.

[12] M. Li, D. Agrawal, D. Ganesan, A. Venkataramani, and H. Agrawal, “Block-switched Networks: A New Paradigm for Wireless Transport”, under submission, 2008

[13] http://standards.ieee.org/getieee802/802.11edownload/802.11e-2005.pdf : Quality of Service enhancements to 802.11.

[14] IEEE Computer Society LAN MAN Standards Committee. "Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications", New York, New York, 1997. IEEE Std. 802.11–1997.

[15] S. Zhao, Z. Wu, A. Acharya, and D. Raychaudhuri, "PARMA: A PHY/MAC Aware Routing Metric for Ad-Hoc Wireless Networks with Multi-Rate Radios", IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM 2005), June 2005

[16] www.isi.edu/nsnam/ns/

[17] http://www.winlab.rutgers.edu/%7Ezhibinwu/html/Routing_Agent.html

[18] http://www.isi.edu/nsnam/ns/doc/node218.html

[19] J. Lee et. al., "A Study of TCP Fairness in High-Speed Networks", 2007, URL: http://whitepapers.silicon.com/0,39024759,60300997p,00.htm?wp_user_rating=1

[20] J.P. Ebert and A. Willig, "A Gilbert-Elliot Bit Error Model and the Efficient Use in Packet Level Simulation", TKN Technical Report, Technical University Berlin, 1999, URL:www.tkn.tu-berlin.de/publications/papers/tkn_report02.pdf

[21] R.K. Guha and S. Sarkar, "Characterizing temporal SNR variation in 802.11 networks", Wireless Communications and Networking Conference (WCNC) , 2006.

[22] J.P. Pavon and S. Choi, "Link Adaptation Strategy for IEEE 802.11 WLAN via Received Signal Strength Measurement"

[23] I. Kozintsev and J. M. Veigh, "Improving Last-Hop Multicast Streaming Video over 802.11", 2004

7 Appendix A: SNR Threshold Calculation for SNR based auto rate adaptation in 802.11b

Using the following SNR vs. BER graph taken from [13] for each 802.11b transmission rate, we calculate the PER for each transmission rate corresponding to different values of SNR based on our packet size of 1000 bytes.

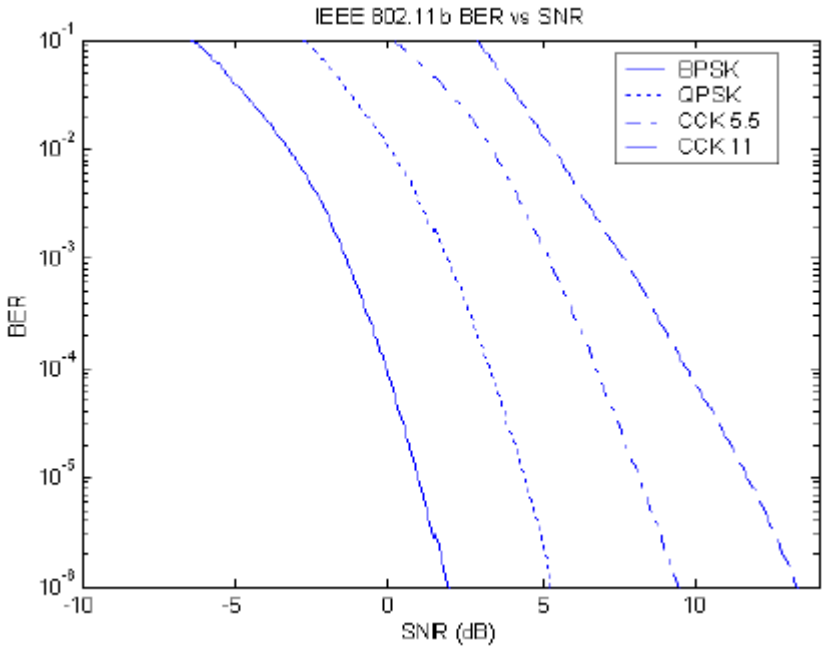

Figure 7-1 SNR vs. BER for 802.11b

Next, using PER and bit rate we calculate the expected transmission time of a packet for each transmission rate. In doing this calculation we also take into account the fact that we are disabling MAC level acknowledgments for CNF data packets. We get the following graph out of this calculation

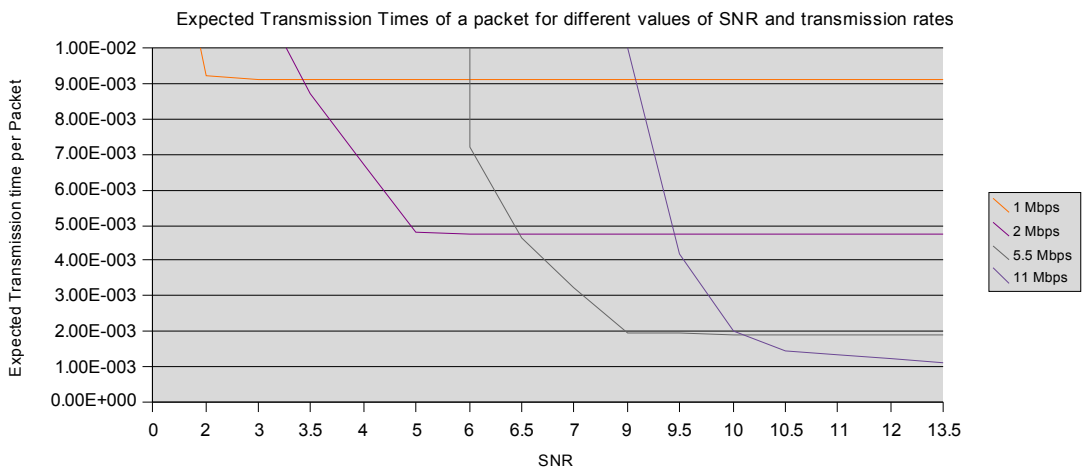

Figure 7-2 Expected transmission time per packet for various transmission rates in different SNRs

We can see that in different SNR regimes different transmission rates provide the minimum expected transmission time per packets and hence higher throughputs. Using this graph we can create SNR boundaries that can be used to allow MAC auto rate operation. Note the method used here to derive the SNR thresholds for the different transmission rates is based on the method used in [13] but we had to re-derive it since our packets size is different than that used in the paper and we are also turning off MAC level acknowledgments for the CNF DATA packets. Hence the operating SNR thresholds for different transmission rates based on the above graph would be

Rate (Mbps)	SNR
1	< 3.5
2	3.5 – 6.5
5.5	6.5 – 10
11	> 10