NOTE TO USERS

Analysis of TCP and UDP Performance over Static Wireless Multi-hop Ad-hoc LANs

Vahid Afrakhteh

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of Master of Applied Science (Electrical Engineering) at

Concordia University

Montreal, Quebec, Canada

July 2005

Your file Votre référence
ISBN: 0-494-10230-6
Our file Notre référence
ISBN: 0-494-10230-6

ABSTRACT

Analysis of TCP and UDP Performance over Static Wireless Multi-hop Ad-hoc LANs

Vahid Afrakhteh

Ad-hoc Wireless Local Area Networks (WLANs) are becoming very attractive and useful in many kinds of communication and networking applications. This is due to their efficiency, simplicity (in installation and use), relatively low cost, and availability.

Unfortunately, Internet application performance over wireless links is disappointing due to wireless impairments that adversely affect higher layers, specifically TCP and UDP which are the de facto standards for connection oriented and connectionless transport layer protocol, respectively.

This work analyzes how TCP and UDP (individually and together) perform in an IEEE 802.11 WLAN consisting of several non-mobile nodes.

The focus is on performance evaluation in terms of throughput, delay, jitter, and packet loss ratio at the transport layer. Based on the obtained results, suggestions and recommendations are made which can help improve the performance in such scenarios. These include packet size optimization, effect of RTS/CTS scheme, TCP versions, effect of loading the network with UDP traffic, and MAC buffer size optimization, among other things.

File transfer and Constant Bit Rate (CBR) applications are simulated, and the simulations are done using the Network Simulator Version 2 software.

ACKNOWLEDGEMENTS

I would like to express my great appreciation and thanks to my thesis supervisor Dr. A.K. Elhakeem. He always has new ideas in the field of telecommunications, specially regarding wireless networks, and it was him who gave me the right direction to start and to move along. His insight into the field, from different aspects, is very much appreciable. Here I also give my sincere gratitude to all the professors of Concordia University ECE department who instructed and formed me through my program.

I want to acknowledge the ECE department of Concordia University, and my supervisor Dr. Elhakeem for their financial support throughout the program.

Finally, I would like to give my words of thanks to the members of my thesis defense session, who put the effort to read this thesis and give me valuable comments and suggestions, and eventually evaluate my work.

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

ABR	Associativity Based Routing
ACK	Acknowledgment
AODV	Ad-hoc On-demand Distance Vector Routing
AP	Access Point
BRAN	Broadband Radio Access Networks
BSA	Basic Service Area
BSS	Basic Service Set
CBR	Constant Bit Rate
CFP	Contention Free Period
CGSRP	Clusterhead Gateway Switch Routing Protocol
CP	Contention Period
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
CSMA/CD	Carrier Sense Multiple Access with Collision Detection
DACK	Delayed ACK
DBPSK	Differential Binary Phase Shift Keying
DCF	Distributed Coordination Function
DQPSK	Differential Quadrature Binary Phase Shift Keying
DS	Distribution System
DSDV	Dynamic Destination-Sequenced Distance-Vector Routing Protocol
DSR	Dynamic Source Routing Protocol
DSSS	Direct Sequence Spread Spectrum
ESS	Extended Service Set
ETSI	European Telecommunications Standards Institute
FCC	Federal Communications Commission
FDDI	Fiber Distributed Data Interface
FHSS	Frequency Hopping Spread Spectrum
FTP	File Transfer Protocol
GFSK	Gaussian Frequency Shift Keying
HIPERLAN/2	High Performance European Radio LAN Type 2
IBSS	Independent BSS
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IP	Internet Protocol
IR	InfraRed
ISM	Industrial, Scientific, and Medical
LAN	Local Area Network
LLC	Logical Link Control
MAC	Medium Access Control
MANET	Mobile Ad-hoc Network
Mbps	Mega Bits per Second
MPDU	MAC PDU
MRL	Message Retransmission List
MSDU	MAC SDU
MSS	Maximum Segment Size

MTU	Maximum Transmission Unit
NIC	Network Interface Card
NS	Network Simulator
OFDM	Orthogonal Frequency Division Multiplexing
OS	Operating System
OSI	Open System Interconnect
OTcl	Object Tcl
PC	Point Coordinator
PCF	Point Coordination Function
PDU	Protocol Data Unit
PPM	Pulse Position Modulation
RFC	Request for Comment
RTS/CTS	Request to Send / Clear to Send
SACK	Selective ACK
SDU	Service Data Unit
SSR	Signal Stability Routing
TCL	Tool Command Language
TCP	Transmission Control Protocol
TORA	Temporally Ordered Routing Algorithm
UDP	User Datagram Protocol
UNII	Unlicensed National Information Infrastructure
Wi-Fi	Wireless Fidelity
WLAN	Wireless Local Area Network
WRP	Wireless Routing Protocol

LIST OF FIGURES

Chapter 1 Introduction

1.1 Wireless LANs

Wireless Local Area Network (WLAN) is a very attractive technology in wide and ever-more-popular use today, which is quickly replacing its counterpart wired solution. The reason for this fast adoption is the many advantages it offers, compared to a wired network, all because of one fact: there's no need for physical connectivity between devices in the network. This fact brings about all the advantages of a wireless network, including: the freedom of user mobility, the ease of installation because of not having to cable the whole area, and finally the low cost of setting up such a network.

There are many applications to this kind of networks, many appearing in our day to day life. Some examples include wireless networks and Internet access in an office, corporate building, or a university/school campus, wireless networks present in large department stores which are used for instantaneous inventory management, wireless sensors scattered around a factory or scientific laboratory, and the like.

Newer applications of these networks are transferring high data rate voice and video over radio links, which demands very high capacity links.

The most popular and widely used standard for WLAN is the IEEE 802.11 [2] WLAN standard. (Others include the European Telecommunications Standards Institute Broadband Radio Access Networks (ETSI BRAN) High Performance European Radio LAN (HIPERLAN/2) [6], which supports up to 54 Mbps in the 5 GHz band, Bluetooth [7], which supports up to 2 Mbps in 2.45 GHz band, and finally HomeRF with 10 Mbps in the 2.4 GHz band, which is no longer supported by vendors or working groups.)

There are several versions of the IEEE 802.11 wireless standard defined and in use today. Among them, the most widely used and accepted is 802.11b [3] (which is commonly referred to as Wi-Fi, for *Wireless Fidelity*, in the commercial market). It is seeing a remarkable success in the market mainly because of the ever-shrinking cost and ever-shrinking hardware size, simplicity and ease of use/setup, high data rates (up to 11 Mbps), and support from big manufacturers [48].

1.2 Flavors of the IEEE 802.11 Standard

The IEEE 802.11 standard is comprised of several WLAN standards developed by Working Group 11 of the IEEE LAN/MAN Standards Committee (IEEE 802) [16]. The original standard was 802.11 released in 1997 and now is called "*802.11 legacy*". Later, other standards were added to it, including some on modulation techniques (a, b, g), on security (i), on service enhancement, extensions, management, and integrity with other kinds of networks (c-f, h-j, n).

The three most popular additions are the *a*, *b*, and *g* amendments, each defining an operating frequency band and an over-the-air modulation technique.

Following, there is a brief description of each of these three standards taken from [17].

1.2.1 802.11 Legacy

The original version of the IEEE 802.11 standard was released in 1997 and specifies two raw (channel) data rates of 1 and 2 Mbps. Also, two transmission techniques were defined: *InfraRed* (IR) transmission, and radio transmission in the *Industrial Scientific Medical* (ISM) frequency band at 2.4 GHz. The IR technique has no actual implementations as of yet.

The medium access method defined in this standard is *Carrier Sense Multiple Access with Collision Avoidance* (CSMA/CA), which is being used by all subsequent versions of the standard as well. It uses a significant amount of the raw channel bandwidth to increase reliability of data transmission in adverse wireless environments.

A few commercial products appeared on the market using this standard, including Alvarion (PRO.11 and BreezeAccess-II), Netwave Technologies (AirSurfer Plus and AirSurfer Pro) and Proxim (OpenAir). But the problem with this first release was that it left too many things to the implementer's choice, and as a result some compatibility and inter-operability issues were arisen.

Legacy 802.11 was rapidly supplemented and popularized by 802.11b.

1.2.2 802.11b

The 802.11b standard was released in 1999. Its raw channel data rate is 11 Mbps, but the throughput an application can achieve using this standard is only about 5.9 Mbps over TCP and 7.1 Mbps for UDP (because of MAC protocol overhead).

It operates in the 2.4 GHz ISM band, and since this band is unlicensed (unregulated), 802.11b equipment can incur interference from microwave ovens, cordless phones, and other appliances using the same 2.4 GHz band.

The physical layer of this standard supports 1, 2, 5.5 and 11 Mbps, all four of which are mandatory. 802.11b cards operate at 11 Mbps by default, but will scale back to 5.5, then 2, then 1 Mbps, if the signal quality drops because of interference or noise. These lower data rates use more redundant algorithms for modulation and coding, resulting in less susceptibility to signal attenuation.

The huge speed increase of this standard compared to the original 802.11, and significant price reductions in a very short time period, has lead the 802.11b standard being accepted as the mainstream wireless LAN technology of choice.

The network setup for 802.11b is usually a point-to-multipoint communication, where a central access point (similar to base station in cellular networks) with an omni-directional antenna can communicate with mobile nodes scattered around its coverage range. It is also possible to use high-gain directional antennae to establish point-to-point configurations, thus allowing more distance between the access point and the client node, sometimes up to tens of kilometers.

The first commercial product using the 802.11b standard was by the Apple Computer under the trademark *AirPort*. Now LinkSys is the market leader.

1.2.3 802.11a

The 802.11a amendment was ratified in the same year as the b, 1999. This standard operates in the 5 GHz band, uses *Orthogonal Frequency Division Multiplexing* (OFDM) as the modulation technique, and it's maximum raw channel data rate is 54 Mbps which will result in application throughputs of about 20 Mbps. According to the signal condition, the data rate is reduced to 48, 36, 34, 18, 12, 9 then 6 Mbps if required.

The total bandwidth is 20 MHz, with 16.6 MHz of occupied bandwidth.

This standard is not inter-operable with the 802.11b (because of different frequency bands), except when using devices that support both standards.

Using the 5 GHz frequency band (as opposed to the occupied and heavily used 2.4 GHz band), gives 802.11a the advantage of less interference, but at the same time this high frequency results in some disadvantages. Penetration through walls is close to impossible

at this frequency, thus it restricts the use of 802.11a mostly to line of sight scenarios. This implies the need for more access points in the same surface area than 802.11b, and also less coverage range (assuming equal power).

The manufacturing of 802.11a products was somehow delayed and lagged behind 802.11b, mostly because of the frequency regulatory issues. Still it is not as widely adopted as 802.11b (or the new standard 802.11g), because at the time of its appearance on the market 802.11b was already widely adopted and used, and because of the poor initial products with short ranges.

But recently 802.11a products have almost the same range characteristics as 802.11b. Also, currently there are dual-band, dual-mode and even tri-mode devices that support two or three of the standards (a, b, g) at the same time.

1.2.4 802.11g

The most recent released version of this set of standards is the IEEE 802.11g standard. This flavor is very attractive, and is predicted to be the next popular wireless LAN technology after 802.11b, soon replacing it completely. The main reason for this is that it has the advantages of both of the previous standards: It has a data rate of 54 Mbps (net throughput of about 25 Mbps), like 802.11a, but it operates in the 2.4 GHz frequency band, like 802.11b, so it is fully (backwards) compatible with the b products.

The consumer market availability of 802.11g products was very rapid (early 2003), even before the final version of the standard was released (summer 2003).

Despite all this, there are still a few compromises in 802.11g networks/products: conflicts with 802.11b devices which causes a significant reduction in the speed of an 802.11g network, susceptibility to the same interference/noise sources as 802.11b (because of the

ISM band), having only 3 non-overlapping frequency channels, and finally being a lot more susceptible to noise and interference compared to 802.11b because of its higher data rates. This sometimes causes g devices to reduce their speed to effectively the same rates as b devices.

The best possible solution which would make optimum use of all the three networks is the use of dual-band/tri-mode devices, so that the maximum possible throughput is achieved.

Again Apple was the first major company to use 802.11g, under the trademark *AirPort Extreme*. Then was Cisco (who bought LinkSys) and produced its mobile adaptor with the name *Aironet*.

1.3 Topologies (modes) of 802.11 WLAN

There are two modes of operation for 802.11 WLANs: Ad hoc and Infrastructure [8].

In both modes, a fundamental building block, *Basic Service Set* (BSS), is defined. The geographical area covered by a BSS is called *Basic Service Area* (BSA). BSA is the analogous to a cell in cellular mobile networks. All the devices (nodes, stations) situated in a BSS are controlled by a single MAC layer function (i.e. DCF or PCF, defined in Sec. 1.5.1 and 1.5.2). Below is a brief description of each mode [8].

1.3.1 Ad Hoc Networks

The ad hoc network is called *Independent BSS* (IBSS) in 802.11 standard terms. It is a group of stations all in a single BSS, where each station can communicate directly with all the others. There is no need for any kind of a base station, central station, or infrastructure. Fig. 1.1 illustrates a 3-node ad hoc network (IBSS).

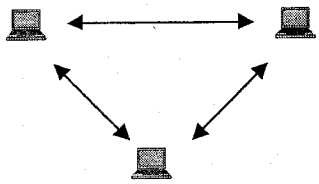


Figure 1.1 - IBSS

1.3.2 Infrastructure Networks

In the infrastructure mode, *Access Point* (AP) is used to establish connection between stations. All traffic passes through the AP. The AP has the similar role as the Base Station (BS) in a cellular network. There is one AP associated with a BSS.

An *Extended Service Set* (ESS) is a collection of multiple BSS/AP's, and has the appearance of one large BSS to the Logical Link Control (LLC) sublayer of each station. The APs in an ESS are linked together via a common Distribution System (DS). The DS could be any kind of transport network (e.g. a wired IEEE 802.3 Ethernet LAN, IEEE 802.4 token bus LAN, IEEE 802.5 token ring LAN, fiber distributed data interface (FDDI) metropolitan area network (MAN), or another IEEE 802.11 wireless medium), so it is implementation-independent. The role of the DS is simply to act as a transport network for carrying packets between different BSSs in an ESS.

Fig. 1.2 illustrates a simple ESS developed with two BSSs and a DS.

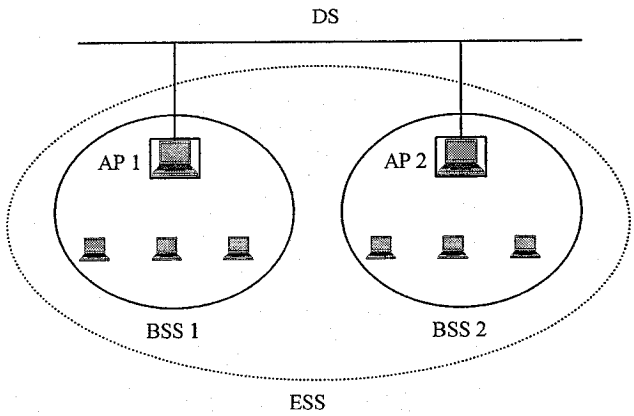


Figure 1.2 - ESS

1.3.3 Advantages and Disadvantages of Ad Hoc Networks

As noted in [47], comparing an ad hoc network with its infrastructure counterpart, each has its own advantages and disadvantages:

- **Cost savings**: Because of no need for purchase and installation of access points, significant cost savings can be made when using ad hoc networks.
- **Time savings**: Again because of no need for any kind of infrastructure installation, the setup time of an ad hoc network is much less than that of an equal-sized infrastructure network. The only necessary step (if not already done) is installing NICs (Network Interface Card) in the user devices.
- **Performance**: If the number of users in the network is relatively small, then the performance of the ad hoc network is much better than the infrastructure mode, because the traffic doesn't have to pass through an access point. But when the group size is larger than a certain threshold, the performance can be boosted using access points and having an infrastructure. In this case, users can be split into

groups (BSSs) each having its own AP and operating frequency, thus taking benefit of spatial frequency reuse.

- **Network access**: In case only communication between wireless stations is needed, ad hoc is a perfect solution. But if access to external networks (such as the Internet) is required, then having a DS in place which is wired to external networks will be of great help, and will increase efficiency of access to those networks.
- **Network management**: Because there's no centralized device present in an ad hoc network, and the network topology changes frequently, managing such a network is a difficult task. Whereas when having and infrastructure in place with APs, one can easily monitor and effectively manage the network through that central station. Here again, the ad hoc network is not suitable for large enterprise WLANs.

1.4 802.11 Physical Layer

As explained in [8], there are three different physical layer implementations defined in the original IEEE 802.11 standard: *Frequency Hopping Spread Spectrum* (FHSS), *Direct Sequence Spread Spectrum* (DSSS), and *InfraRed* (IR).

The FHSS operates in the 2.4 GHz Industrial, Scientific, and Medical (ISM) band (2.4000-2.4835 GHz). Three different hopping sequence sets each with 26 hopping sequences are present, thus enabling multiple BSSs to coexist in the same geographical area. The minimum hop rate permitted is 2.5 hops/s. *Gaussian Frequency Shift Keying* (GFSK) is used as the modulation technique.

The ISM band is also used for DSSS, where the modulation is done with DBPSK for 1 Mbps rate and DQPSK for 2 Mbps rate. Only two overlapping or adjacent BSSs can operate without interference.

The IR is for indoor use only, and uses a wavelength range from 850 to 950 nm. The modulation technique used is *Pulse Position Modulation* (PPM). As mentioned earlier, there are no commercial implementations of the IR technique.

1.5 802.11 Medium Access Control (MAC) Layer

The MAC sub-layer of the Data Link Layer has channel allocation, *Protocol Data Unit* (PDU) addressing, frame formatting, error checking, and fragmentation and reassembly as part of its tasks [8]. There are two modes of operation in the transmission medium: *Contention Period* (CP), and *Contention-Free Period* (CFP). During the CP, all stations must contend for the channel before transmitting each MAC PDU (MPDU, frame), but during the CFP it's the job of the AP to control access to the shared medium, and this is done by polling the stations according to a table. Thus in the CFP, the AP is the mediator, and stations don't need to contend for the channel any more, as long as they are given access to use the channel by the AP.

Three different frame types are defined by the standard: management, control, and data. The management frames are used for station association and disassociation with the AP, timing and synchronization, and authentication and deauthentication. Control frames are used for handshaking, for positive acknowledgments, and to end the CFP. Data frames are used for the transmission of data.

There are two functions which control medium access during CP and CFP, namely DCF and PCF, which are defined below [8].

1.5.1 Distributed Coordination Function (DCF)

The DCF is the fundamental access method for data transmission on a best effort basis. All stations should support DCF. It is the only function operating in the ad hoc mode, and it is present in the infrastructure mode either by itself or at the same time with PCF. As shown in Fig. 1.3, the DCF sits directly on top of the physical layer and supports contention services. As mentioned earlier, contention means that each station who has a MAC *Service Data Unit* (MSDU) to transmit, should queue the MSDU (frame) in its transmission queue, then contend for the channel according to DCF algorithms. After successful transmission of each frame, if there are more frames to be transmitted, the same procedure should be repeated.

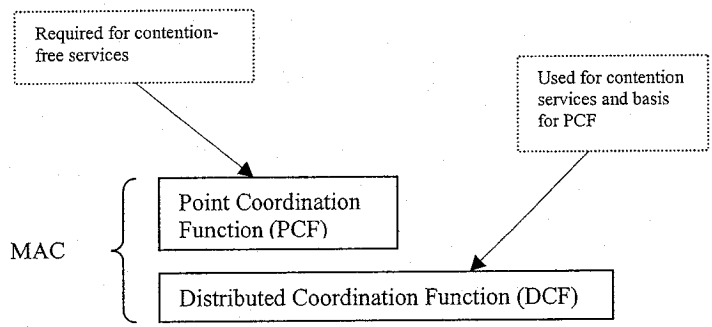


Figure 1.3 - MAC Architecture

The contention algorithm is based on a random back-off timer: when a station wants to send a frame, it senses the medium, if it's free, it transmits the packet, if it's busy, then the station (and all other stations contending at the same time), backs off for a random amount of time, and then tries to send its frame again. This procedure repeats itself until the station is able to send its packet, or after a certain number of unsuccessful retransmissions (usually 7), the packet is dropped from transmission queue.

DCF is based on the CSMA/CA scheme. Carrier sensing is performed at both the air interface, referred to as *physical currier sensing,* and at the MAC sub-layer, referred to as *virtual carrier sensing.* For more detailed explanation of the DCF, please refer to [8].

1.5.2 Point Coordination Function (PCF)

Implementing PCD is optional for 802.11 stations. It is connection-oriented, and operates in the CF transfer mode. Stations within the BSS that are capable of operating in the CFP are known as *CF-aware* stations. It is the AP who performs the PCF functionality and polls the stations to transmit without contending for the channel. It is left open to the implementer to determine the polling sequence and how to create and maintain polling tables. The PCF is required to coexist with the DCF and logically sits on top of the DCF (Fig. 1.3). For more detailed explanation of the DCF, please refer to [8].

1.5.3 RTS/CTS Scheme

The IEEE 802.11 MAC specification defines implementing an optional scheme called Request-to-Send/Clear-to-Send (RTS/CTS) [8]. This function allows a level of control over the shared wireless medium.

In most NICs/devices, users can set a maximum frame length threshold where the NIC will activate the use of RTS/CTS. For example, a frame length of 1,000 bytes will trigger RTS/CTS for all frames larger than 1,000 bytes. The use of RTS/CTS alleviates (but doesn't completely remove) the *hidden terminal* problem [36], that is, where two or more stations can't hear each other and they both want to transmit to a third station.

Stations can either choose not to use RTS/CTS at all, use it when the frame size exceeds the value of RTS-Threshold (manageable parameter), or always use RTS/CTS.

The advantage of the scheme is that if a collision occurs with an RTS or CTS MPDU, far less bandwidth is wasted than when collision happens with a large data MPDU. The disadvantage is that it introduces lots of overhead and delays into the system.

If RTS/CTS is activated, the source node will first send an RTS frame before sending a data frame. The destination node will then respond with a CTS frame, indicating that the source can send the data frame. With the CTS frame, the node will also provide a value in the duration field of the frame header that holds off other stations from transmitting until after the source initiating the RTS has finished sending its frame. This avoids collisions between hidden nodes. The RTS/CTS handshake continues for each frame, as long as the frame size exceeds the threshold set in the corresponding radio NIC.

As a simple example, suppose there are three 802.11 stations (Station A, B, C). Station A and Station B can't hear each other because of high attenuation (e.g. substantial range), but they can both communicate with Station C (Fig. 1.4). Because of this situation, Station A may begin sending a frame without noticing that Station B is currently transmitting (or vice versa). This will very likely cause a collision between packets of Station A and Station B to occur at the location of Station C. As a result, both Station A and Station B would need to retransmit their packets, which results in wasted bandwidth.

If either Station A or Station B activates RTS/CTS, however, the collision will not happen because the stations will be aware of each other's activities. Thus, the use of RTS/CTS reduces collisions and increases the performance of the network if hidden stations are present.

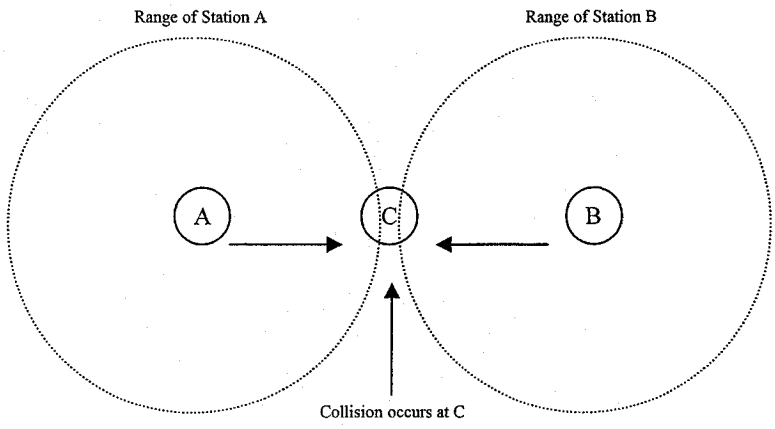


Figure 1.4 - Hidden Terminal Problem

It should be mentioned, though, that an increase in performance using RTS/CTS is the net result of introducing overhead (i.e. RTS/CTS frames) and reducing overhead (i.e. fewer retransmissions). If there are not any hidden nodes, then the use of RTS/CTS will only increase the amount of overhead, which reduces throughput. A slight hidden node problem may also result in performance degradation if RTS/CTS is implemented. In this case, the additional RTS/CTS frames cost more in terms of overhead than the gain achieved by reducing retransmissions. Thus, one should be careful when implementing RTS/CTS [47].

1.6 Ad hoc Routing Protocols

In infrastructure networks, mobile networks communicate directly with an Access Point. This AP acts as a bridge for the network, and routes the packets coming from the mobile nodes in its coverage range, to their proper destination using existing routing protocols. When a node moves out of the range of an access point, it associates itself with a new AP whose signal it receives, and starts to communicate with this new AP. This process is

called handoff and is relatively simple. So, in addition to the mobile hosts, there are always a number of (at least one) fixed nodes, present in the network.

But in an ad hoc network, all nodes are mobile and each node can communicate with any other node in a dynamic and arbitrary manner. Thus the topology of the network is very fluid, and changes frequently. In such a scenario, each node acts as a router, and in addition to receiving packets destined for itself, participates in forwarding packets belonging to other nodes and connections in the network.

Hence an appropriate routing protocol, quite different from routing protocols designed for wired networks with fixed hosts and relatively fixed topology, is required for an ad hoc network. There are several ad hoc routing protocols designed and in use today, which can generally be divided into two categories (Fig. 1.5): table-driven (also called proactive) and on-demand (also called reactive) routing, based on when and how the routes are discovered [9].

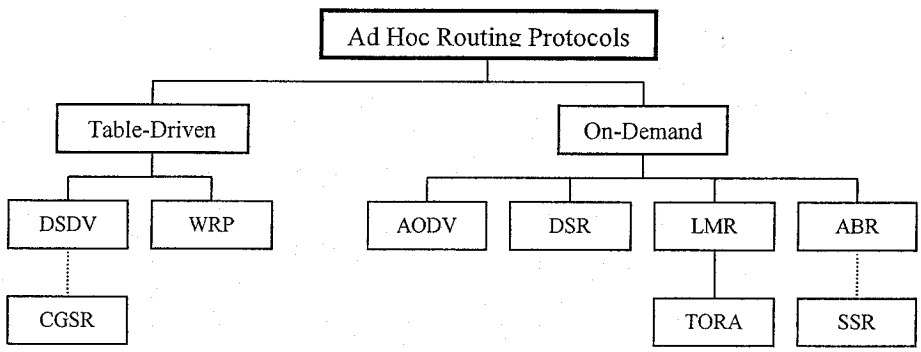


Figure 1.5 - Categorization of ad hoc routing protocols

1.6.1 Table-Driven Routing Protocols

The routing protocols in this category, tables are maintained by each node that contain routing information to all other nodes in the network. These routing tables are always

updated, so that at any given time each node in the network has a consistent and up-to-date view of the network topology.

In order to keep the routing information up-to-date, whenever a topology change happens in the network (a node moves out of the range of another node, a route breaks, a node is turned off, ...), update messages propagate through the whole network carrying information about this change.

The protocols in this category differ from each other mainly in their method of distribution of topology change and route update messages, and also the number and format of the necessary routing-related tables.

Three of the most popular protocols in this category are:

- Destination-Sequenced Distance-Vector (DSDV) [10]
- Hierarchical State Routing (HSR) [11]
- Wireless Routing Protocol (WRP) [12]

For more information on table-driven routing algorithms, please refer to [9].

1.6.2 On-Demand Routing Protocols

On-demand ad hoc routing protocols are a source-initiated type of routing protocol. They take a more relaxed approach to routing: the routes are created only when required. As a result, all up-to-date routes are not maintained at every node.

When a node has data to send to a destination, it initiates a route discovery procedure to find the path to the destination. This request propagates through the network passing the intermediate nodes one by one, and recording the route taken in the route request message itself, until either the destination is reached, or an intermediate node who knows the route to the destination is reached.

Some of the most popular protocols in this category include:

- Ad hoc On-demand Distance Vector (AODV) [13]
- Dynamic Source Routing (DSR) [14]
- Temporally Ordered Routing Algorithm (TORA) [15]

For more information on on-demand routing protocols, please refer to [9].

1.7 Thesis Organization

The remainder of this thesis is organized as follows:

In Chapter 2, along with a brief introduction to TCP mechanisms, the issues of TCP over wireless links are discussed. Furthermore, a brief introduction to UDP is given. Then previous work on the subject is presented, and the approach and objectives of this thesis are given.

Chapter 3 explains the simulation software used, and the setup and procedure of the experiments done with it.

Chapter 4, which would be the essential part of this thesis, will present the experiments done, along with the motivation of each one, the results, the interpretation, and also any possible suggestions for improvement.

And finally in Chapter 5, the summary and conclusions of this work, along with suggestions for future work is presented.

Chapter 2 TCP over Wireless Links

This chapter is an overview of the TCP protocol, the problems and issues of TCP over wireless networks, the current and previous research done in this regard, and finally the scope and approach of my thesis on this issue.

2.1 TCP Mechanisms in Brief

TCP is a connection oriented transport (Layer 4, Fig. 2.1) protocol that provides reliable end-to-end point-to-point communication on top of a datagram (like IP, Internet Protocol) network. The TCP sender divides the data into *segments* that are transmitted and upon receipt, acknowledged by the receiver. Different mechanisms and algorithms are defined in the TCP standard, which make it a very reliable transmission protocol and avoid overflowing the receiver or the underlying network. Some of these mechanisms (including slow start, congestion avoidance, fast retransmit, and fast recovery) are briefly described below [49, 50].

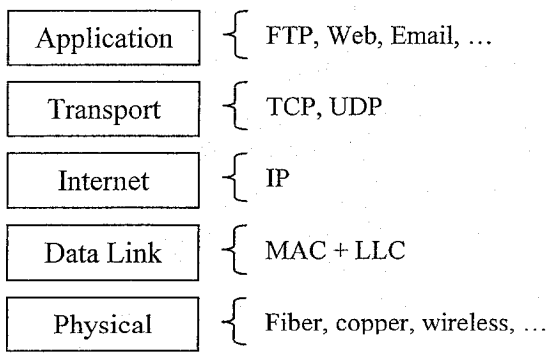


Figure 2.1 - TCP/IP network architecture

2.1.1 Acknowledgments

In TCP, the reliability of the protocol (meaning delivering in-order, error-free, and loss-free packets) is assured by the use of an acknowledgement scheme, in which the reception of every data segment is acknowledged by the receiver. Each acknowledgement actually confirms the reception of the last in-sequence received data segment. Duplicated ACKs are generated whenever an out of sequence segment is received.

If *Delayed ACK* procedure is used, the transmission of the acknowledgement information is delayed for 200 ms or till two data segments are received. This reduces (to half) the number of generated ACK packets.

2.1.2 Congestion Control

Using the feedback which is provided by the acknowledgments received, the TCP sender can probe the network capacity and accordingly, limit the amount of data that is under transmission at any given time. This is the essence of congestion control, and is implemented by the following two mechanisms:

- *Slow Start:*

The idea behind the slow start mechanism is that new packets should be injected into the network at the same rate as the acknowledgments are received from the other end.

Hence, in addition to the *advertised window* (AWND) which is used to prevent overflowing the receiver, a new window is defined: *congestion window* (CWND). At the beginning of the connection, the CWND is initialized to one MSS (Maximum Segment Size). With the receipt of each ACK, the size of congestion window is increased by one segment. So when the ACK for the first segment is received, the congestion window is increased to two segments, and now two segments can be sent. After each of these

segments has been acknowledged, congestion window will increase to four segments, and so on. This provides for an *exponential growth* of the window size.

This grow continues until either a threshold called *slow start threshold* (SSTHRESH) is reached (after which the congestion avoidance phase begins), or the capacity of the network is reached and packets start to get dropped in intermediate routers (in which case slow start or fast recovery phase begins).

- *Congestion Avoidance:*

Congestion occurs when data streams from a high bandwidth link arrive at the entry of a lower bandwidth link, or when multiple data streams arrive at the entry of a router whose output bandwidth is less than the sum of all the input bandwidths. This causes newly arrived packets to the entry point to be dropped. Congestion avoidance is the method TCP uses to deal with packet loss.

TCP becomes aware of network congestion by assuming that all packet losses are caused by congestion somewhere in the network, and not by damaged or corrupt packets due to errors in transmission medium (though this assumption is not true for wireless networks).

As mentioned earlier, assuming there is no network congestion, slow start continues until a threshold is reached (SSTHRESH, usually 64 KB), after which point TCP enters a new phase called congestion avoidance.

In this phase, CWND increases by 1/CWND for each ACK received, or in other words, it increases one segment for each set of CWND AKCs received. This is a *linear growth* of CWND, compared to slow start's exponential growth.

Now if there is network congestion occurring at some point, TCP reacts in one of the following two ways, depending on what has indicated a packet loss:

- A duplicate ACK is received, which shows that packets are still being received by the destination host, and the level of congestion is low. In this case a procedure called *fast retransmit* is followed. (explained in the next section)
- The retransmission timer for a missing segment is expired, which means that the congestion is so high that no segments are received by the receiver any more. In this case TCP lowers its CWND to 1 MSS and enters the slow start phase again.

In both the above cases, the SSTHRESH is reduced to half of its current size.

It should be noted that the actual value of the TCP window size, is always the minimum of the AWND (advertised window) and the CWND (congestion window).

In summary:

- When the congestion window is below the threshold, the congestion window grows exponentially.
- When the congestion window is above the threshold, the congestion window grows linearly.
- Whenever there is a timeout, the threshold is set to one half of the current congestion window and the congestion window is then set to one.

The evolution of TCP's congestion window is shown in Fig. 2.2. In this figure, the threshold is initially equal to 8*MSS. The congestion window increases exponentially during slow start and reaches the threshold at the fourth transmission. The congestion window then climbs linearly until loss occurs, just after transmission 8. Note that the congestion window is 12*MSS when loss occurs. The threshold is then set to 0.5**CongWin* = 6*MSS and the congestion window is set to 1. And the process continues.

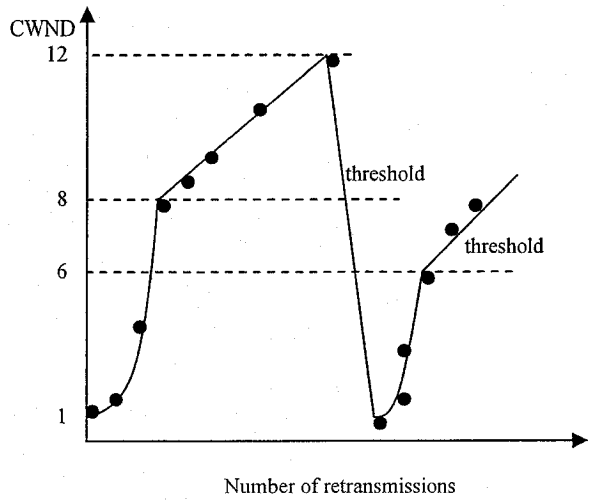


Figure 2.2 – A Typical Evolution of TCP's congestion window

2.1.3 Retransmissions

Recovery from data loss is done in TCP using two ways:

First, is when the retransmission timer for a segment expires, which means that either the segment is lost and not received by the destination host, or its ACK has been lost. TCP then retransmits the segment, and also interprets this event as a congestion signal. So the congestion window is reset to the initial 1 MSS value, and slow start phase begins again.

Second, is when 3 duplicate ACKs are received for a segment, where the Fast Retransmit/Fast recovery algorithm is incorporated:

- *Fast Retransmit:*

When an out of order segment is received, the destination TCP immediately generates a duplicate acknowledgement. This acknowledgement shows the other end that an out of order segment has been received, and the sequence number the receiver is waiting for.

Now since the source TCP does not know whether a duplicate ACK is caused by a lost segment or by a delay/reordering of segments, it waits until three duplicate ACKs are

received. This is because it is generally assumed that if there is just a reordering of the segments, there will be only one or two duplicate ACKs before the reordered segment is processed, after which a new ACK will be generated. But if three or more duplicate ACKs are received in a row, it is a good indication that a segment has been lost. In this case, TCP immediately retransmits the missing segment without further waiting for the retransmission timer to expire, thus saving time and bandwidth. This procedure is called Fast Retransmit.

- *Fast Recovery:*

When TCP retransmits a packet using the fast retransmit method, instead of going back to slow start phase like it does when a retransmission timer expires, it goes to the congestion avoidance phase. This allows TCP to maintain its high throughput in low or moderate congestion condition.

The procedure is like this:

When the third duplicate ACK in a row arrives, first the missing segment is retransmitted using fast retransmit, then the current congestion window is halved and the congestion avoidance phase is followed starting at this value.

2.2 TCP Versions

There are different versions (implementations, flavors) of TCP in use today. These different versions are mostly the same, except for their congestion control algorithms. Many congestion control algorithms have been proposed during the last decade, to improve the performance of classic Tahoe/Reno TCP congestion control.

The original congestion control algorithm was first added to TCP by Van Jacobson at the end of the eighties, and is know as Tahoe TCP [37]. Then came its first modification

called Reno TCP [38]. Today's Internet stability is still largely a result of these congestion control algorithms.

As mentioned in Sec. 2.1, two variables, congestion window (*CWND*) and slow start threshold (*SSTHRESH*) are used to throttle the TCP sending rate to the channel in order to match the network's available bandwidth. Almost all the congestion control algorithms exploit the *Additive-Increase/Multiplicative-Decrease* (AIMD) method, which additively increases the *CWND* to grab the available bandwidth and suddenly decreases the *CWND* when network capacity is hit and congestion occurs via segment losses, i.e. timeout or duplicate acknowledgments.

The stability of the network and prevention of a congestion collapse is ensured by these AIMD algorithms, however, there's no guarantee of network resources being fairly shared and used.

Following is a short description of some of the more common TCP versions [39, 40].

2.2.1 Tahoe

Early TCP implementations were not concerned about network congestion, and they kept sending data from the source to the destination regardless of the network condition.

But as the networks expanded, and congestion problems appeared, the need for congestion control was felt. The first TCP extension aimed at controlling congestion and maintaining the throughput at the same time, was Tahoe TCP. It added a number of new algorithms and refinements to earlier implementations. The new algorithms include *Slow-Start*, *Congestion Avoidance*, and *Fast Retransmit*. The refinements include a modification to the round-trip time estimator used to set retransmission timeout values [37].

2.2.2 Reno

In Reno TCP, the enhancements incorporated into Tahoe were all kept, but a new procedure called *Fast Recovery* [38] was added to the Fast Retransmit operation of the original TCP.

This new algorithm helps TCP to grab the available bandwidth more rapidly after Fast Retransmit is performed, by avoiding TCP to go to Slow Start after a single packet loss.

Reno's Fast Recovery algorithm performs best when a single packet is dropped from a window of data, because the Reno sender can retransmit at most one dropped packet per round-trip time. In such situation, Reno outperforms Tahoe TCP by a significant amount. But when multiple packets are dropped from a window of data, the performance can degrade.

2.2.3 New Reno

New Reno [41] TCP is the leading Internet congestion control protocol in use and implemented today. It is a modified and improved version of Reno that avoids multiple reductions of the CWND when more than one segment from the same window of data is lost.

In Reno TCP, during the Fast Recovery phase, when *partial ACKs* are received, they cause TCP to go out of Fast Recovery. A partial ACK acknowledges some but not all of the packets that were outstanding at the start of that Fast Recovery period.

In New Reno this does not happen, but instead, these partial ACKs are treated as an indication that the packet immediately following the acknowledged packet in the sequence space has been lost, and should be retransmitted. As a result, when several

segments are lost from a single window, New Reno simply retransmits one lost packet per round-trip time until all of the lost packets from that window have been retransmitted. So unlike Reno, it doesn't wait for a retransmission timeout to expire.

2.2.4 Vegas

Vegas TCP [42] was a very innovative approach since it introduced a mechanism of detecting or sensing congestion in the network before the actual packet losses occur.

Three techniques are employed in TCP Vegas to help increase throughput and decrease losses. One results in a more timely decision to retransmit a dropped segment. The second enables TCP to predict or anticipate congestion before it happens, and adjust its congestion window (i.e. transmission rate) accordingly. And the third technique is a modification to the well-known slow start mechanism so as to avoid packet losses during this initial phase.

In particular, prediction of network congestion is realized by the use of two new parameters in TCP Vegas: *actual input rate* (*CWND/RTT*) and the *expected rate* (*CWND/RTTmin*), where *RTT* is the Round Trip Time and *RTTmin* is the minimum measured round trip time. The difference between these two parameters is compute at regular time intervals, and the amount of network congestion is inferred from this difference. If the difference is smaller than a threshold α then the *CWND* is additively increased, whereas if the difference is greater than another threshold β then the *CWND* is additively decreased; finally, if the difference is smaller than β and greater than α, then the *CWND* is kept constant.

So Vegas TCP actually proposes a new mechanism for throttling the congestion window that is based on measuring the network congestion status via RTT measurements.

2.2.5 Selective Acknowledgement (SACK)

Because of the cumulative nature of TCP acknowledgements, performance may degrade when multiple packets are lost from a single window of data. This is because TCP sender can only learn about a single lost packet per round trip time. Thus multiple lost segments in a window will cause a significant reduction in throughput.

Selective Acknowledgment (SACK) mechanism [43] was introduced to overcome this limitation. In SACK, the receiving TCP sends back SACK packets to the sender informing the sender of the exact data segments that have been received (in contrast to only sending cumulative acknowledgements, which show the correct reception of all segments up to a certain sequence number). The sender can then retransmit only the missing data segments.

Therefore SACK TCP uses retransmit timeouts as the recovery method of last resort, keeping all other properties of Reno and Tahoe. The main difference between the SACK TCP implementation and the Reno TCP implementation is in the behavior when multiple packets are dropped from one window of data.

2.2.6 Delayed ACK (DACK)

This is probably the simplest improvement to the operation of a TCP connection. Actually, as specified in IETF RFC 1122 [44], every TCP implementation SHOULD implement the Delayed ACK (DACK) algorithm, but its use is optional.

Efficiency can be increased in the Internet when a TCP receiving host chooses to send fewer than one ACK segment per data segment received; this is known as a DACK.

In DACK, an ACK is sent for every two data segments correctly received. If there is no second data segment in 200 ms, the ACK is not delayed any more, and will be sent for that single segment. This delay is adjustable, but should never be more than 0.5 second.

2.3 A Short note on UDP

As one can see from Fig. 2.1, there is also another Transport protocol: User Datagram Protocol (UDP). UDP is mostly used for real-time (delay-sensitive) applications, such as transfer of video/audio streams, online gaming and other multimedia applications. Compared to TCP, which provides a reliable data transfer (that is the packets in a network that employs TCP will be delivered *in order*, *without loss*, and *without error*), UDP provide an unreliable transfer which means that packets could be lost, received in different order, or with errors. But TCP employs congestion and flow control algorithms to provide such reliability that is not included in UDP. As a result, TCP users' data rates are limited but the UDP can push the data in the network and grab the available bandwidth as much as possible. This latter feature of UDP makes it a favorite candidate for multimedia data transfer in which more bandwidth is required.

Because the UDP protocol does not involve any error or flow control mechanism, it is much simpler than TCP, and hence there is generally less delay in delivering packets with UDP (no congestion/advertised window limit, no waiting for acknowledgements, no retransmission timer time-outs). This is also a feature of UDP which makes it more attractive for delay-sensitive real-time applications.

2.4 Problem Statement

Wireless ad hoc networks can provide users with a very convenient communication environment, which is infrastructure-free and operates over a shared multihop wireless channel. Such networks are an example of the peer-to-peer networking model.

Although the communication is done over error-prone wireless links, network applications often require reliable data delivery, and they depend on a reliable transport protocol for this purpose.

TCP has been playing this role for years now, and has successfully indicated its capability in doing this. Besides, a large base of applications is already making use of this protocol. For all these reasons, TCP also seems to be the natural choice for users of ad hoc networks that want to communicate reliably with each other, and with the rest of the Internet. This will also allow and facilitate seamless integration of such networks with the existing Internet.

Two key requirements of an ad hoc network are reliable data transfer and congestion control. TCP supports both of these features, but the question is that how it interacts with the wireless protocols, specially the MAC layer protocols. This question arises from the fact that TCP has been designed and fine-tuned for wired networks, in contrast to wireless multihop networks.

Both MAC and TCP layers attempt to provide efficient transport in a shared environment. The difference is that MAC layer has only a narrow link-layer view of the network, while TCP has a broader end-to-end view and control over errors and congestion.

Thus, TCP breaks the first rule that we have in a layered network, that is, independency of a higher layer protocol from its underlying lower layers such as link layer and physical

layer. TCP makes strict assumptions on the reliability of the lower network layers, probably because it has been designed for wired networks with such reliable channels. As the main reason for this poor performance of TCP, we can raise the fact that TCP cannot distinguish between packet losses due to wireless errors from those due to congestion. Moreover, TCP sender cannot keep the size of its congestion window at an optimum level and always has to retransmit packets after waiting for time-out, which significantly degrades the end-to-end delay performance too [18].

Remembering the mechanisms of TCP from previous sections, we know that TCP assumes that all losses are due to congestion, thus after a loss it reduces its transmission rate so as to allow routers to drain their queues, and then gradually increases this rate to probe the network.

TCP recognizes a congestion occurrence within a network by a packet loss, and performs congestion control by throttling the congestion window. The problem arises right here: packet losses due to transmission errors on the wireless link (a result of the link's susceptibility to the problems of path loss, shadowing, multipath fading, and interference) cause unexpected degradation of TCP throughput.

The reason for this degradation is that TCP, after seeing a packet loss (which has been caused by an error on the wireless link, and not by congestion in the network), automatically lowers its congestion window, limiting the number of on-the-fly (unacknowledged) packets, and thereby limiting the throughput. Because errors are frequent on a wireless link, this congestion avoidance mechanism is engaged several times during a connection, and causes the overall throughput (which is seen by the application layer) to drop drastically, compared to a same-capacity wired link.

2.5 Previous Work

As a result of all the issues and problems associated with the performance of TCP over wireless channels, a research challenge has been started in recent years on how the transport protocol can be *modified* to be efficient even in error-prone and high-latency wireless channels. Note the word modified, as opposed to proposing or suggesting a new protocol, since, as stated earlier, the TCP protocol suite is already dominant in the world of Internet and is implemented in almost all hardware and software (operating systems, applications) in use today. So at best, and for the foreseeable future, we as researchers can try to modify or in other words *tweak* the current TCP protocol to make it more efficient and suitable for wireless links.

According to [18], this topic has made up many masters and PhD theses as well as numerous technical journal and conference papers in the last decade, bur the appropriate solution is yet to be discovered.

Some more important and major enhancements include:

- *Selective Acknowledgement* (SACK) [26]
- *Indirect (Split) TCP* (I-TCP) [27]
- *Snoop Protocol* [28]
- *Explicit Congestion Notification* [32]
- *Explicit Loss Notification* [33]

In SACK, instead of the regular cumulative acknowledgements, a selective acknowledgement mechanism is used, so that the exact missing data segments will be known to the sender.

In I-TCP, the TCP connection between a mobile host and a fixed host is split in two: one between the mobile host and the immediate AP serving that mobile (over the wireless medium only), and the other between the AP and the fixed host (over a wired network only). This way, the issues associated with a TCP over wireless connection is taken care of exclusively, and transparently to the rest of the link, by the TCP software in the AP. This approach is suitable only for infrastructure single-hop networks.

The snoop protocol is mainly an improvement over I-TCP, and uses the same idea. It is based on the assumption that retransmitting lost segments over the wireless link locally (using the AP), will be better compared to the original TCP end-to-end retransmission. Again this works well with one-hop connections with the presence of a base station (AP), but is not applicable to multi-hop networks, since the whole end-to-end connection is assumed to be wireless in such networks.

The ELN and ECN proposals, suggest using notifications to inform the sender of a loss or a congestion in the network, so that the TCP sender is able to distinguish between losses caused by congestion and those caused by wireless link loss. These two protocols are very suitable and efficient for all kinds of wireless networks, but they require major modifications in both TCP software in hosts, and also intermediate routers and network layer IP packets.

Other examples are listed under references [19-25].

The focus of work in [19] is on how TCP and UDP perform in a realistic environment, using different hardware/software setups. There are only three nodes communicating with each other, so the scenario is a single hop one. Conclusions are derived on how different

processor speeds, wireless interface types, and operating systems affect the end-user performance.

Some link layer enhancements are suggested in [20], to hide link layer impairments from the upper TCP and UDP layer. It is concluded that TCP-unaware link layer schemes perform excellent in most cases, unlike TCP-aware schemes. The scenario is a 1-hop link and uses base stations in both a GSM and an 802.11 network.

A multi-hop wireless network is used in [21], but the IEEE 80.11 is not used as the MAC layer protocol. Instead, the classic CSMA and a protocol called FAMA are assumed to operate at MAC layer. The channel is assumed to be perfect, and the TCP packet size is fixed at 1460 bytes. The performance measure considered is throughput. UDP protocol is not considered. It concludes that the FAMA protocol is generally performing better than CSMA in most cases. It also predicts that the use of Selective Acknowledgement in TCP will make it more efficient in heavy collision environments.

The Delayed ACK option of TCP is analyzed in [22], and it suggests using DACK with the parameter d=3 instead of the default value of 2. Also it concludes that limiting TCP's maximum window size when the number of nodes is not known, will result in dramatic decrease of throughput.

In [23], the performance of TCP is monitored for different values of maximum window size. Then it suggests that assuming the number of hops in a connection is known, setting the window size to h/4 (where h is the number of hops) will achieve the best throughput. Also, it introduces a link layer enhancement called LRED which controls the packet injection rate into the network so as to avoid overloading and packet drops at the link

layer, and thus results in up to 30% increase in throughput. No results are obtained for UDP, and how it reacts to overloading and causes excessive packet drops.

A cellular network with a base station and one wireless device operating under the IMT-2000 standard with a bandwidth of 384 Kbps is considered in [24]. Thus there is only one hop involved in the connection, and the setup is cellular (infrastructure) rather than WLAN. They propose an adaptive FEC scheme combined with ELN, and show that this method achieves better performance than conventional fixed FEC.

And finally in [25], the throughput performance of TCP is measured over a connection of several hops. The delay/jitter and packet loss measures are not presented, and a single probability of error of 1 percent is assumed when the channel is not perfect. Also, the effect and interaction of simultaneous TCP and UDP traffic, as well as the effect of the imposed load on the network by a UDP sender, is not analyzed. The article concludes that 802.11 ad hoc networks are not suitable when operating on more than 2-3 hops. Although our results show that UDP can achieve very good performance in hop counts of up to 6, if used in balance and in a regulated fashion.

2.6 Justification and Thesis Approach

The lack of analysis of several aspects of TCP/UDP operation under 802.11 ad hoc WLANs was felt in the current literature, and thus motivated us to focus on some of those aspects. Among the issues considered in this work, are:

1. When measuring the performance, in addition to the *throughput* parameter which is most commonly used, the *delay*, *jitter,* and *packet loss* parameters (commonly ignored) are also measured and compared under different network configurations or conditions.

These parameters are of utmost importance in real time and delay sensitive applications, like voice and video transmission.

2. The effect and implications of introducing UDP CBR load on an ad hoc network, when it is below the network capacity, and when it goes beyond that capacity, is illustrated. It is shown how vulnerable the network is to UDP overloading.

Besides, the interaction of simultaneous TCP/UDP connections, how fair TCP connections can run together, and how UDP can negatively affect ongoing TCP connections by showing the capture effect, is shown.

3. Operation of both TCP and UDP in presence of channel packet errors, and how different they react to it, is analyzed and numerically presented for a wide range of packet error rates.

4. Performance of different TCP versions is simulated over the multi-hop wireless network, and it is shown that the best choice varies depending on the number of hops involved.

5. The packet size is always considered to be a maximum of 1500 bytes, but here we analyzed the effect of different packet sizes all the way to 3000 bytes, which reveals the throughput saturation effect when going beyond 2500 bytes. Assuming the ad hoc network is independent (not connected to external networks), larger-than-usual packet sizes are recommended to be used for optimum throughput.

The approach throughout the thesis is performing independent yet related experiments, each focused around a specific issue or parameter of TCP/UDP operation (individually, or together). Then presenting the results of the experiments, followed by a detailed

discussion of why and how the network reacted in the way it did, and finally what should be done to get the optimum performance.

In brief, the thesis objectives are twofold:

1. To add new insight, new results and analysis, and new measures on the performance of wireless ad hoc multi hop networks, not previously covered. This is in the hope of providing the research community with a set of results which could be used effectively in future works.
2. To make suggestions and improvements based on the obtained results and the understanding of the author of those results.

Chapter 3 Simulation Model

3.1 Simulation Software

The software used for the simulation is ns-2 (Network Simulator Version 2, [29]).

Ns is a discrete event simulator targeted at networking research. It provides substantial support for simulation of TCP, routing, and multicast protocols over wired and wireless (local and satellite) networks.

Ns began as a variant of the REAL network simulator [30] in 1989, and has evolved substantially over the past few years. In 1995 ns development was supported by DARPA (Defense Advanced Research Projects Agency) through the VINT (Virtual InterNetwork Testbed) project at LBL (Lawrence Berkeley National Laboratory), Xerox PARC, UCB (University of California, Berkeley), and USC/ISI (University of Southern California/Information Sciences Institute).

It is a free and open-source software, which is continually updated and expanded through its developers and also contributions made by its huge user community.

Ns is written in C++, and uses the OTcl as a command and configuration language. (OTcl is MIT's object-oriented extension of the original Tcl/Tk [31]).

It is an extremely powerful and flexible language both in its original pre-packaged form, and even more flexible given the opportunity that the source code is available and modifiable.

The ns program runs (compiles) on UNIX-like platforms (FreeBSD, SunOS, Solaris, Linux), although there is a limited pre-complied version available for Windows, under Cygwin. (This version is not supported and is prone to having problems).

In my case, all the simulations are done using the most recent version of ns-2 (Release 2.27, January 2004), on a Linux PC platform.

3.2 Simulation Setup

3.2.1 Configuration Parameters

The base wireless network used for the simulations is an IEEE 802.11 network in the ad hoc mode. Each mobile node has an omni-directional antenna with unity transmit and receive gain, and is 1.5 meters above the node.

The following parameters and settings were *constant* during all the simulation runs:

Physical channel: Wireless, using AT&T Lucent WaveLAN wireless system (2.4 GHz DSSS, CSMA/CA)

Radio channel data rate: 2 Mbps

MAC Layer: IEEE 802.11 DCF, using RTS/CTS/DATA/ACK pattern (unless otherwise specified)

Network area: 1500m * 1000m rectangular

Simulation length: 300 seconds

Transmission range of mobile nodes: 250m

Carrier sensing and interfering range of mobile nodes: 550m

Wireless interface (MAC) buffer type: Drop-tail priority queue

Radio propagation model: Two-ray ground

ACK packet size: 40 bytes

UDP packet size: 1480 (bytes)

Routing protocol: AODV

TCP window size (advertised): 20 (packets)

TCP traffic source: FTP

The following parameters were *modified/changed* during simulation runs to reflect different network conditions:

(Underline font indicates the values which were used as default)

Wireless link error rate: from 0.0 to 0.3 (packets in error)

Wireless interface (MAC) buffer size: 5 to 50 (packets)

TCP packet size: 256 to 3000, 1460 (bytes)

TCP flavor: Tahoe, Reno, New Reno, Vegas, SACK, Delayed ACK, SACK + Delayed ACK

RTS/CTS scheme: used, not used

UDP traffic source: CBR with bit rate 0 to 2000 Kbps, 1.7 Mbps

For more explanations about these parameters and their implementation, please refer to *The ns Manual* (formerly known as *ns Notes and Documentation*) [35].

AODV routing protocol is chosen because it represents the most commonly implemented ad hoc routing protocol. ACK packet size of 40 bytes is the standard in TCP. TCP packet size of 1460 is used as the default size, since with the added TCP/IP header of 40 bytes, the network layer packet size will be 1500 bytes, which is the standard MTU (Maximum Transmission Unit) for most of today's network routers.

MAC buffer size of 50 packets is commonly used in simulation of this kind of scenario, as in [22, 23].

3.2.2 Performance Measures

The following measures were used to evaluate the performance of TCP or UDP protocols in different network configurations or conditions:

Throughput:

This is a measure of the average throughput of receiving bytes at the transport layer of a specific node.

It is obtained by dividing the total number of received bytes at the transport layer, by the simulation time.

The unit is Kbps (kilo bits per second), unless otherwise specified.

Delay:

This is a measure of the average delay of the received packets.

It is obtained by summing up the individual packet delays at a specific node, and dividing the sum by the total number of received packets.

The delay of each individual packet is defined as the time passed between when the packet was sent by the sender agent, and when the packet was received by the receiving agent. (both at transport layer)

The unit is milliseconds (ms).

Jitter:

This is a measure of the average jitter of the received packets.

It is obtained by summing up the individual packet jitters at a specific node, and dividing the sum by the total number of received packets.

The jitter of the *i* th packet is defined as:

$j[i] = | d[i+1] - d[i] |$

where d[i] is the delay (as defined above) of the *i* th packet, and |x| is the absolute value of x.

The unit is milliseconds (ms).

Packet Loss:

This is the percentage of packets lost during a connection between a source/destination pair of nodes.

Lost packets are defined as packets sent by the sending node's router (Network layer), but never received by the receiving node's router (Network layer).

There could be several causes for packet loss:

- Loss at physical layer, due to wireless link errors
- Loss at MAC layer, due to packet collisions in the shared medium, or due to MAC buffer overflows
- Loss at network layer, due to packet drops by routing agents

It is expressed as the percentage value of the ratio of lost packets to total number of sent packets.

3.2.3 Network Topology

Throughout the experiments, a topology is set up which is referred to as a *chain* or *string* topology in the literature [21, 23]. It consists of seven nodes numbered from 0 (left) to 6 (right), all located along a straight line, and separated by a distance of 200 meters. (Fig. 3.1)

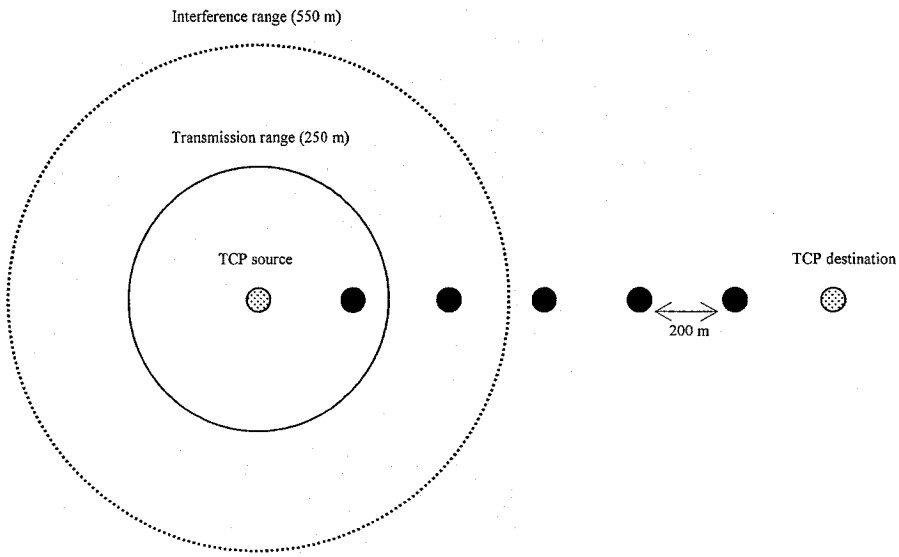


Figure 3.1 - Network topology

For each wireless node, the transmission range is 250 meters, the carrier sensing range and the interference range are 550 meters. As a result, in this scenario, each node can transmit or receive packets only from its immediate neighbor, but can sense the carrier (and hence stop its transmission if there is one already going on) from nodes up to two hops apart from it.

All nodes are static, and there is no power-saving mode.

This setup allows us to investigate and undermine most properties of a multi hop network:

1. The number of hops ranges between 1 and 6, which covers most realistic scenarios.
2. The transmission range of 250 m is chosen, as it's usually the supported range in most of today's interface implementations.
3. The node spacing of 200 m is chosen, so that each node is only able to communicate with its neighbor, and thus when a source/destination pair is not

able to communicate directly, all intermediate nodes are involved in forwarding packets. This will reveal the true nature of a pure multi hop network.

4. The nodes are considered to be static, so that the effects of routing are minimized and nodes don't move out of range of each other (resulting in destination unreachable events), thus making the focus only on TCP/UDP and MAC layer operation.
5. The justification for each of the individual setups and traffic patterns is given at the beginning of the experiment, under the Motivation title.

3.3 Simulation Procedure

In each simulation experiment, either one or two of the configuration parameters (Sec. 3.2.1) are changed, and the performance of the network in terms of the parameters stated in Sec. 3.2.2 is measured.

These performance measures are extracted and then analyzed from *Trace* files which are generated by the ns program after each simulation run.

The 300 seconds that each simulation runs, allows for complete stabilization of the network and the achieved throughputs. This was verified by monitoring the instantaneous throughput of received packets over the duration of simulation time.

Chapter 4 Simulation Results

In this chapter, the major results of the simulation experiments are given.

In each experiment, I've tried to follow the following sequence of explanations:

1. Motivation of doing the experiment
2. Type of traffic used and the source/destination node pair(s) (FTP, CBR, ...)
3. Variable parameters along with their ranges or values
4. Results (explained below)
5. Interpretation and/or explanation of the obtained results
6. Suggestion(s) for possible improvement(s)

The results are mostly presented in graphs with the horizontal axis representing one of the configuration parameters, and the vertical axis representing one of the performance measures.

Please note that I have used the configuration parameters and general setup of Experiment #1 as my default (or basic) experiment, meaning that in all other experiments, everything is the same as in Experiment #1, and the only parameters changed are the ones explicitly specified in the explanation of the experiment.

(These default parameters are also listed in Sec. 3.2.1)

4.1 Experiment #1: Basic

Motivation:

This experiment is the most basic one, serving as a reference for all the other setups.

The purpose is to measure TCP performance over different number of hops. (up to 6)

Traffic pattern:

In this scenario, Node 0 is always the source node, but the destination node ranges from node 1 to node 6. So the hop count is ranging between 1 and 6.

An FTP application (with unlimited data to send) is attached to Node 0, and it starts sending its data to the transport agent (TCP) at 0 second and stops at 300 seconds.

The TCP version used is New Reno, each TCP segment is one packet (so virtually packet = segment), and the packet size is 1460 bytes. The ACK packet size is 40 bytes.

The wireless medium is assumed error free, TCP maximum (advertised) window size is equal to 20 packets, and the MAC buffer size is 50 packets.

Variable Parameters:

The only parameter change during this experiment is the receiving node, or in other words the hop count.

Results:

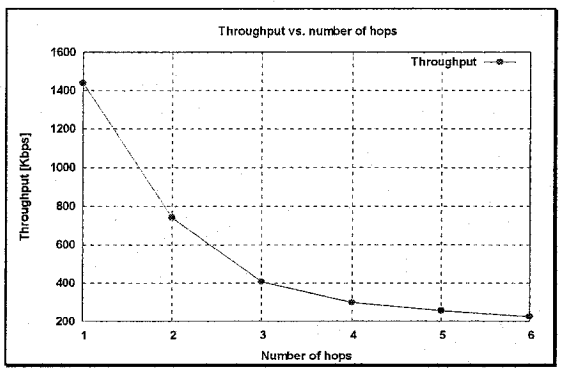


Figure 4.1 - Throughput vs. number of hops

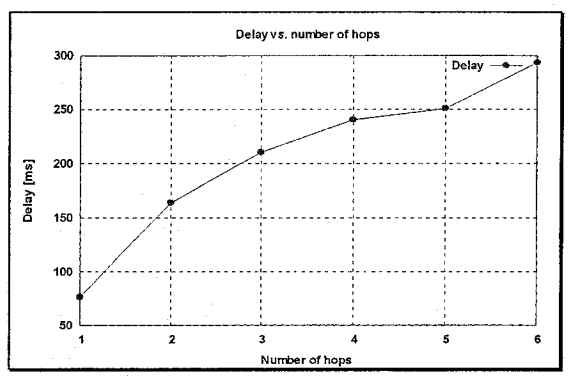


Figure 4.2 - Delay vs. number of hops

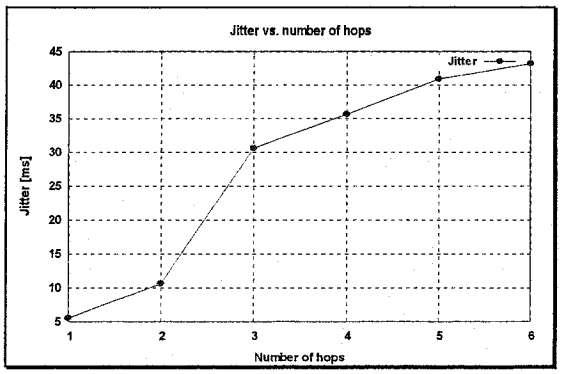


Figure 4.3 - Jitter vs. number of hops

Interpretation:

Throughput:

Figs. 4.2 to 4.4 show very interesting results. In terms of throughput, it is seen that as the number of hops (from source to destination) increases from one to two, and then from two to three, there is a large (almost 50%) drop in TCP throughput. But as the number of hops becomes larger, the throughput stabilizes.

First, we should note that although the channel data rate is 2 Mbps, the achievable throughput at TCP layer is only about 1.4 Mbps, even for a one-hop link. This is because there are protocol overheads that use some of this bandwidth, namely: MAC overhead and TCP overhead[1].

- MAC overhead: consists of MAC frame overhead of 34 bytes, Ack frame of 14 bytes, and the CSMA/CA scheme itself. Essentially at the MAC layer a type of stop-and-wait transmission scheme is used, where each frame is transmitted only if the acknowledgement of successful receipt of the previous frame has been received.

[1] Because all nodes are stationary, and each node can only send/receive packets with its immediate neighbor, the network layer (routing) overhead in this scenario is negligible.

- TCP overhead: 40 bytes of TCP/IP header size, and the Data/Ack transmission of TCP whereby an Ack packet (of size 14 bytes) should be received for all the previous transmitted packets, before a new packet or a new set of packets are transmitted.

Therefore, the combined effect of these two types of overhead result in an almost 30% drop in throughput at the transport layer compared to raw channel data rate.

Second, the reason for the rapid drop as the hop count increases is as follows:

In a Mobile Ad hoc Network (MANET) like this, each node acts as a router: It receives packets from neighboring nodes, and forwards them to their destinations. In this specific chain scenario, for example, suppose the case of 2 hops (three nodes, Fig. 4.5). The middle node, which acts only as a forwarder, should receive TCP packets from node 0 at its left, and forward them to node 2 at its right. At the same time, it should receive TCP Ack packets from node 2 and forward them to node 0. But the point is that none of these four steps can be done at the same time, because of the MAC CSMA/CA scheme. While a packet is on the fly in the shared medium, the node should wait until it reaches its destination and an Ack (MAC layer Ack) is received.

As a result, it is clear to see why, when the number of hops increases from one to two, there is such a dramatic decrease in TCP throughput.

In addition, because the MAC scheduling is not ideal, there will always be collisions, between data packets, between Ack packets, and specially between data and ack packets which are traveling in opposite directions. These collisions will further degrade the performance, because of the loss of Ack packets, and causing TCP to reduce its window and consequently, its throughput.

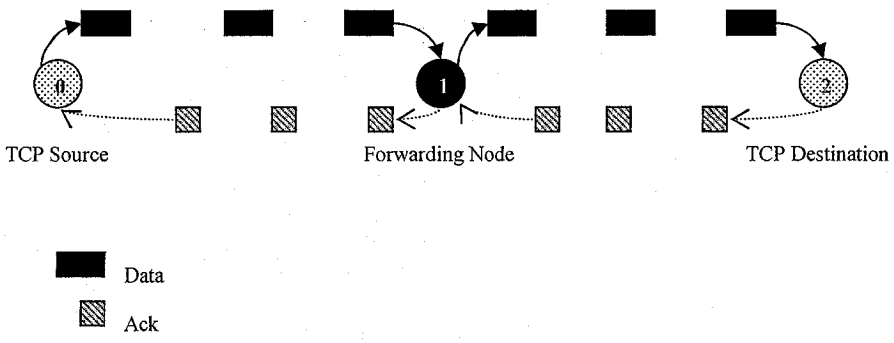


Figure 4.4 - Packet and Ack forwarding

Delay and Jitter:

The delay and jitter performances are shown in Fig. 4.3 and 5.4, respectively. These two measures are commonly ignored while analyzing the performance of TCP over wireless ad hoc networks, although they are two very important performance measures when considering real-time (streaming) media, like audio, video, or gaming, which is quite common in these days' network applications, wireless included.

For these types of applications, if the packets are received with a delay higher than a specific value (depending on the application, e.g. 250 ms for voice communications), they are useless and will be dropped by the receiving agent. Therefore in this case, high delay could be translated to high loss rate, which is of course not desirable.

On the other hand, jitter (which is variations in delay), defined in Sec. 4.2.2, is a very annoying factor for streaming data (voice, video, and the like) decoders. As a stream of packets traverses the network, the information between them can be altered. These variable delays introduce jitter at the receiver. In stream applications, the decoder that plays back the information must be fed a steady stream of information blocks (packets) for the system to operate correctly. Typically the decoder assumes that all information blocks will arrive within some maximum delay, and it plays back information within such

a delay. The receiver uses a buffer to hold blocks of information until their playback time. The jitter is a measure of how much the blocks of information will tend to bunch up and hence the buffer size that will be required to hold the blocks.

Now, as it can be seen from Fig. 4.3, the delay is increasing almost linearly with the number of hops. The 250 ms ceiling is reached at around 4-5 hops, this shows that for real-time delay-sensitive applications, this kind of configuration is not desirable. So, other options should be considered, like using another transport protocol (UDP + RTP) which is less complex than TCP, or serving the mobile station using a local AP.

Jitter also increases with the number of hops, from a value of only 5 ms for 1 hop, to a value of about 43 ms for 6 hops. The reason is that, as there are more nodes involved in forwarding the packets, and the packets travel more hops, the potential of collision increases, thus making the MAC layer involve its retransmission scheme and retransmit the collided packets, while in the meantime, other packets which were originally behind the collided packets, are successfully delivered to the receiver. This will cause different arrival times, and variations in delay, which is the jitter.

Suggestions:

The only suggestion which can be made here (though not very significant) is that using wireless connections with a high (more than 3-4) hop count should be avoided as much as possible, specially when high throughput and low delay/jitter performance is required. The alternative could be using an infrastructure (or hybrid, i.e. infrastructure combined with ad hoc) network instead, so that far-apart nodes can also have the possibility of communicating with each other through nearby base stations, and thus avoiding the problems of multi hop network.

4.2 Experiment #2: Effect of Packet Size

Motivation:

The purpose of this experiment is to see what is the effect of different TCP packet sizes on the performance of a wireless multi hop network.

Traffic Pattern:

The traffic pattern used is the same as Exp. #1.

Variable Parameters:

In addition to the hop count, a second variable is introduced here: TCP packet size.

The number of hops changes like Exp. #1 from 1 to 6, and the packet size from 256 to 3000 bytes.

Note that packet size in our experiments is equivalent to segment size in a TCP connection, because we are using single-packet segments.

Results:

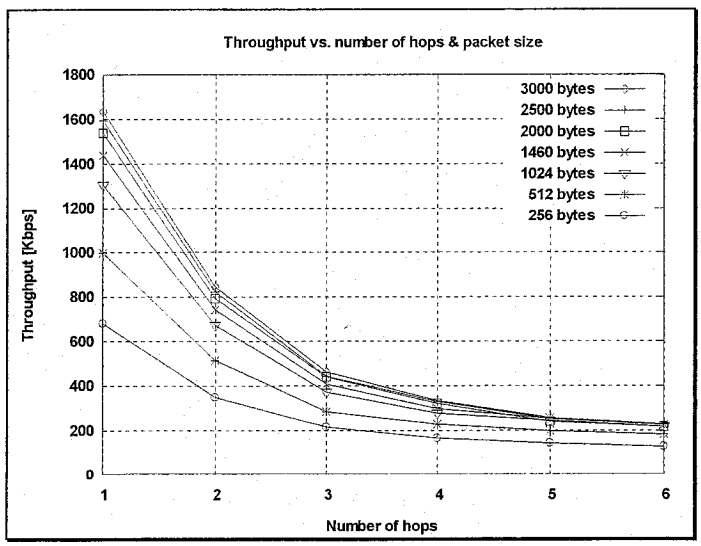


Figure 4.5 - Throughput vs. number of hops & packet size

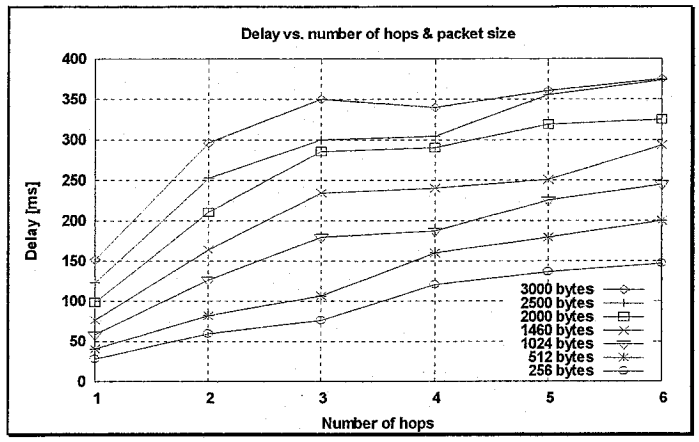


Figure 4.6 - Delay vs. number of hops & packet size

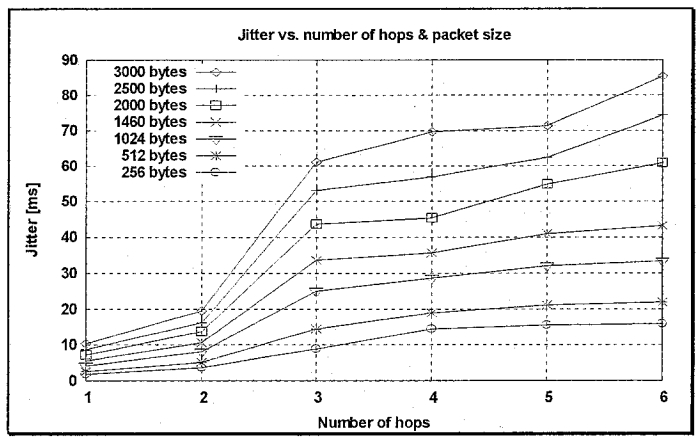


Figure 4.7 - Jitter vs. number of hops & packet size

Interpretation:

Throughput:

It is very clear from Fig. 4.6 that the throughput is improved as the TCP packet size is increased. The reason is that with bigger packet sizes, bigger chunks of data can be transmitted and received by the receiver, before the sender waits for acknowledgement and sends the next set of packets.

Furthermore, with larger packet sizes, the percentage of TCP/IP overhead (40 bytes) will be less. (for example, the overhead for a packet size of 256 bytes will be about 15% percent, while that of a 1460-byte packet will be about 3%)

With a deeper examination of the figure, we can see that:

1. The difference in throughput when going from packet sizes of 256 to 512 (a 47% increase) and from 512 to 1024 (a 30% increase) is much bigger than the difference when going from a packet size of 1460 to 3000 (an increase of only 13.7%); although in all three cases the packet size is doubled. The reason is that, with larger packet size, when a packet is lost and needs to be retransmitted, it takes longer to send it, compared to a smaller packet. In other words, more of the bandwidth (or time) will be wasted when large packets are lost. As a result, we can't increase the packet size indefinitely to get indefinite throughputs!

This kind of *saturation* trend is better visible from Fig. 4.9, which shows that the throughput approaches a saturation limit, as the packet size increases beyond 2500 bytes.

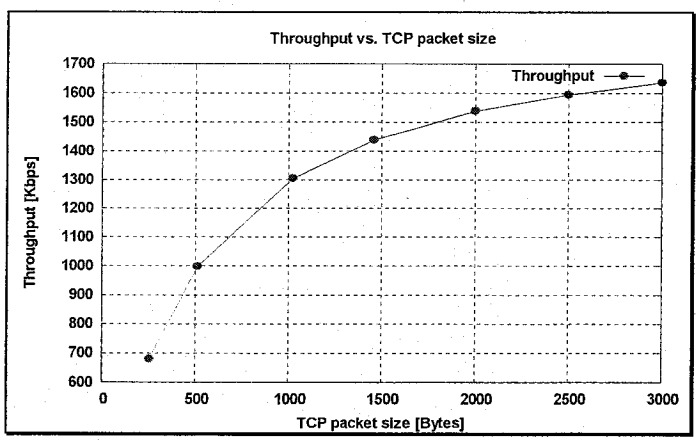


Figure 4.8 - Throughput vs. TCP packet size, single-hop connection

2. As the number of hops increases, all packet sizes converge to almost the same throughput, so much that for a 6-hop connection, all packet sizes will yield a 200 Kbps throughput.

The reason is exactly the same as the one stated in 1: packet loss and retransmission. Here, as the number of hops increase, more and more packets are lost (due to collisions, link layer drops because of medium access time-outs, buffer overflows, ...), so more retransmissions are necessary, and as a result bigger packet sizes won't achieve considerable improvements.

- Note: Although packet sizes of more than 1460 (+40 = 1500) bytes are not common in today networks (because of the router MTU limitations of 1500 bytes), I have gone to sizes of up to 3000 bytes, to illustrate the effect of larger packet sizes. I'll explain more about this in the *Suggestions* part of this section.

Delay and Jitter:

Figs. 4.7 and 4.8 show the delay and jitter performance. It can be seen that with larger packet sizes, both the delay and jitter increase, compared to smaller packet sizes.

The delay increases because transmission, queuing, and reception of large packets take longer, so the final end-to-end delay will obviously be longer.

Note that the gaps between delays for different packet sizes remain almost constant as the number of hops increases; which means bigger packet size doesn't necessarily translate to longer delays over lengthier hops.

But it's very interesting to see that this is not the case for jitter. First, it is seen that the jitter also increases with larger packet sizes, but as more hops are involved in the connection, the gap between jitters for different packet sizes also widens: For a single-

hop connection, jitter for all packet sizes are in the range of about 2 to 10 ms, while jitter varies between 15 and 85 ms for a 6-hop connection.

Suggestions:

As witnessed by the results of this section, larger packet size translates to higher throughput. On the other hand, as it was mentioned above, packets larger than 1460 (or 1500 bytes in network layer), will cause the network layer to split the segments to route them via paths which don't support packet sizes larger than 1500 bytes, and this will introduce more delays and complexities.

But, since here we are discussing ad hoc networks, and there could be scenarios where the only nodes involved in a network are these mobile nodes, it is recommended that larger packet sizes (up to 2300 bytes which is the MAC frame data payload size limit) be used in order to increase the throughput.

The price to pay for this higher throughput is longer delays and higher jitters. So, if the application is not delay/jitter sensitive (like file transferring, downloading, Web browsing, email, and the like), those large packet sizes are more desirable to use.

But if the application is a real-time or streaming one (like voice conversations, live audio/video transmissions, interactive gaming, ...), it is better to bound ourselves to smaller packet sizes.

One could make a trade-off here, depending on the specific usage or application of the wireless network.

4.3 Experiment #3: Effect of RTS/CTS Usage

Motivation:

As explained in Sec. 1.5.3, since using the RTS/CTS scheme is an optional function in implementing 802.11 networks (either ad hoc, or infrastructure), it is tempting to see what would be the effect of using this scheme in our specific scenario.

Traffic Pattern:

The traffic pattern used is the same as Exp. #1.

Variable Parameters:

In addition to the hop count, here the RTS/CTS scheme is turned off in the first configuration, and is turned on in the second configuration.

In ns-2, the RTS/CTS scheme is used by default. To turn it off, the value of 5000 is assigned to the RTS-Threshold parameter, and since there will be no packets of size larger than 5000 bytes, this means that the RTS/CTS scheme is never used.

Results:

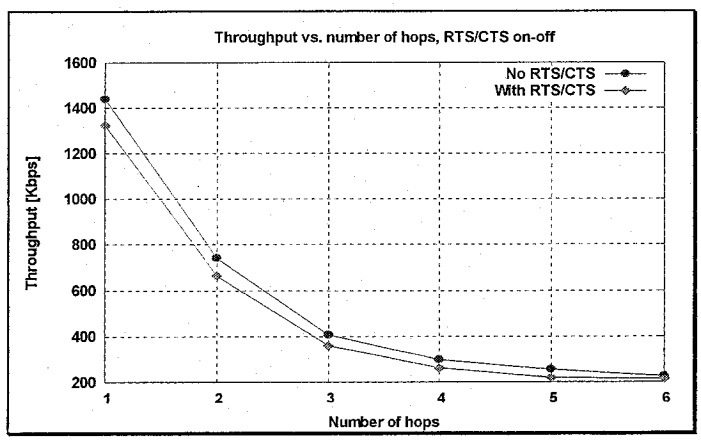


Figure 4.9 - Throughput vs. number of hops, RTS/CTS on-off

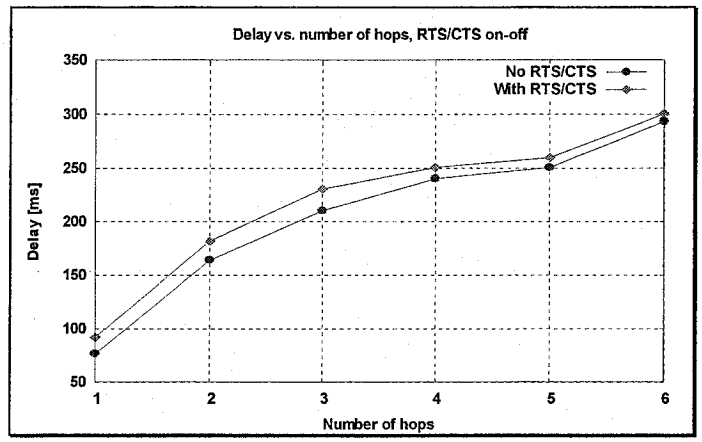


Figure 4.10 - Delay vs. number of hops, RTS/CTS on-off

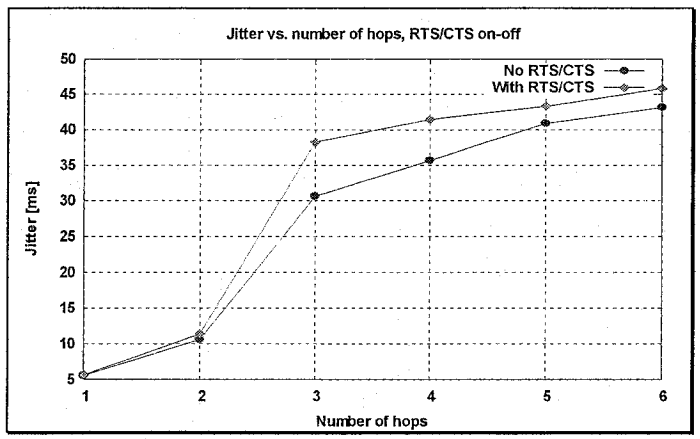


Figure 4.11 - Jitter vs. number of hops, RTS/CTS on-off

Interpretation:

Throughput:

Fig. 4.9 shows that there is a roughly 8% (for one hop) to 15% (for 5 hops) reduction in throughput, when using the RTS/CTS scheme, compared to when turning it off. This reduction in throughput is more noticeable in higher number of hop counts.

The reason (as discussed in Sec. 1.5.3) is the overhead that the scheme brings in, when being used for transmission of all packets, which does not compensate for the few number of collisions that may occur otherwise.

Delay and Jitter:

Figs. 4.10 and 4.11 show that the delay and jitter increase when using RTS/CTS scheme. Again this is because of the overhead that the scheme introduces. When there is an RTS/CTS handshaking before each transmission, the delay involving sending the RTS, receiving and processing it, sending back the CTS, and again receiving and processing it, will be added to the overall end-to-end delay of transmitting a packet.

As for jitter, the increase is not much in a one-hop and two-hop connection, and can be ignored. But for higher hop counts, the jitter increases about 7-10%. The reason is that the RTS/CTS scheme makes the transmission of each packet more complex, and complexity in a system (wired or wireless) always degrades jitter.

Suggestions:

The use of RTS/CTS is not recommended in simple and lightly-loaded networks, since, as seen from the results above, it not only doesn't resolve any issue, but also introduces more overhead to the network, and causes degradation in performance in terms of throughput and delay/jitter.

So, as a network administrator, one should monitor the wireless LAN for occurrence of collisions. If there are a large number of collisions, and the users are relatively far apart and likely out of range, then one can try turning on the RTS/CTS functionality on the mobile nodes.

After activating RTS/CTS, simple tests can be done to determine if the number of collisions is less AND the resulting throughput is better. Because RTS/CTS introduces overhead, it should be shut off if there is a drop in throughput, even if there are fewer collisions. After all, the goal is to improve performance.

And as for delay/jitter, activating RTS/CTS will provide for longer delays and worse jitters, in any kind of network.

4.4 Experiment #4: TCP Versions

Motivation:

As summarized in Sec. 3.2, there are several TCP versions and implementations which are completely defined and are being used on different platforms today. Each of these versions, have different performance characteristics, specially in wireless scenarios and in presence of link errors.

There has been some work on performance analysis and comparison of different TCP versions, like in [39, 40]; but most of these studies are performed on wired links, and specifically no previous study was found on the performance (throughput, delay, jitter) of TCP implementation in wireless multi hop ad hoc networks, with link errors.

I have simulated how some of these TCP flavors act in such scenarios, using the available TCP implementations in the ns-2 simulator.

Traffic Pattern:

The traffic pattern used is the same as Exp. #1.

Variable Parameters:

In addition to the hop count, and everything else remaining the same, these TCP versions have been used for the TCP connection between the source and destination:

Tahoe, Reno, New Reno, Vegas, DACK, SACK, SACK+DACK

In DACK, an ACK is sent for every other segment (packet in our case), and if the next segment doesn't arrive within 200 ms, an ACK is sent without further waiting.

Results:

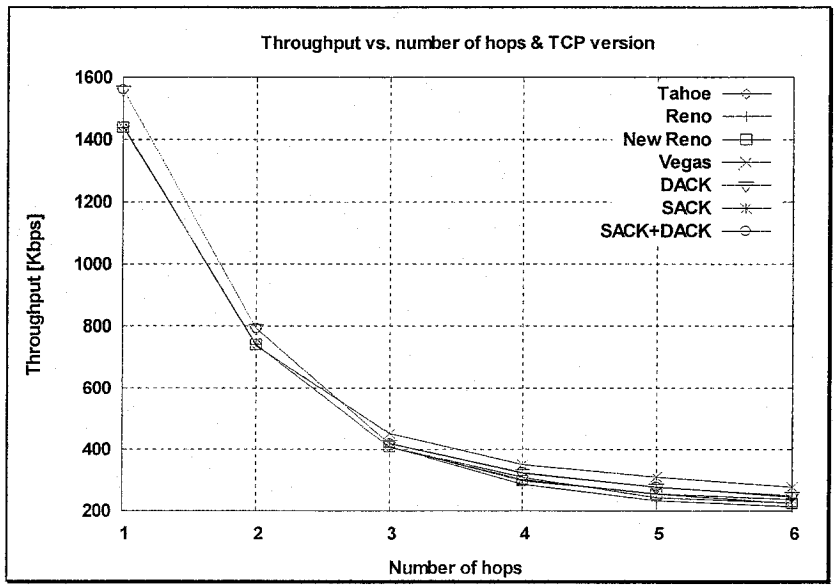


Figure 4.12 - Throughput vs. number of hops & TCP version

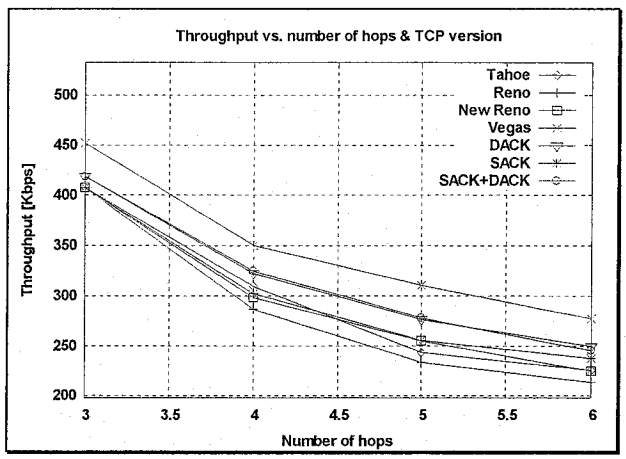


Figure 4.13 - Throughput vs. number of hops (3 to 6) & TCP version

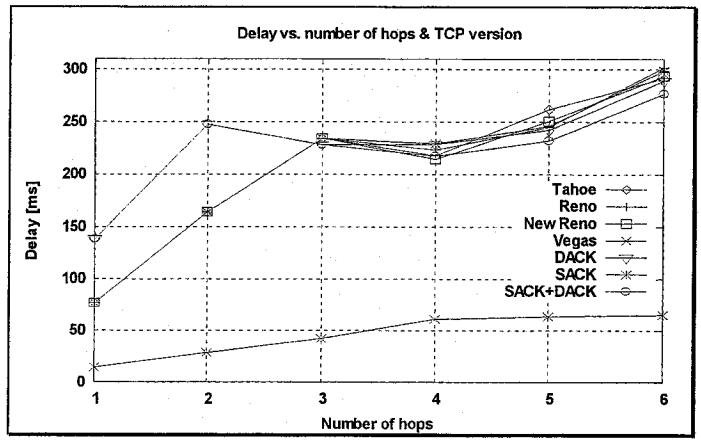


Figure 4.14 - Delay vs. number of hops & TCP version

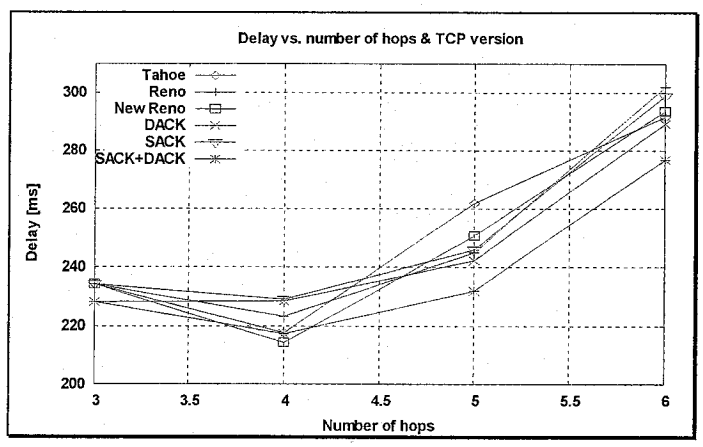


Figure 4.15 - Delay vs. number of hops (3 to 6) & TCP version, except Vegas

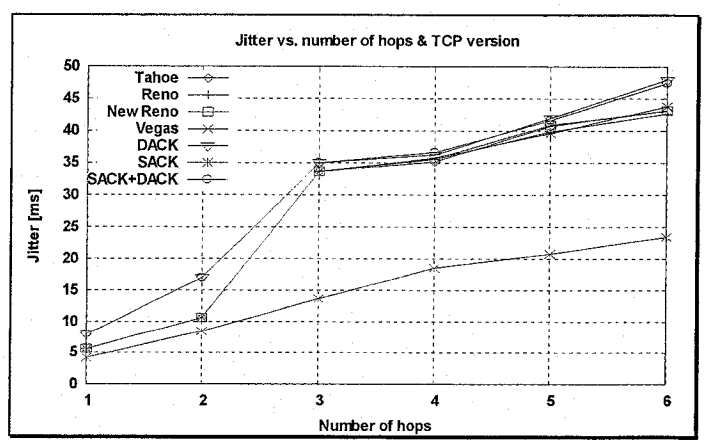


Figure 4.16 - Jitter vs. number of hops & TCP version

Interpretation:

Throughput:

Fig. 4.13 shows the throughput for different versions of TCP over a connection of one to six hops. Fig. 4.14 is actually a zoom-in of Fig. 4.13 on the region of 3 to 6 hops, so that the differences are clearer.

Some interesting points can be inferred from these two figures:

- Performance of DACK and SACK+DACK is almost the same all through the graph.

- From two hops onwards, Vegas is outperforming all the other versions, and as the hop count increases, this improvement gets even better. (30% improvement over Reno for a 6-hop connection).

- In a single-hop connection, Tahoe, Reno, New Reno, Vegas, and SACK are all performing exactly the same, and DACK and SACK+DACK are outperforming them by about 8%.

- The poorest performance seems to be that of Reno.

The out performance of TCP Vegas is apparently for its intelligent congestion control mechanism. As described in Sec. 3.2.4, this version of TCP adjusts its rate according to the amount of congestion in the network. If it *feels* that the network is becoming congested (using its *expected* and *actual* throughput measurements), it will decrease its congestion window accordingly.

Now in the case of our wireless scenarios, obviously the packet losses are not due to congestion, but due to wireless link errors (collisions for example). So even if a few packets get lost, Vegas, unlike other TCPs, doesn't suddenly decrease its congestion window, and as a result the throughput maintains its value and doesn't drop.

As for the poor performance of Reno, it is because Reno TCP doesn't have any Fast Recovery (Sec. 3.1.3). So when it encounters packet losses, and after it has retransmitted the lost packets, instead of going to Congestion Avoidance phase (like what New Reno or Vegas do), it will go to the Slow Start phase which is a sudden decrease in the congestion window (to 1 or 2 MSS), and thus causes huge drops in throughput.

Delay and Jitter:

Fig. 4.15 and 4.17 show the delay and jitter respectively, for different versions of TCP over a connection of one to six hops. Fig. 4.16 is actually a zoom-in of Fig. 4.15 on the region of 3 to 6 hops, and with the exclusion of Vegas TCP, so that the differences in that tight area are clearer.

The following points can be inferred from these figures:

- Clearly, TCP Vegas is performing excellent, both in terms of delay and jitter. On average, it is performing about 70% better than all other TCPs in delay, and about 50% in jitter.
- The other TCP versions have more or less the same performance in terms of delay and jitter.

The reason for this low delay and low jitter profile of Vegas, is that its robust congestion control algorithm prevents queuing at intermediate nodes and routers, by carefully matching the sending rate to the rate at which packets are successfully being drained by the network. Less queuing always translates to less delay and less delay variation.

Suggestions:

Based on the results above, it is recommended that the TCP Vegas be implemented (and be used as an option) in wireless ad hoc network mobile nodes' Operating System (OS).

There is a TCP Vegas implementation available for the Linux OS [45]. This implementation can be enabled, disabled, and configured, and has been incorporated into the official Linux 2.6.6 source release.

4.5 Experiment #5: Effect of Buffer Size

Motivation:

In the Network Simulator that I have used, every mobile node has a queue called Interface Queue (IFQ). It is implemented at the wireless sender side, and its location is between the Link Layer (LL) and the MAC layer. So, packets waiting to be transmitted on the wireless medium are queued in this buffer.

Since buffer size in every network has tremendous impact on the performance of the network (in network layer, and as a result in transport layer), this experiment is done to illustrate the impact it has on the performance at the transport layer of nodes in an ad hoc wireless scenario.

Traffic Pattern:

The traffic pattern used is the same as Exp. #1.

Variable Parameters:

Here the main variable parameter is the buffer size, which ranges from 1 to 50 packets. And as before, simulations are done for different number of hops.

Results:

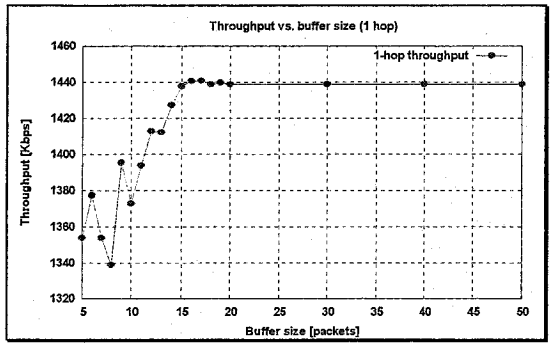


Figure 4.17 - Throughput vs. buffer size (1 hop)

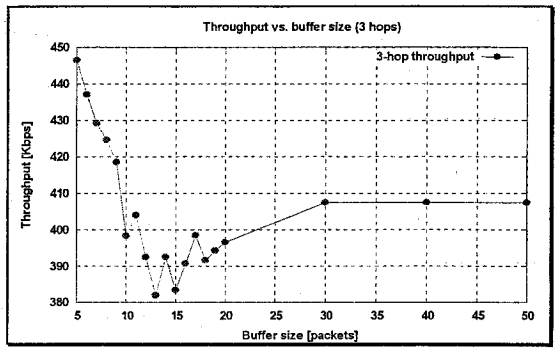


Figure 4.19 - Throughput vs. buffer size (3 hops)

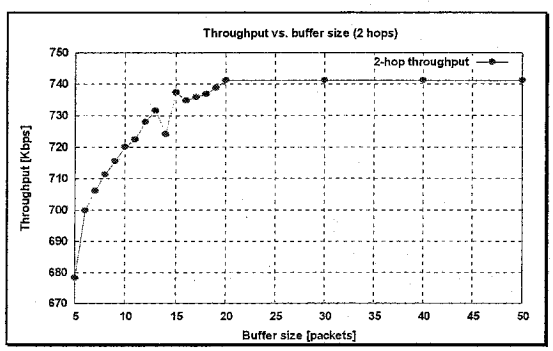


Figure 4.18 - Throughput vs. buffer size (2 hops)

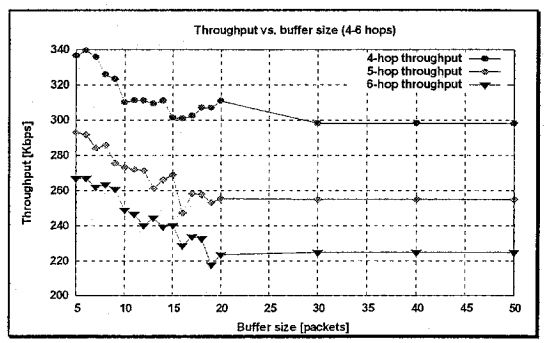


Figure 4.20 - Throughput vs. buffer size (4-6 hops)

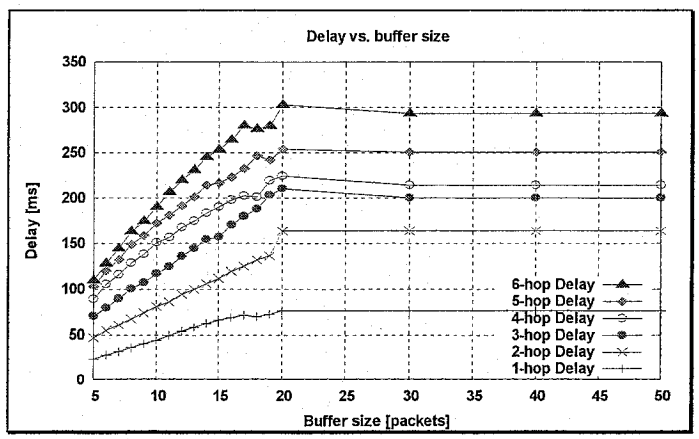


Figure 4.21 - Delay vs. buffer size

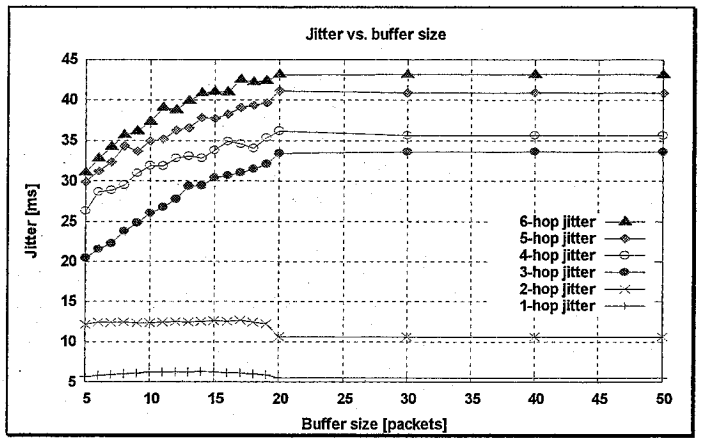


Figure 4.22 - Jitter vs. buffer size

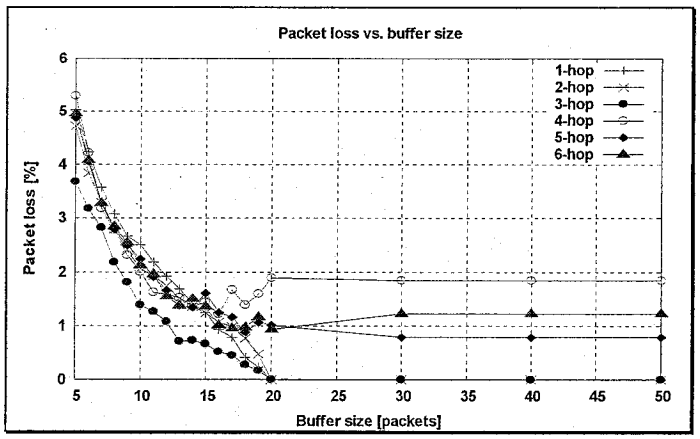


Figure 4.23 - Packet loss vs. buffer size

Interpretation:

Throughput:

For a one-hop and a two-hop connection, the throughput increases with larger buffer size (Figs. 4.18 and 4.19). In both cases, the maximum throughput is reached at around a buffer size of 20 packets, and remains constant at this value.

For a 3 to 6-hop connection though (Figs. 4.20 and 4.21), it is interesting to see that the throughput is decreasing as the buffer size grows. In these cases, the stability point is

reached at around a buffer size of 30 (for 3 and 4 hops), and a buffer size of 20 (for 5 and 6 hops).

An explanation for this is that, increasing the number of hops effectively increases the load on the network, because each packet has to travel a longer way, i.e. it has to pass more nodes to its destination, so the load on each node increases.

In this case, increasing the buffer size is acting counter-productive, since old packets are piling up on newer ones and finally causing the buffers to overflow, thus leading to a decrease in throughput.

While for a low-hop connection, the buffers usually contain a smaller number of packets (because of lighter load), so increasing the buffer size doesn't act like it does in the above explanation.

For comparison, the gain in throughput when going from a buffer size of 5 to 20, is about 6 percent (for a 1-hop connection). On the other hand, the drop in throughput when going from a buffer size of 5 to 20, is about 15 percent (for a 6-hop connection).

Delay and Jitter:

From Figs. 4.22 and 4.23 (for delay and jitter results, respectively), we can clearly see that both delay and jitter are increasing almost linearly with the buffer size getting larger, and that, again the stability point is reached at a buffer size of 20 packets.

This is an almost classic issue, bigger buffers in the networks (wired or wireless) increase the end-to-end delay and delay variance (jitter in our case). The reason is that, bigger buffer size means longer queue (in our case MAC transmission queue), and longer queue means more waiting time for the packets to pass through the interface. This longer waiting time translates to longer end-to-end delays.

Also, larger buffer sizes at each intermediate node, translates to a bigger variance in the number of packets in each buffer, and thus a bigger variance in the waiting times of packets in each buffer, resulting to a higher jitter value.

Packet Loss:

Fig. 4.24 shows that increasing the buffer size is having a positive effect on packet loss, i.e. lowering it. One can verify that:

- For a 1 to 3-hop connection, packet losses start at around 4-5% for a buffer size of 5, and all reduce to zero at a buffer size of 20.

- For a 4 to 6-hop connection, the stability point comes at a buffer size of 30 packets, and the packet loss at this stage is between 1-2%.

The reason for this trend (percentage of packet loss decreases as buffer size increases) is quite obvious: smaller buffers overflow (and thus drop excess packets) much faster than bigger buffers.

Suggestions:

From the overall (throughput, delay/jitter, packet loss) results of this experiment, very interesting points can be made.

There are some scenarios where it is to our benefit to reduce buffer size a little bit (and thereby sacrifice some packets), in order to get higher transport layer throughputs. For example, for a 3 to 6-hop scenario, and as mentioned in the throughput part above, although we know that reducing buffer size will result in a little more packet loss, but eventually more packets are able to "pass through" the network, and therefore higher throughputs could be achieved.

Also, if the application is delay and/or jitter sensitive, obviously the use of lower sized buffers will help reduce the delay and jitter of the received packets.

So again, based on these results, and on each specific scenario or application, one can make a trade-off between buffer size and throughput/delay/jitter/packet loss.

For example, if the lowest amount of packet loss is desirable, it is recommended that the buffer has at least 20 packets capacity.

Or as another example, if the connections cover only one or two hops (which is the majority of cases), a buffer size of at lease 20 is necessary for achieving a the maximum possible throughput.

4.6 Experiment #6: UDP Channel Capacity

Motivation:

UDP is a protocol which is commonly overlooked in performance analyses of wireless ad hoc connections. One reason for this is that it is a much less complicated protocol than TCP, and as a result there are few parameters which can play a role.

But still, since UDP has many applications, specially in real-time traffic (like streaming audio and video), and with the booming of wireless networks, it is for sure that it will be used extensively in these networks, and many aspects of it can be analyzed.

The purpose of this experiment is to measure what is the UDP bandwidth of a 2 Mbps wireless channel between two mobile nodes.

I call this *raw transport capacity* of the channel (for a single hop only, of course), since it shows what is the maximum amount of data throughput that can be achieved in the transport layer; without any kind of protocol overhead (like AKCs, retransmissions, waiting time for AKCs and all the other kinds of overhead associated with TCP).

Traffic Pattern:

In this scenario, Node 0 is transmitting to Node 1.

A Constant Bit Rate (CBR) traffic source is attached to Node 0, and it starts sending its stream to the transport agent (UDP) of Node 1 at 0 second and stops at 300 seconds.

The UDP packet size is 1480 bytes. (Again, with the added UDP header of 20 bytes, the network layer packet size will come to 1500 bytes)

Everything else is the same as Exp. #1.

Variable Parameters:

The only variable in this experiment is the load (bit rate) of the CBR source. It changes from 1 to 2 Mbps, and then the resulting UDP throughput is measured.

Results:

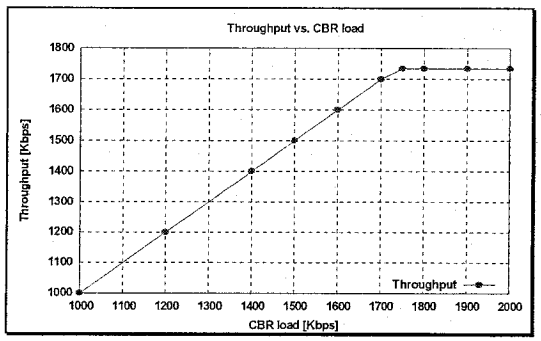


Figure 4.24 - Throughput vs. CBR load

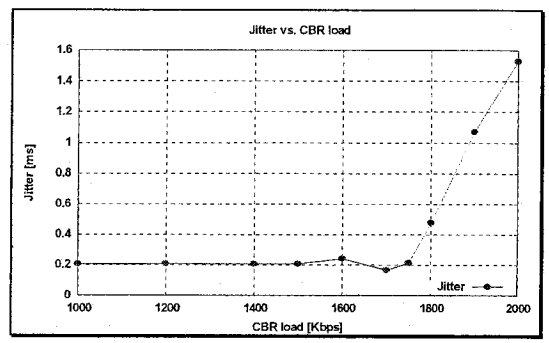


Figure 4.26 - Jitter vs. CBR load

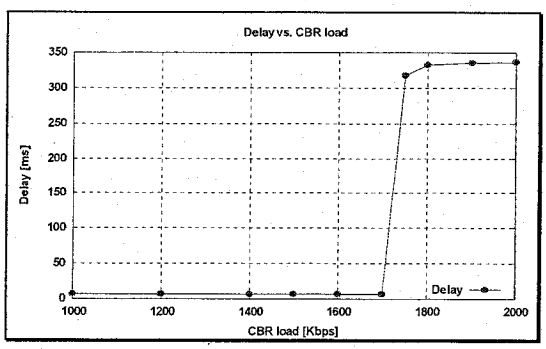


Figure 4.25 - Delay vs. CBR load

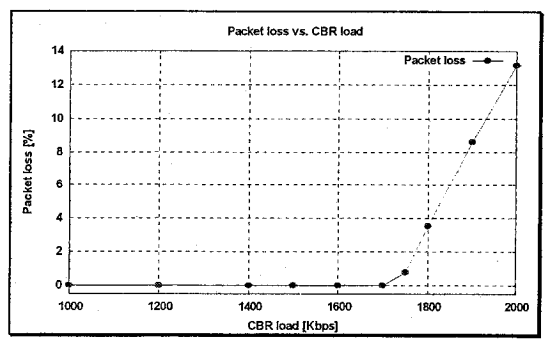


Figure 4.27 - Packet loss vs. CBR load

Interpretation:

Throughput:

From Fig. 4.25 we can see that the maximum *raw transport capacity* of our 2 Mbps channel is about 1730 Kbps (1.7 Mbps). It means that there is no way (using existing transport protocols, and the underlying link layer, MAC layer, and finally physical layer protocols) that we can inject more than 1.7 Mbps of data into the wireless channel.

It is interesting to see that below that ceiling, the capacity of the channel is exactly the same as what is input to it.

In other words, UDP allows the exact data rate of the source to be transferred to the destination (assuming there are no channel errors).

Also, please note that in TCP, even in a scenario like here, where there are only two nodes present, and there is no other load on the network, and there is only a single one-way connection between the two nodes, still collisions occur and cause a drop in throughput. The collisions occur because of the ACKs that are traveling on the same link but on the reverse direction.

But in case of a single UDP connection, like here, and assuming a perfect wireless channel (or a link and MAC layer which make the wireless channel look perfect to the upper layers), we can get out the exact rate that we input to the channel.

Delay and Jitter:

Figs. 4.26 and 4.27 show that, again the critical load is 1730 Kbps: when the load goes beyond that, delay and jitter start to increase rapidly.

This is because as soon as the load increases beyond 1.7 Mbps, packets which are generated at application layer (CBR), and delivered to transport layer (UDP), and also

received their routing directions at the network layer, cannot be delivered by the MAC layer (because the capacity of the channel is reached), so they get queued in MAC buffers waiting to be transmitted. It is this queuing which causes the delay (and jitter) to go up suddenly. Then, when the buffers get filled to capacity, delay remains constant (at a value of about 330 ms).

For loads below the capacity, the actual amount of delay and jitter is constant (6 ms, and 0.2 ms, respectively) and is very low.

Packet Loss:

Fig. 4.28 shows that packet loss starts to surge unbounded as soon as the load goes beyond 1.7 Mbps. This packet loss is because of packet drops in the MAC transmit buffers.

Because in UDP (unlike TCP) there is no feedback mechanism on the conditions of the channel, so the sending source (here our CBR) never knows how many and at what rate packets are being delivered, and it makes no change to its rate during the whole connection time. So the only way of getting rid of the excess packets (which cannot be transmitted because the capacity of the channel is reached), is to drop them at the MAC buffer of the sender side.

One can verify that the percentage of dropped packets is exactly equal to the extra percentage (to the channel capacity) of the load offered.

For example, when the input rate (load) is equal to 2000 Kbps, it is

2000 – 1730 = 270 Kbps, 270 / 2000 = 13.5%

over the channel capacity, so 13.5% of packets are dropped.

Suggestions:

The performance of UDP looks very attractive, both in terms of throughput and delay/jitter. Compared to TCP, its high throughput and low delay and jitter make it very attractive and suitable for real-time or interactive applications.

The price to pay for this, is unreliability. There is no guarantee that all packets will be delivered (dropped or lost packets are gone, no retransmission!), or that they will be delivered in order.

But a low percentage of packet loss is tolerated by such applications, for example in audio and video delivery.

4.7 Experiment #7: Comparison of TCP and UDP

Motivation:

It is very useful to get an insight into how these two protocols perform in comparison to each other, over an ad hoc multi hop wireless channel, keeping all the other variables constant.

In essence, this experiment is Exp. #1 repeated for the UDP protocol, and then compared with the results already obtained from Exp. #1.

Traffic Pattern:

There are two traffic patterns used: one is exactly the same as Exp. #1, the second is the same as Exp. #6, except that instead of having only two nodes, here we have 7 nodes, the source always being Node 0, and destination changing from Node 1 to Node 6.

Also, the CBR load is constant at 1.7 Mbps, so that it is below the capacity threshold of the first hop.

Variable Parameters:

Number of hops (from 1 to 6), transport protocol and traffic (FTP over TCP, CBR over UDP)

Results:

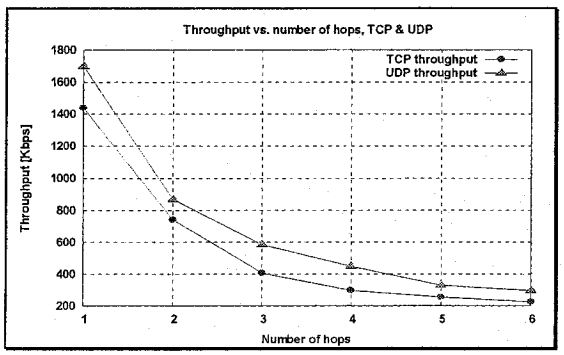


Figure 4.28 - Throughput vs. number of hops

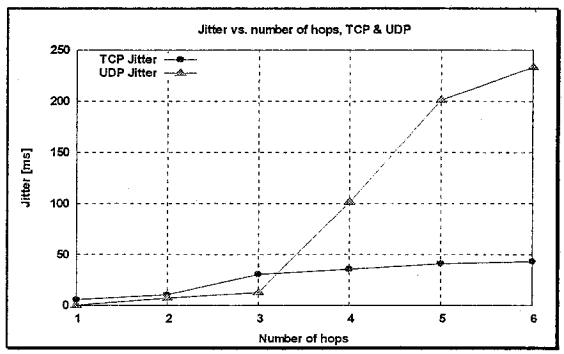


Figure 4.30 - Jitter vs. number of hops

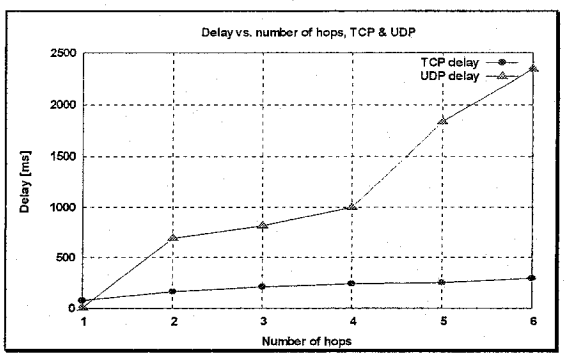


Figure 4.29 - Delay vs. number of hops

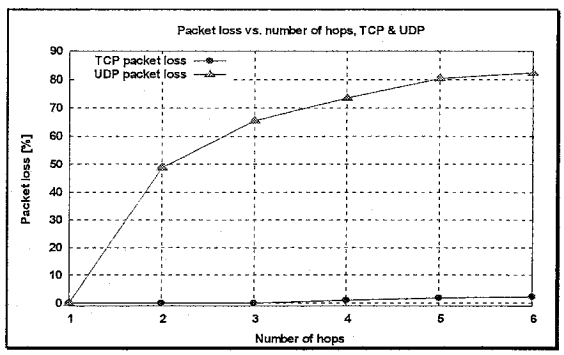


Figure 4.31 - Packet loss vs. number of hops

Interpretation:

Throughput:

When we look at the throughput (Fig. 4.29), everything seems in favor of the UDP connection: It yields throughput rates from 18% (for 1 hop) to 32% (for 6 hops) more than TCP. The reason for this high throughput is the same as the one stated in the throughput interpretation of the previous section (Sec. 4.6).

But if we take a deeper look, this is doing no good to our end-to-end connection. Because the bit rate of the source is 1.7 Mbps, which means packets ARE coming out of its UPD agent at this rate.

For 1 hop, there is no problem, we are receiving the exact same data rate (1700 Kbps), but advancing just one more hop, the receiving rate drops to about 870 Kbps, while the source rate is still at 1700 Kbps. Going more hops, this becomes worse, and in the last hop we are receiving only about 300 Kbps.

What has happened here is actually overloading the network, with unnecessary load. True that 1 hop can receive a maximum of 1.7 Mbps, but as the number of hops is increased beyond that, the throughput drops rapidly, and for the exact same reason that the TCP throughput started to drop after one hop in Exp. #1: MAC layer mechanisms.

Specifically, Node 1 cannot transmit (forward) packets to Node 2 at the exact same time that it is receiving some other packets from Node 0. MAC layer doesn't allow simultaneous transmission of frames on the medium, otherwise collision will happen between the incoming and outgoing packets (or frames in MAC terms).

Returning to our UDP connection, the throughput graph shows what is the maximum UDP capacity of a connection covering 1, 2, 3... hops. This is what I called *raw transport capacity* in the previous section, only here we can see this capacity for a bigger number of hops.

More on the usefulness and applications of this result, will follow in the Suggestions part of this section.

Delay and Jitter:

As for delay (Fig. 4.30), the graph is showing a very meaningful result: For 1 hop, because the sending rate is below the channel capacity, delay is only 6 ms for UPD, as opposed to 76 ms for TCP. We expected this, as it was also measured and explained in Exp. #6. But as we advance only one hop, and then up to 6 hops, not only the delay increases, but it even goes much higher than TCP delay (about 10 times longer!).

This is the same effect as what happened to the delay in Exp. #6 as we increased the CBR load beyond the 1-hop transport capacity of the channel. Only here, the CBR load is higher than the transport capacity of two hops (or more).

As a result, network overloading happens, causing the intermediate buffers to fill up and cause such long delays, and jitters. (The same discussion holds for jitter, Fig. 4.31)

Packet Loss:

Fig. 4.32 shows the percentage of packet loss for both UDP and TCP connections. Following the above explanations, it is obvious why packet loss is going up so rapidly for the UDP connection. All the packets generated at the source, which cannot pass through the channel, are dropped. Some right at the beginning of their trip, i.e. in Node 0, some at other intermediate nodes.

As for TCP, the packet loss rate remains below 2 percent, and this is only caused by packet losses on the wireless channel due to collisions, and not due to buffer overflow.

Suggestions:

UDP has some very attractive features, specially over wireless channels, which make it an excellent candidate for applications requiring those features, e.g. low complexity, low overhead, low delay, and higher throughput.

But the point is that, since it's a passive protocol (meaning there is no feedback to the sender, so there is no rate adjustment), it can easily overload a network if not used with care.

In our multi hop case, when considering the use of UDP, one should estimate the average number of hops that the packets would travel to reach their destination. Based on that, he should adjust the source's sending rate according to what the longest-hop connection can handle. Because if the rate is more than that, we'll encounter all the negative effects discussed above, namely: network overloading, buffer overflowing, long delays and high jitter, and packet loss.

For example, imagine in the 6-hop scenario, Node 0 wants to transmit a UDP stream to Node 6, so we have a 6-hop connection, and according to the throughput results of this section (Fig. 4.29), the UDP capacity of a 6-hop connection is a maximum of about 290 Kbps. So the sending source should adjust its application sending rate so that its output is no more than 290 Kbps (for example, by compressing the audio or video stream), otherwise the available resources (wireless medium, buffer space, ...) are wasted.

4.8 Experiment #8: Using Optimum UDP Load

Motivation:

The results of the previous experiment, encourages us to see how the network will perform under a more realistic CBR/UDP load. So in this experiment I reduce the load to a value which can be handled by the last node in the chain, and see how the network will perform under these optimum conditions.

Traffic Pattern:

A single UDP connection is established between Node 0 and other nodes in the chain. The CBR traffic source is outputting packets at a bit rate of 290 Kbps, so that it is below the capacity threshold of the longest number of hops (6).

Variable Parameters:

The only parameter change during this experiment is the receiving node, or in other words the hop count.

Results:

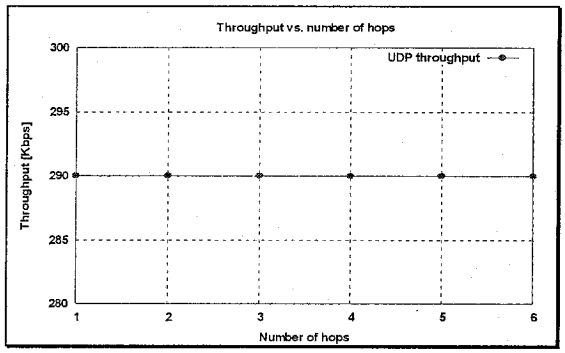


Figure 4.32 - Throughput vs. number of hops

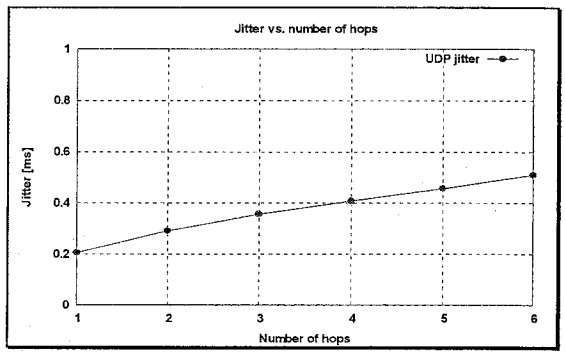


Figure 4.34 - Jitter vs. number of hops

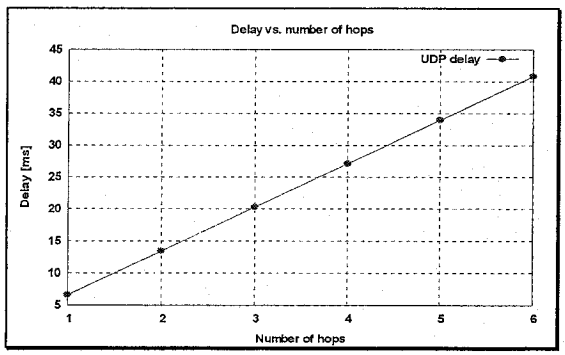


Figure 4.33 - Delay vs. number of hops

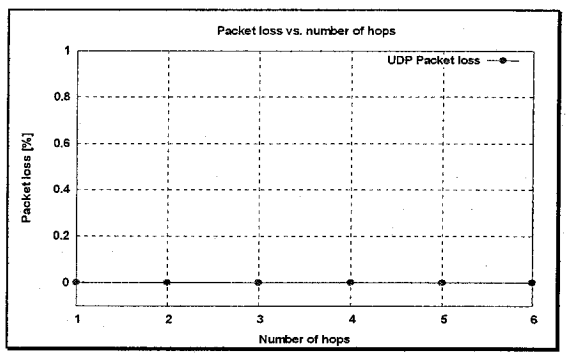


Figure 4.35 - Packet loss vs. number of hops

Interpretation:

Throughput:

From the throughput graph (Fig. 4.33) it is clear that the network is operating at its optimum situation. The bit rate is kept constant throughout all the nodes, and is equal to the source's sending rate: 290 Kbps.

This was somehow predictable, because we already knew what is the 6-hop *raw transport capacity* of the network, and adjusted the CBR load at exactly this capacity.

Delay and Jitter:

Delay (Fig. 4.34) is increasing linearly with the number of hops, with a minimum one-hop delay of 6 ms and a six-hop delay of only 40 ms. Compare this to the TCP six-hop delay of 290 ms! (Sec. 4.1)

We could say that jitter (Fig. 4.35) is almost constant, ranging only between 0.2 and 0.5 ms, which is a really negligible jitter for any type of application. Again, compare this to a 43 ms delay of a six-hop TCP connection.

Packet Drop:

It is zero, no packet drops at all, even for a six-hop connection! (Fig. 4.36)

This indicates that there are not even any collisions happening, although intermediate nodes are receiving and forwarding packets from and to their neighbors. This is simply because a load of 290 Kbps is low enough from the MAC layer perspective, that the scheduling of contention for the wireless channel is done perfectly.

Please note that this does not hold for a TCP connection of the same last-hop rate (about 230 Kbps), and because of the ACK traveling on the reverse direction, there are a few packet losses happening. (Fig. 4.32, about 2.2% for the six-hop connection)

Of course this scenario is a very controlled one, and may hardly occur in reality: there are no wireless bit/packet errors, and there is no other connection going on between mobile nodes. But as a reference, this gives us a deep inside into the workings of UDP protocol and its interaction with the wireless MAC protocol.

Suggestions:

Here we just witnessed that operating a balanced UDP connection over a multi-hop wireless network, gives us very good results, in terms of all the four performance metrics. Please also refer to the suggestions made in Sec. 4.7.

4.9 Experiment #9: Effect of Packet Errors on TCP

Motivations:

Most of the time, in TCP/UDP performance analyses over wireless channels, it is assumed that the wireless channel is error free, the only packet errors being the ones created by collisions of MAC on the channel.

The basis of this assumption is that "the seven retransmissions provided by 802.11 are sufficient in order to recover from typical channel errors" [23].

This is true, since the IEEE 802.11 MAC error detection and retransmission mechanism doesn't deliver any erroneous packets to the upper (LLC) layer. If the MAC receiver detects that the received frame contains errors, it doesn't send back an ACK to the sender, so the sender (after waiting for an ACK_Timeout period, and executing the backoff procedure), retransmits the last unacknowledged frame [46].

But, the number of retransmissions are limited (typically 7), after which the packet is dropped from the sender's transmit queue [8].

So, one can assume that any transmitted packet by a sender is either delivered error-free to the receiver, or dropped and not delivered at all.

Now back to the discussion of assuming an error-free wireless channel, and the basis for this assumption; although the assumption is true, but the way is still open to investigate, e.g. what would be the effect of all these MAC layer retransmissions and drops to the eventual performance of the upper (specifically TCP/UPD) layers?

This gave me the motivation to introduce (inject) packet errors on the channel and analyze the performance of the resulting network.

Traffic Pattern:

The traffic pattern used is the same as Exp. #1.

Variable Parameters:

As before, the number of hops, which range from 1 to 6, and newly introduced for this sections, the PER (Packet Error Rate) which ranges from 1 to 20 percent.

The model (in ns-2) used for injecting packet errors on the wireless channel, is a very simple one: A uniformly distributed random variable (in the range 0 to 1) is used to generate errors; and the error rate is the input parameter to this procedure.

Results:

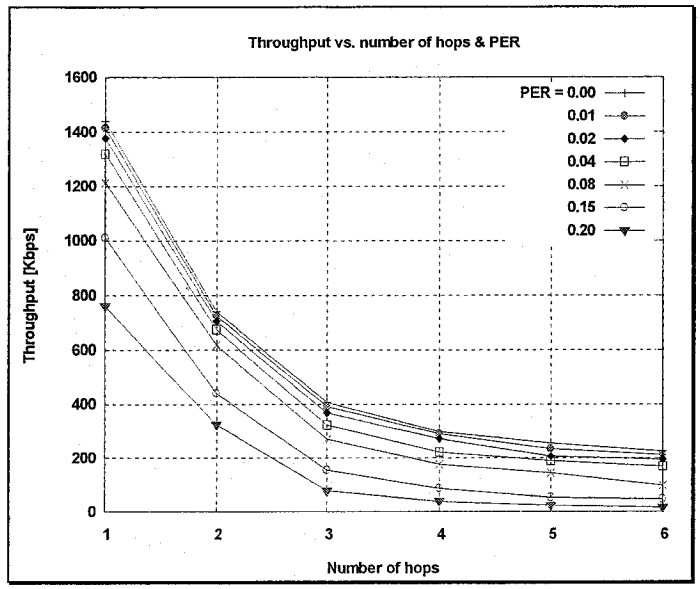


Figure 4.36 - Throughput vs. number of hops & PER

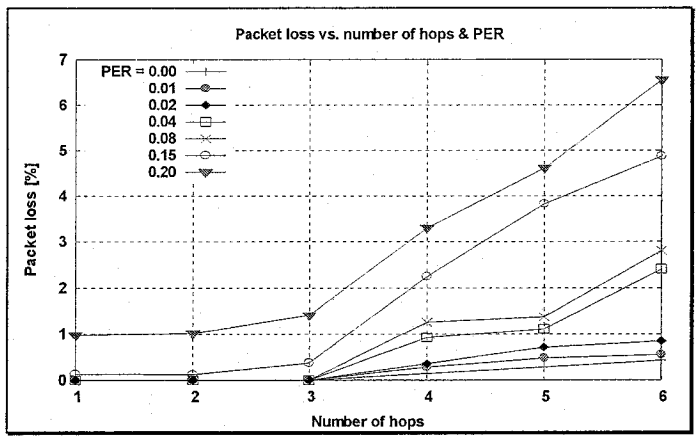


Figure 4.37 - Packet loss vs. number of hops & PER

Interpretation:

Throughout:

In Fig. 4.37, we can see the 1 to 6-hop throughput of the network for different PERs, where in each case the PER almost doubles compared to the previous one.

It is interesting to see how, despite the error handling of the MAC layer and its packet retransmissions in case of errors, the TCP throughput drops as the packet error rate increases. There are a few points worth mentioning here:

- The drop in throughput is more noticeable in high hop count connections compared to the shorter ones. For example, throughput drops about 28% in a single-hop connection, when going from an error rate of 1 percent to 15 percent. However, for a six-hop chain, the drop is about 77% for those same error rates.

This shows that the longer the path (hop count) of a multi-hop connection, the more vulnerable it will be to errors.

- At higher error rates, the throughput drops more rapidly than it does at lower error rates. Foe example, in a 1-hop connection, doubling the error rate from 4 percent to 8 percent, causes a decrease in throughput of about 8%, while (almost) doubling the error rate from 8 percent to 15 percent, causes a decrease in throughput of about 16%.

This shows how vulnerable the network is to high error rates (for example bursts of errors caused by interference).

- Despite all this, this experiment shows us the amazing power of IEEE 802.11 MAC layer error handling scheme in presence of errors. A packet error rate of 20 percent is actually a *very* high error rate, but still we can see that for a one-hop connection, the TCP is able to maintain its throughput to as much as half of the value it has for an error-free (PER=0) medium.

- As to why the TCP throughput decreases even though the MAC layer never delivers an erroneous packet to its upper layer, there are several reasons:

1. Packet (data frame in MAC terms) retransmissions take time, of course. Although the time is not as long as TCP packet retransmissions, because it is performed over only one hop, while TCP retransmissions are performed over the entire length of the path.
2. Before the node decides to retransmit the packet, it should wait an amount ACK_Timeout (equal to an ACK frame time, plus a SIFS). These waiting times reduce the throughput.
3. The number of retransmission is limited (usually 7 times), and after this number is exhausted, the packet is dropped from the node's transmit queue. This drop causes a waste of bandwidth and time, and again reduces throughput.
4. The above three reasons could affect TCP data packets, *and* TCP ACK packets as well. So the loss (or delay) of a TCP ACK also adversely affects the TCP throughput.

Packet Loss:

Everything is clear from the packet loss graph (Fig. 4.38).

Packet loss increases with number of hops (as we already knew), and it increases with PER as well (as we expected).

Note however that there is no packet loss for a connection covering 1 to 3 three hops when the PER is below 8 percent, which means that MAC is able to retransmit all erroneous packets within its 7 retransmission attempts.

Also note that packet losses are increasing very sharply for higher than 3 hops and PERs above 4 percent.

Almost all packet losses in this case are caused by:

1. MAC buffer overflows, because of too many packets waiting to be retransmitted, so there is no space for newly arrived packets; or
2. MAC itself dropping packets after it is not able to get an ACK after 7 retransmissions over the channel.

4.10 Experiment #10: Effect of Packet Errors on UDP

Motivation:

To study the effect of channel errors on the performance of UDP protocol in wireless ad hoc networks. Please refer to Sec. 4.9 for further details on motivation.

Traffic Pattern:

The traffic pattern used is the same as Exp. #1, except that the UDP protocol is used instead of TCP. Please refer to Sec. 4.6 for further details on the traffic.

The CBR source's load is 290 Kbps, as explained in Sec. 4.8.

Variable Parameters:

Number of hops, channel Packet Error Rate (PER).

Please refer to Sec. 4.9 for further details on the error model used.

Results:

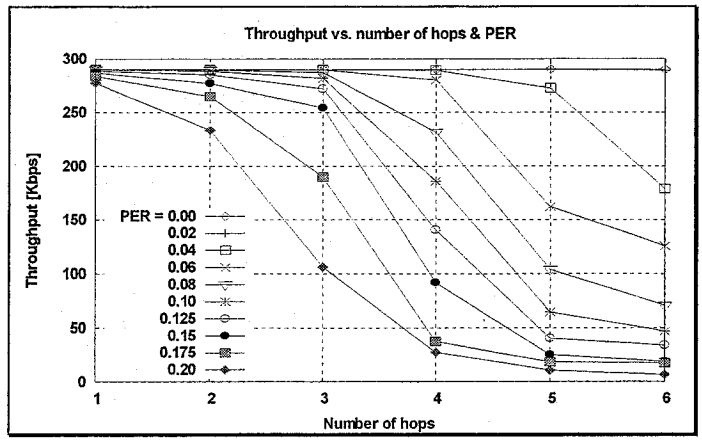


Figure 4.38 - Throughput vs. number of hops & PER

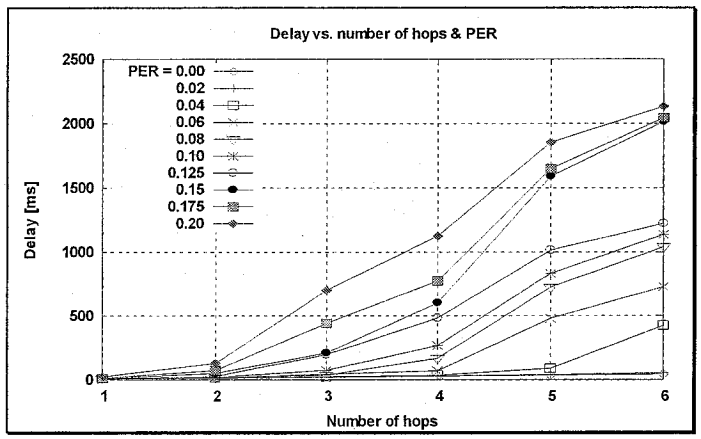


Figure 4.39 - Delay vs. number of hops & PER

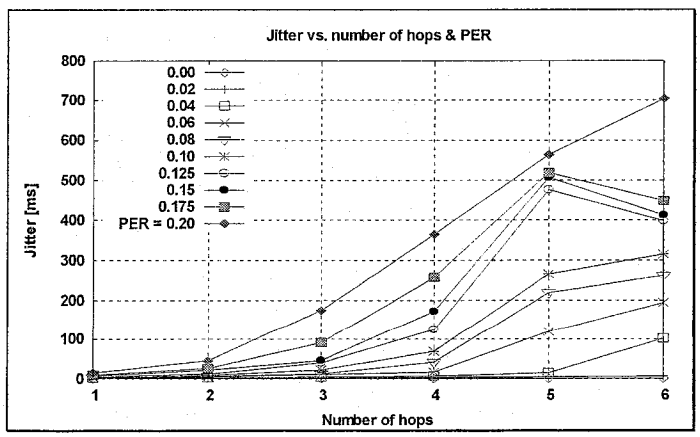


Figure 4.40 - Jitter vs. number of hops & PER

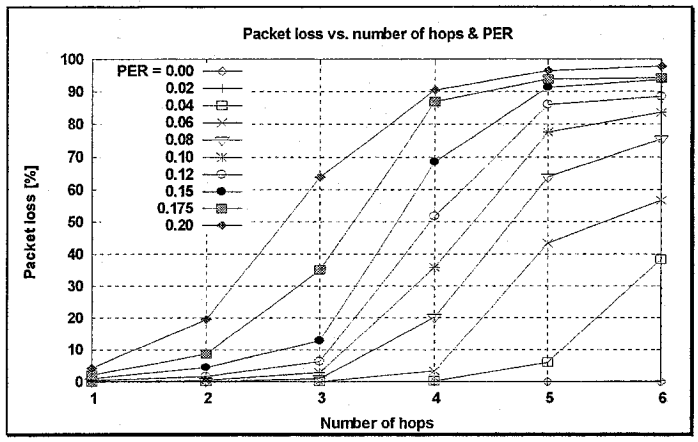


Figure 4.41 - Packet loss vs. number of hops & PER

Interpretation:

Throughput:

Fig. 4.39 shows how the UPD throughput drops, as the number of hops and PER increase.

Note that the throughput maintains its value constant at 290 Kbps over the 6-hop chain in an error-free medium, as discussed in Sec. 4.8, and shown in Fig. 4.33. As a result, throughput starts to deviate from its constant 290 Kbps value as the error rate increases. This is somehow different from the TCP case (Exp. #10, previous Section) where even for an error-free medium, the throughput was decreasing as the number of hops increased.

Here, because the source node's sending rate is relatively low, the reaction of the network to packet errors is different from what it was for the TCP case.

We can observe the following trends in the graph:

- For PERs below 2 percent, there is almost no drop in throughput for the whole span of hops.

- For a PER of 4 percent, the throughput starts to decrease from a hop count of 4, 6 percent from 3, 8 percent from 2, and for PERs of more than 8 percent the decrease starts from a connection with a single hop.

- As the case for TCP, the throughput drop is much more rapid in high hop count connections (5, 6 hops), than it is for low hop ones. For example, in a 6-hop connection, going from 0 PER to 6% PER, the throughput drops more than 50%; while in a 4-hop connection, for those same error rates, the throughput decreases only about 4%.

- The reason for throughput drops in UDP case is quite different form that of TCP case (the three reasons stated in Sec. 4.9 > Interpretation > Throughput). Here, the only reason for decreased throughput is because less number of packets are received in each case, and this in turn is because of more packets have been lost (dropped) and have not been able to make it to the destination UDP node. There are no TCP rate adjustment mechanisms involved, and there are no ACK losses to adversely affect the UDP connection.

Delay:

Very interesting results can be obtained from the delay plot (Fig. 4.40). What we expected was a slight (linear) increase in delay as the hop count of connection increases, and as the PER increases. But instead, the delay increased up to a value of 2000 ms!

For a better insight on this issue, note the following graph (Fig. 4.43) which is a zoom-in of Fig. 4.40 in the delay range of 0 to 200 ms.

This plot shows how fast (almost vertically) the delay is increasing as the PER goes beyond 4 percent.

The reason for this trend is the effect of these three events, combined:

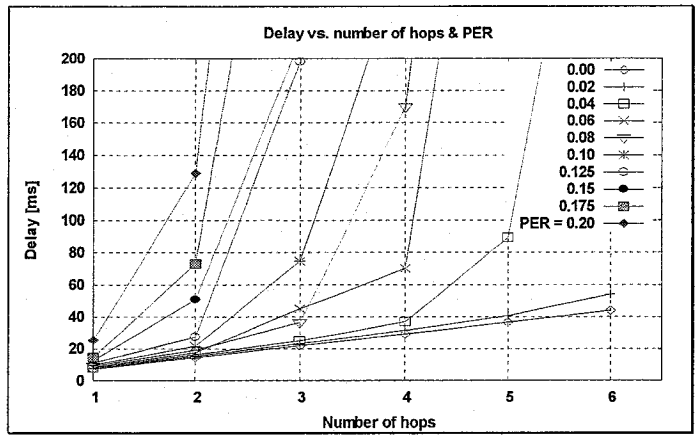


Figure 4.42 - Zoom-in of Fig. 4.40

1. As we know, UDP has no flow/congestion control algorithm, so it just keeps sending the packets to the network, no matter how they are delivered to their destination.

2. Like the case for TCP, because of high packet error in wireless medium, MAC frequently incorporates its retransmission mechanism, to make for packets for which is hasn't received an ACK from the next-hop MAC. As a result, transmit queues get longer and MAC buffers get filled with packets waiting to be retransmitted.

3. The sending node's UPD agent, unaware of what's happening because of no feedback mechanism from the receiver, keeps sending its packets to the network and populates MAC buffers even more.

As a result of all this, the average number of packets in intermediate buffers grows, and this is exactly what creates those long delays.

We investigated a similar trend in Exp. #7, comparing that with the results of this Section tells us that, *when considering delays*, increased packet errors has the same effect on the network as overloading it with UDP traffic.

The same trend and discussion holds also for jitter performance (Fig. 4.41).

It is worth mentioning here that, this effect (enormous delays in presence of packet error) was not observed with TCP connections. Although delay performance of TCP connection in presence of noise was not presented graphically in Sec. 4.9 (because no specific characteristic was noticeable), but our numerical results show that the delay was only slightly increased in presence of packet errors, and was bounded between a range of about 100 ms (for 1 hop) to 300 ms (for 6 hops).

The reason is TCP's feedback mechanism through flow/congestion control algorithms: TCP senses that (in high PERs) packets are not delivered as they should be to their destination, thus decreasing its sending rate, and therefore it does not cause the intermediate nodes' buffers to get filled up. So delay remains at more realistic levels.

Packet Loss:

Fig. 4.42 shows the graph of packet loss versus hop count and PER. As can easily be seen from the figure, packet loss increases rapidly with the hop count of connection, and with raising PER.

The reason for packet losses is the same as the case for TCP, which is explained in Sec. 4.9 > Results > Packet Loss.

But there is a difference between the packet loss experienced by the UDP network and that experienced by the TCP network. Here, for the UDP case, we can see that packet losses are growing very fast, and reaching to percentages as high as 97; While in TCP (Fig. 4.38), the maximum loss was only about 7%. The reason again is TCP's flow control mechanism, and UDP's lack of such a feedback mechanism. As was discussed above in the last paragraph of Delay discussion, UDP keeps injecting packets to the network, even when the intermediate buffers are occupied with packets waiting to be

retransmitted. So buffers start to overflow after a certain amount of time passes, and most newly arrived packets start to drop. This drop increases with higher PERs because when the error rate is high, more packets get corrupt in the wireless medium and need to be retransmitted, so more of the MAC buffer spaces become occupied, and as a result they start to overflow faster and drop packets at a higher rate.

In the Throughput analysis of this section (last paragraph), it was mentioned that the decrease in throughput is caused by packets being lost (dropped), here we can verify it visually, and numerically.

By looking at Figs. 4.39 and 4.42, we can say that they kind of 'complement' each other, which means that one can derive one from the other. To make this clearer, take the 10% packet loss line, and consider a connection covering 4 hops. Packet loss value at this point is about 36%. Now from the throughput graph, throughput at this point is 186 Kbps, which is exactly 36% below the no-error rate of 290 Kbps.

This clearly shows us that the drop in throughput is caused solely by packet losses at the MAC layer.

Suggestions:

Comparing the performance of UDP and TCP in presence of errors, we can say that UDP performance (in terms of our three main performance measures) is enormously affected and degraded in such conditions. Specially, packet delay/jitter and packet loss are both beyond accepted limits, when PER is higher than 2 percent.

Also, enough attention and exact measurements/simulations should be done when considering implementation of ad hoc networks with high hop counts in noisy mediums,

since, as was mentioned in the Interpretation part, performance deteriorates much rapidly in connections involving 4 or 6 hops.

So these kinds of setups should either be avoided (e.g. using the strategy stated in Sec. 4.1), or if inevitable, every aspect of their performance should be measured and predicted, so that users located several hops away from the other side of the onnection, are served as good as the other users.

4.11 Experiment #11: Packet Errors and TCP Versions

Motivation:

As it was mentioned in Sec. 4.4, most studies on comparison of different versions of TCP are either for wired links, or for wireless links which are assumed error-free [39, 40].

This motivates us to analyze and compare the performance of a few TCP flavors on multi hop links with the presence of packet errors.

Traffic Pattern:

In one scenario, there are only two nodes (1 hop), one sending TCP traffic and one receiving it. In the other scenario, there are seven nodes and Node 0 is sending to Node 6 (6 hops). TCP traffic characteristics are same as before (Sec. 4.1).

Variable Parameters:

Number of hops, TCP versions

Results:

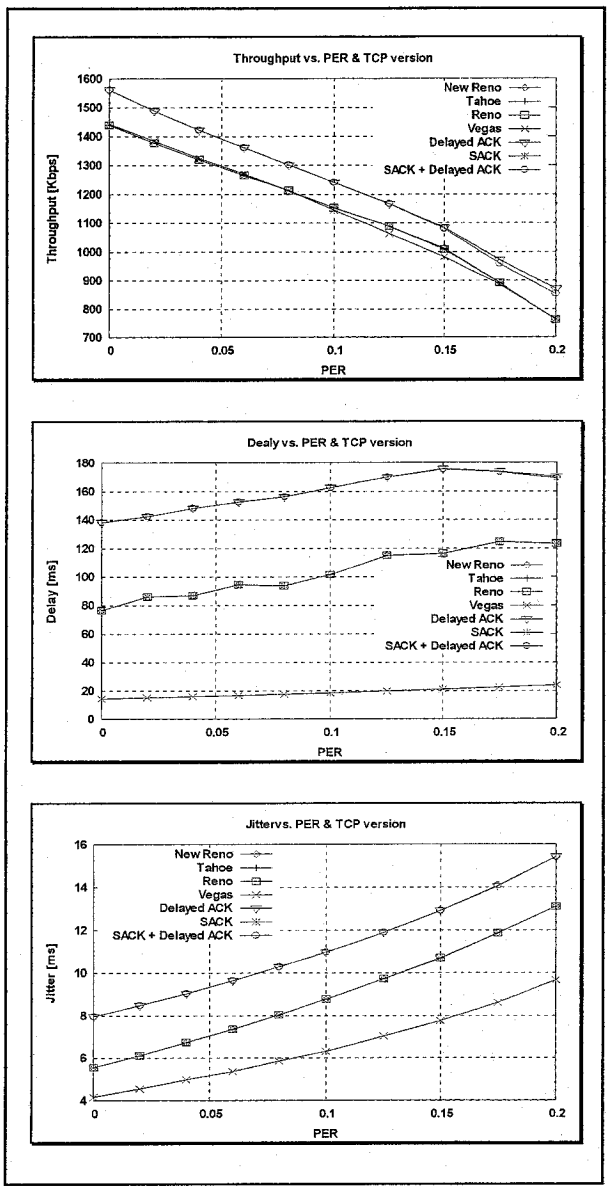


Figure 4.44 - Performance for 1-hop connection

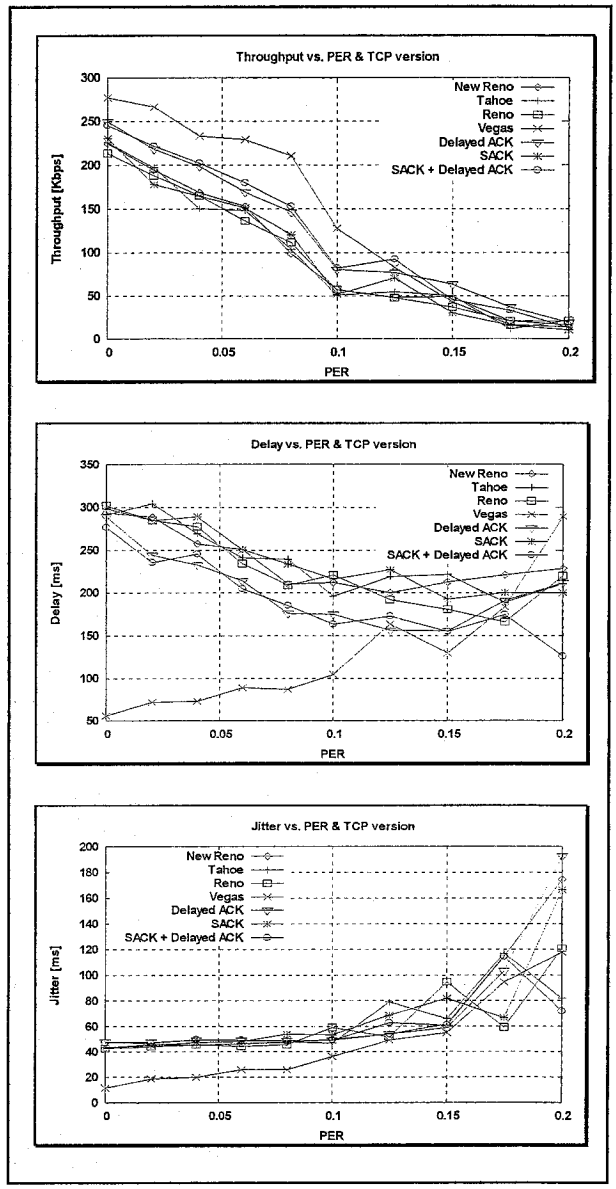


Figure 4.43 - Performance for 6-hop connection

Interpretation:

- One-hop connection (Fig. 4.45):

Throughput:

DACK (and SACK+DACK) is outperforming all the other versions, over all the PERs.

This is because of the delayed ACK mechanism, where an ACK packet is sent for every two segments, instead of every segment. Only if the next segment doesn't arrive within 200 ms, and ACK is sent without further waiting. This actually halves the number of ACKs flying on the link, and as a result fewer collisions happen, which in turn causes the throughput to increase.

All the other versions are operating at pretty much the same throughput.

Delay and Jitter:

Both delay and jitter are increasing with the hop count, though the increase in jitter is more noticeable than the increase in delay.

In both graphs, there are three separate traces:

The best-performing one (lowest delay/jitter) is Vegas, the worst is DACK (either alone, or in combination with SACK), and in the middle are the rest (Reno, Tahoe, New Reno).

It is very interesting to see that TCP Vegas is performing so well in terms of delay (on average, 75% better than New Reno or Tahoe) and jitter (on average, 30% better than New Reno or Tahoe).

- Six-hop connection (Fig. 4.44):

Throughout:

As we can see from the Throughput graph, in packet error rates of below 10 percent, TCP Vegas is performing well above all other versions, at times more than 50% better. (Refer to Sec. 4.4 for an explanation of this behavior).

Just below Vegas, comes DACK. There is no obvious distinction among other flavors.

Note that in high PERs (above 13 percent), it is DACK which performing the best, even better than Vegas. Because in such high-error media, the number of corrupt packets is so

high, that the fewer number of TCP ACKs flying in the reverse direction helps a lot, by reducing collision between ACK and data packets, and also reducing the number of unnecessary retransmission attempts/delays associated with ACKs.

Delay and Jitter:

It is natural to expect that, as it was the case for UDP, delay will increase as error rates grow. But what the delay graph is showing us is exactly the reverse: delays for all TCP versions (except Vegas) are decreasing with increasing PER, from a value of about 300 ms to a value of about 200 ms.

Further investigation reveals the reason behind this trend: It's all because of TCP's dynamic window adjustment mechanism (congestion control). As the packet errors increase, more packets can't make it to their destination, and less ACKs are sent back to the TCP sender. So the sender, incorporating its congestion control algorithm, reduces the size of its sending window (i.e. congestion window) and as a result, the throughput decreases. (This is well shown in Fig. 4.44, Throughput graph)

Now when the throughput is reduced, less packets are sent, and there is less occupation in intermediate nodes' MAC buffers. And as we know, fewer numbers of packets in buffers always means shorter delays.

There is no noticeable difference in delay performance between different TCP versions.

As for low delay of Vegas TCP, the reason was explained in Delay and Jitter analysis of Sec. 4.4.

Jitter is staying almost constant for all PERs below 10 percent (again because of low buffer occupancy), but it grows in higher PERs because more packets need to be retransmitted, and hence they'll arrive at the destination with larger variations in their

delay. (Note that although delay itself is not high, jitter which is delay variations can be high)

Again Vegas is performing well below 40 ms jitter, though it starts to grow over this value at PERs higher than 10 percent.

Suggestions:

The Delayed ACK option of TCP should always be used in wireless networks, since it boosts throughput and has no negative impact on other aspects of the network.

It's called *option* because it is an optional feature of TCP, and in almost all implementations the user can either turn it on or off.

TCP Vegas is a very suitable version of TCP for noisy mediums where there are lots of link errors which cause packet losses. Its use specially over multi hop networks is highly recommended, because as it was presented in this section, it is performing excellent in presence of errors, both in terms of throughput and packet delay/jitter.

4.12 Experiment #12: Simultaneous TCP Connections

Motivation:

Up to now, the scenarios were not probably very realistic because only one pair of nodes (source, destination) was involved in a single TCP connection.

Moving towards a more realistic situation, in this section we try to evaluate the performance of our ad hoc network when more than one TCP connection is set up between more than two nodes.

Traffic Pattern:

TCP connections (with the same characteristics as before) are established between more than two nodes, and in different directions. The exact pattern in each case is described in the respective Results part.

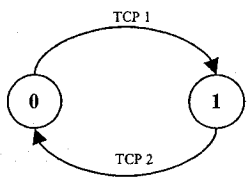
Variable Parameters:

No parameters are involved in this experiment, except that connections are set up between different nodes. Node numbers are as described in Sec. 4.1, starting from 0 and going up to 6.

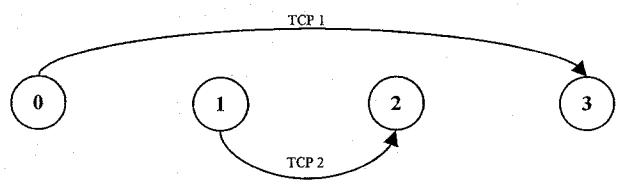
Results:

In the diagrams below, different setups for this experiment are shown, each with a setup case number (Case #). Numbers inside the circles are node numbers, arrows show the directions of the connection (tail: source, head: destination), and labels on the arrows show connection numbers. The performance results (throughput, delay) of each setup are presented in the table that follows the diagrams.

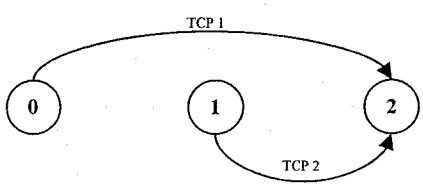
Case #1:



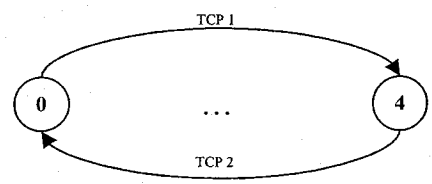
Case #2:



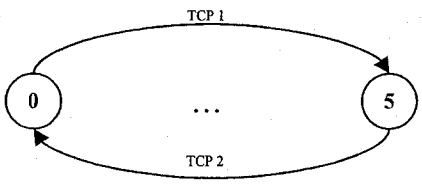
Case #3:



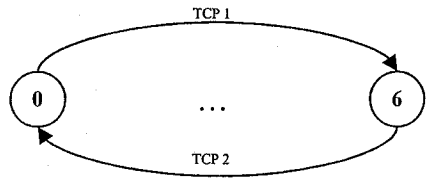
Case #4:



Case #5:



Case #6:



	TCP 1		TCP2	
	Throughput [Kbps]	Delay [ms]	Throughput [Kbps]	Delay [ms]
Case #1	712	171	712	155
Case #2	248	409	727	203
Case #3	378	294	713	276
Case #4	208	231	101	222
Case #5	211	280	38	363
Case #6	185	317	41	434

Interpretation:

Case #1:

Node 0 and Node 1 are sending to and receiving from each other at the same time. Throughput is 712 Kbps in both directions, which is almost half of a single one-hop connection (1438 Kbps, Exp. #1). This shows that the wireless medium is shared quite fairly between the two connections, and that a wireless node cannot send and receive at

the same time. Packets which are to be sent, contend for the exact same medium resources that packets which are being received contend for.

And this is because the two nodes are within transmission range of each other, so there is no *spatial reuse* [23] possible.

It is interesting to see that the throughput in this scenario is a little *less* than a single two-hop connection (741 Kbps, Exp. #1). This is because in the double one-hop connection (here, Case #1), in addition to all the packets which are present in the single two-hop connection (i.e. packets in the shared wireless medium between Node 0 and Node 1), there is one additional packet type: ACKs traveling from Node 0 to Node 1. There is no such packet in the single two-hop connection. This is what causes the slight difference in throughput in these two scenarios.

Case #2:

In this setup, we can see what happens if there is a 3-hop connection between two nodes (0, 3), while there is another connection going on between the two intermediate nodes (1, 2) which are supposed to forward the packets of the first connection.

From Exp. #1 we know that the throughput of a 1-hop connection is 1438 Kbps, and that of a 3-hop connection is 407 Kbps.

What's happened here is again an almost-fair allocation of the medium to the two connections: throughput for 0-3 connection is 248 Kbps, while that of 1-2 connection is 727 Kbps.

Here the 3-hop connection is acting like it were a 5-hop connection.

That single one-hop intermediate connection has the effect of increasing hop count to two more hops.

Case #3:

Here, Node 2 is receiving from two sources: 0 and 1. Thus, again, the second connection (any of the two can be considered as the second), is operating as if the source and destination were one more hop away from each other:

713 Kbps is close to 741 Kbps of a single 2-hop connection, 378 Kbps is close to 407 Kbps of a 3-hop connection.

Case #4-6:

These three setups are similar to each other, the only difference being the second node which is either Node 4, 5, or 6. The first node is always Node 0.

The pairs (0, 4), (0, 5), and (0, 6) are sending to and receiving from each other simultaneously. Of course, in between, the rest of the nodes are forwarding their packets. But here, unlike when two adjacent nodes were communicating to each other in Case #1, the bandwidth share is not fair, always one direction of the connection is winning bandwidth over the other. For example, 0 to 6 connection has been able to capture a bandwidth of about 185 Kbps, while 6 to 0 connection has got only 41 Kbps.

The reason is just the randomness nature of the MAC 802.11 scheduling. Because here there are several nodes (e.g. 5 nodes in Case #6) involved in forwarding packets (data and ACK, and in both directions), the combined effect of all contentions for the medium, has been like this, i.e. one connection being able to reach a higher throughput than the other, although everything is symmetrical in this scenario.

Suggestions:

From the above results and explanations, we can infer that loading the network with TCP connections between different source/destination pairs, affects the network the same way as if the number of hops for each connection were increased.

This increase in the hop count is almost equal to 1 hop for each additional *forwarding* node which itself is sending or receiving packets for another connection.

As a result, sometimes rerouting the path through a different set of nodes, where fewer connections are involved in the intermediate nodes, and even though the new route may cover more hops, can help achieving a better performance.

4.13 Experiment #13: Simultaneous TCP and UDP Connections

Motivation:

In Sec. 4.12 we tested the effect of two simultaneous TCP connections. Here one of those connections is changed to a UDP connection.

For a wireless ad hoc network, UDP connections are always considered as *loads* on the network, because by constantly injecting packets to the network, they impose a load or a weight on the ongoing TCP connections in the same region.

So here we try to investigate what is the effect of loading the network with a UDP connection with variable rates.

Network Topology:

There are three topologies here: The first two are our regular chain topology, one with two nodes and one with four nodes. The third topology is still a chain of four nodes, with the only difference that the distance between the nodes is reduced from 250 m to 50 m, so

that all nodes are within the range of each other. (knowing that the transmission range is 250 m for each node)

Traffic Pattern:

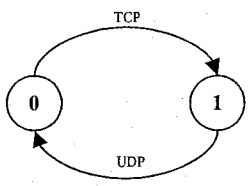
In all the three setups, there is one UDP connection going on between two nodes, and one TCP connection going on between two other nodes.

Variable Parameters:

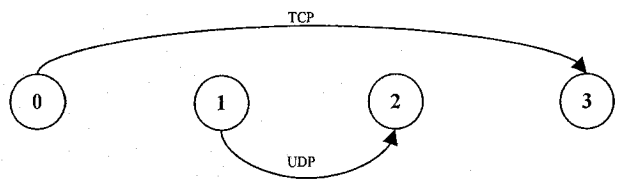
UDP connection's CBR load, ranging from 0 to 1750 Kbps (which is the UDP channel capacity, as was concluded in Sec. 4.6)

Results:

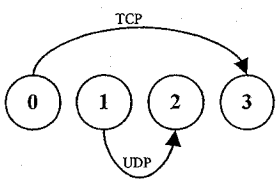
Case #1:



Case #2:



Case #3:



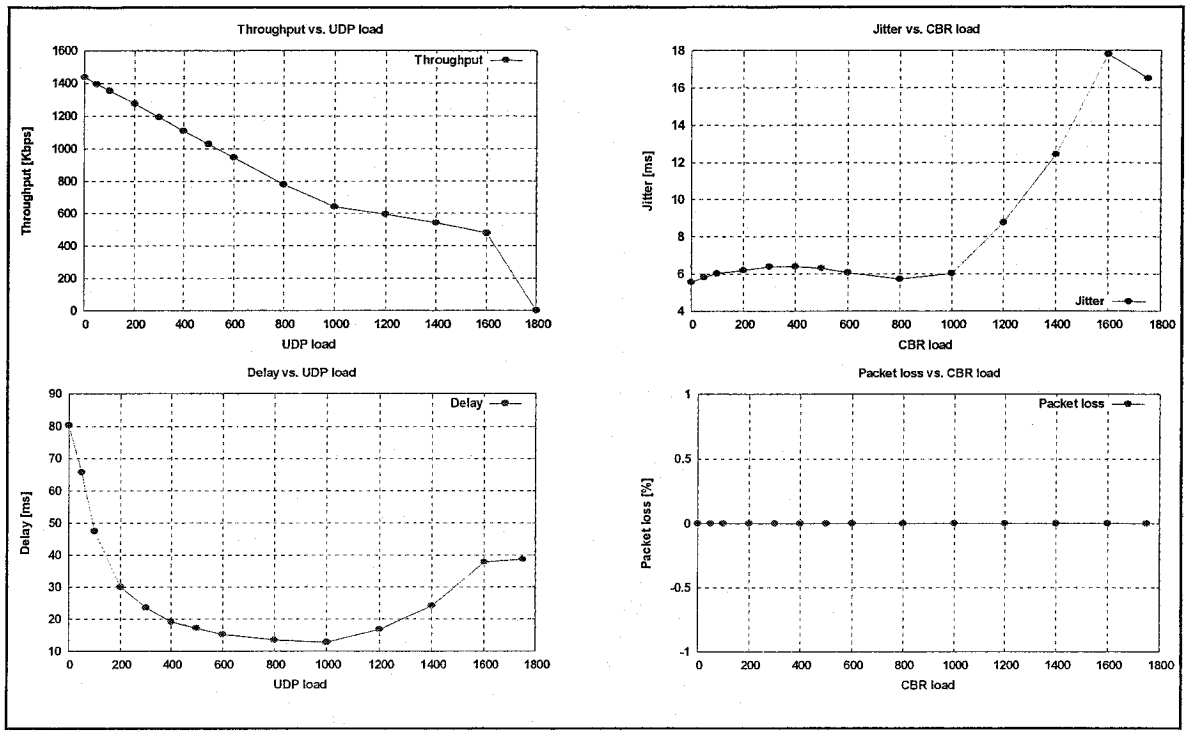


Figure 4.45 - Performance for Case #1

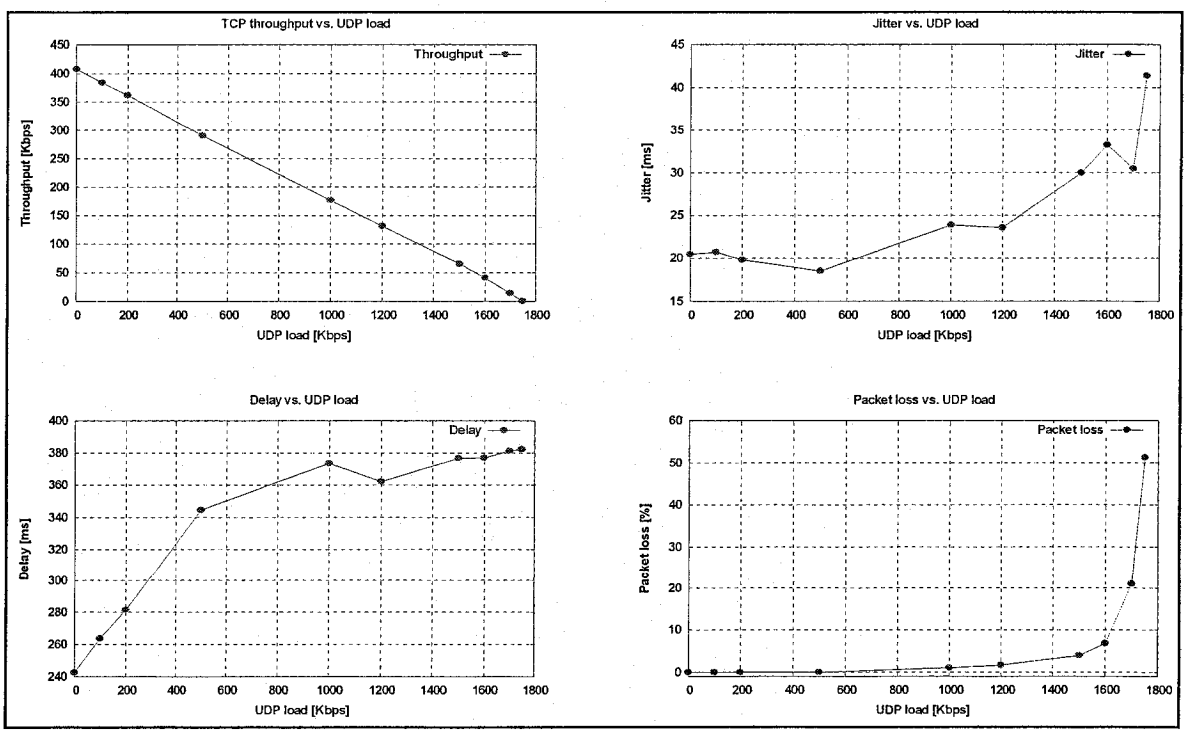


Figure 4.46 - Performance for Case #2

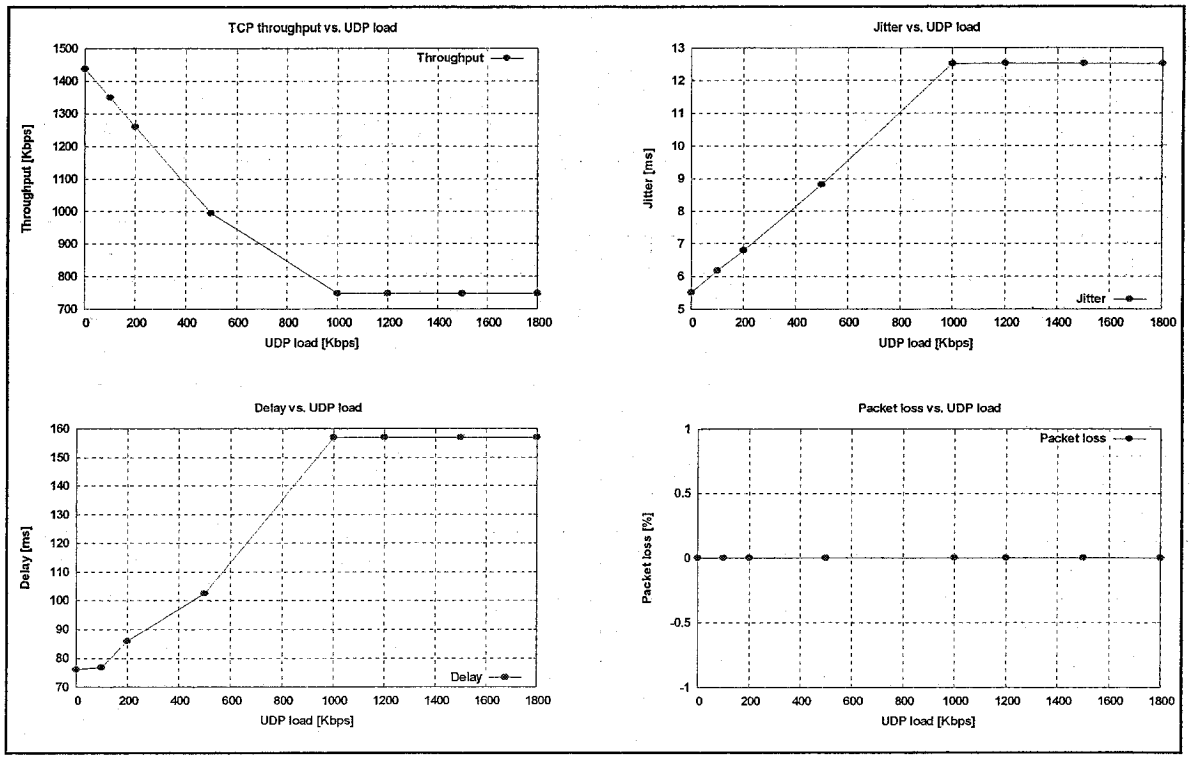


Figure 4.47 = Performance fod Case #3

Interpretation:

Case #1:

This is similar to Case #1 of Exp. #13, but here instead of two TCP connections, there is one TCP and one UDP. The purpose is to see how the UDP connection affects an ongoing TCP connection between the same nodes, and compare this with the TCP results from previous Section.

Fig. 4.46 shows that the throughput is dropping from a maximum value of about 1430 Kbps (the regular one-hop bandwidth, Exp. #1) when the UDP load is zero, to a minimum value of 0 for a UDP load of 1750 Kbps (which is the maximum one-hop UDP bandwidth, Exp. #6). So literally it is the UDP load which is in control of TCP, and it can *capture* the medium completely, and this is unfair sharing of the wireless medium.

Thus, the sensitivity of the network is much more to UDP traffic, than to TCP. (This is also confirmed in [21] in a grid topology, though there the IEEE 802.11 protocol is not used)

Note that this was not the case for two TCP connections, since, as we saw in the previous Section, they shared the medium between them quite fairly.

Also note that there is a slow-down in the throughput drop between a load of 1000 and 1600 Kbps. I call this region a *balanced* region, where TCP throughput has already been reduced by UDP to a rate that further increase is UDP load is not affecting it much (there are enough resources for both of them to operate in the medium). But beyond this region, i.e. with a load of more than 1600 Kbps, the capturing effect of UDP appears again, up to a point where it seizes the medium completely and drags TCP's throughput to 0.

A reasonable explanation for the delay performance, where is first goes down exponentially to a minimum of about 15 ms at 1000 Kbps load, and then starts to rise again and reach the almost-constant value of 40 ms at 1600 Kbps load, is as follows:

At first, when the throughput is decreasing linearly with the offered load, delay is also decreased, because TCP traffic is getting lighter, and UDP traffic is still not *that* heavy to fill the buffers and affect the delay. But at about 800 Kbps (when UDP has occupied about half of its maximum bandwidth), UDP packets start to saturate the medium and fill the buffers, causing the delay to rise back again.

So one can say that at low UDP rates (below 800 Kbps), the TCP traffic is dominant, and delay performance is following the TCP traffic, but as the UDP load increases, it's more UDP packets who are flying in the medium and occupying the buffers, and affecting the delay.

A similar discussion can be presented for the jitter performance.

As for packet loss, there's none, during the whole load range. Note that this is considering only the TCP connection, for UDP there are packet losses, but it is not of interest here.

Case #2:

This is similar to Case #2 of Exp. #13, but here instead of two TCP connections, there is one TCP (between the two end nodes) and one UDP (between the intermediate nodes). The purpose is to see how a UDP connection between nodes involved in forwarding packets for a separate TCP connection, affects the performance of the TCP connection, and compare it with Case #2 of Exp. #12.

Fig. 4.47 shows that the throughput is dropping from a maximum value of about 400 Kbps (the regular three-hop bandwidth, Exp. #1) when the UDP load is zero, to a minimum value of zero for UDP load of 1750 Kbps (which is the maximum one-hop UDP bandwidth, Exp. #6). In this case, the drop is linear and monotonic.

So again UDP's capturing effect is evident here.

Delay and jitter are also increasing with UDP load, as expected.

From the packet loss graph, we can see that packets start to be lost (actually dropped) when the UDP load goes beyond 600 Kbps, and the loss ratio increases exponentially with load, reaching to a maximum of 50% when UDP has saturated the intermediate (1 -> 2) link. These packet losses are induced, because two nodes are involved in forwarding packets, so as the load increases, TCP data packets (coming from Node 0) are dropped from Node 1's MAC transmit buffer, and TCP ACK packets (coming from Node 3) are dropped from Node 2's buffer. They are dropped from those nodes, because the non-stop

flow of CBR source is constantly injecting packets into the air, and won't allow TCP packets to contend for the medium fairly.

Case #3:

This setup is similar to the previous one, except that here all the 4 nodes are within the transmission range of each other, so there is no forwarding involved.

In Case #2, the UDP load was affecting the ongoing TCP connection adversely by consuming resources in two ways: First, it was using the wireless bandwidth, thus making TCP packets flying on hops 0 -> 1 and 2 -> 3 (and the reverse direction) contend more and more for the medium; and second, it was using the available *forwarding capacity* (i.e. bandwidth *and* MAC buffer space) on hop 1 -> 2.

In contrast, in Case #3, the UDP load is having a negative effect on the TCP connection only in the sense of occupying the shared wireless medium (first part above), which is actually the medium surrounding all four nodes.

As a result of this reduced effect of UDP load, we expect the performance to be better than what it was in Case #2. The results in Fig. 4.48 confirm this.

Still the negative impact of UDP load is present: it is reducing the throughput, and increasing delay/jitter. But everything comes to a stable state when the load reaches the value of 1000 Kbps, after which there is no further reduction in throughput and delay/jitter.

At this point the TCP throughput has been reached almost half of its maximum no-load value (740 Kbps). We can safely say that, at this load, the two protocols are operating in harmony with each other. UDP is not able to capture any more of the wireless bandwidth from TCP, like it did in the other two setups of this section.

The reason for this, as stated earlier, is that there is no forwarding involved, and the only resource shared between TCP and UDP is the wireless medium. The MAC resources are no more shared.

It is interesting that there are no packet losses as well. (though this is only regarding the TCP connection, for UDP there will be packet losses.)

Suggestions:

Unregulated UDP connections have quite a negative impact on the performance of TCP connections in wireless multi-hop networks. The word unregulated here means UDP connections that are not administered and managed according to the network conditions and network capacity.

This negative impact is sometimes called capturing effect, where UDP connections capture unboundedly (only bounded by their source CBR rate) the wireless medium's capacity, and the nodes' buffer capacity, causing TCP connections to dry up.

This effect is less severe when UDP nodes are not involved in forwarding TCP nodes' packets, and the capturing effect *saturates* at a specific UDP load, depending on the specific scenario.

As a result, it is always wise to monitor and adjust the sending rate of a UDP source, and bound it to a rate which doesn't spoil the ongoing TCP connections. This rate can be different for each scenario, for example in our simple setup, it is recommended not to increase the CBR rate above 1000 Kbps.

Chapter 5 Conclusion and Future Work

5.1 Summary

In this work, the performance of two transport layer protocols, TCP and UDP, over wireless ad hoc and multi-hop networks was analyzed. The ns-2 network simulator software package was used for the simulation of such network.

Several scenarios (experiments) with different configuration parameters were set up, and a set of conclusions/suggestions were made at the end of each experiment.

In Chapter 1, an introduction to IEEE 802.11 wireless LANs, different versions of the standard, and a summary of their operation was given.

In Chapter 2, TCP protocol was briefly introduced along with some of its important mechanisms. UDP was also briefly discussed. Then the issues of using TCP over wireless LANs, the current research status, and the approach of this thesis were presented.

Chapter 3 introduced the simulation model used for our analysis, along with the configuration/performance (input/output) parameters used, and the procedure followed for each experiment.

Chapter 4 is where the experiments, their results, analysis of the results, and possible suggestions for improvement are presented.

5.2 Conclusions

The optimization of the TCP protocol over wireless (specially ad-hoc multi-hop) networks is still a challenge which is wide open to the research community.

This thesis contributes to this environment by trying to analyze and compare the aspects of this topic not covered or experimented previously.

To this end, we followed these paths throughput the thesis:

- A static network was used, so as to minimize the effect of routing on our performance evaluations, since we were concerned mainly with the transport layer performance.
- A basic simulation of TCP over a multi-hop connection. This serves as our reference framework for other experiments and their comparison.
- Effect of packet size was analyzed, and for best throughput performance a packet size of about 2300 bytes was suggested. This value is resulted from two observations: the throughput saturation point which happens around 2500 bytes, and the MAC fragmentation threshold which is 2312 bytes. This packet size will result a 10% increase in throughput over the commonly used 1500-byte packet.
- The use of the RTS/CTS scheme was analyzed, and it was recommended not to use the scheme for lightly-loaded networks, as it only introduces more overhead and degrades performance (up to 15% in throughput, 20% in delay, and 10% in jitter).
- Different TCP versions were compared. DACK showed about 8% improvement in throughput in 1-2 hop connections (with the penalty of higher delay), and Vegas showed throughput improvement for 3-6 hop connections, up to 30% throughput increase over 6 hops. Vegas TCP's delay and jitter performance were better than all the other versions, up to 50% in some cases.

- Effect of MAC buffer size on TCP was analyzed, and the huge impact of this parameter on TCP performance was made clear. In terms of packet loss, an increase in MAC buffer size from 5 to 20 packets, results in a decrease from 5% to 0-1% packet loss. Also, we concluded that increasing the buffer size beyond the value of 20 packets has almost no effect on any performance parameter.
- For comparison purposes, a UDP connection was simulated with variable CBR rate, and a maximum UDP channel capacity of 1.7 Mbps was derived from the results. Knowing this channel capacity is very useful when implementing a wireless network where UDP traffic is to be passed through, since loading the network beyond this limit will result in unexpected overall degradation.
- A comparison of TCP and UDP was made to illustrate the effect of TCP rate control mechanisms on the achievable throughput, compared to UDP which has no rate control mechanism. It was shown that UDP is performing better (up to 32% in throughput over six hops), but only if its rate is below a certain capacity, after which UDP performance degrades rapidly, even going below TCP.
- Results of operating a UDP connection within the capacity of the network were presented, and it was shown that in these conditions, UDP is performing very well: no reduction in throughput over the whole span of hops (290 Kbps constant), and a delay and jitter performance an order of magnitude better than the similar TCP connection.
- The wireless channel is usually assumed to be error-free when transport protocols are analyzed. We analyzed the effect of packet errors on TCP performance, and it was shown that the effect is much more severe in high hop-count paths compared

to single or double-hop paths (70% and 30%, respectively). Yet the MAC layer is so powerful in error handling, that is can recover from all errors when the packet error rate is below 8%.

- UDP's operation was also analyzed under noisy channels, and it was observed that because of UDP's lack of any feedback or rate control mechanism, the degradation of performance in presence of errors is more than what it is for TCP (a packet loss of 40% for PER of 4% in UDP, while in TCP the packet loss was only about 2.5% for the same PER), specially when considering delay (delays of up to 2000 seconds). It was suggested that to achieve fair results for all users, in such environments with high packet loss rate, it is wise to measure and predict the performance before implementing the network.
- Different TCP versions were simulated over a network with packet errors present, and the performance of these versions were compared in such conditions. Based on the results, the use of Delayed ACK option was recommended, since it increases the throughput about 10% compared to all other versions (although at the same time it has a negative effect on delay/jitter). Also again, it was concluded that TCP Vegas is performing very well, specially in high hop-count paths, increasing the throughput an average of 30% over other TCPs. Delay/jitter performance of Vegas is also well below the other versions.
- Simultaneous TCP connections were set up between different node pairs to analyze a more realistic scenario. In this case, we concluded that when nodes involved in forwarding packets of other nodes, establish TCP connections themselves, the connections passing through them are degraded. This degradation

has the equal effect of a one-hop increase in the original connection path, for each additional forwarding node involved in a connection. Also, it was observed that two TCP connections share the bandwidth fairly between them, although when the number of hops increases, one party always tends to capture more of the bandwidth.

- Effect of loading a network in which TCP connections are going on, with UDP connections, was analyzed. It was observed that unlike the case for TCP, where connections share the bandwidth fairly, UDP connections can capture the medium completely, to a point where TCP connections are put out completely (this happens at the point where UDP is sending at a rate equal or greater than the link capacity, i.e. 1.7 Mbps for a one-hop link). Thus it was suggested that UDP connections be used in wireless networks in a 'regulated' manner, in order to avoid network overloading.

5.3 Future Work

Considering the scope of this work, following are a few suggestions and improvements which can be made for any future work on this topic.

The results of this work can be used in further research very easily and effectively, since the exact setups and configuration parameters used, are explained clearly.

- Use of other types of traffic sources/applications (other than FTP and CBR), like Web traffic, real-time audio/video traffic, ...
- Incorporating mobile node movements and topology changes (effectively routing) and analyzing its interaction with transport layer performance

- Involving more complex topologies, more nodes and establishing more connections between them, to see what's the effect of network loading and scaling
- Analyzing the effect of changing various TCP parameters (like maximum window size, DACK time-out interval, ...) and finding their optimum value for this specific type of network

BIBLIOGRAPHY

[1] Saliga, S.V., "An introduction to IEEE 802.11 wireless LANs", *Radio Frequency Integrated Circuits (RFIC) Symposium, 2000.* Digest of Papers. 2000 IEEE, pp. 11-14

[2] "IEEE 802.11", 1999 Edition,

http://standards.ieee.org/getieee802/download/802.11-1999.pdf

[3] "IEEE 802.11b-1999",

http://standards.ieee.org/getieee802/download/802.11b-1999.pdf

[4] "IEEE 802.11a-1999",

http://standards.ieee.org/getieee802/download/802.11a-1999.pdf

[5] "IEEE 802.11g-2003",

http://standards.ieee.org/getieee802/download/802.11g-2003.pdf

[6] "HIPERLAN/2 Standard", http://portal.etsi.org/bran/kta/Hiperlan/hiperlan2.asp

[7] "Bluetoosh Standard", http://www.bluetooth.com

[8] Brian P. Crow et al., "IEEE 802.11 Wireless Local Area Networks", *IEEE Communications Magazine,* September 1997, pp. 116-126

[9] Elizabeth M. Royer, Chai-Keong Toh, "A Review of Current Routing Protocols for Ad Hoc Mobile Wireless Networks", *IEEE Personal Communications*, April 1999, pp. 46-55.

[10] C.E. Perkins and P. Bhagwat, "Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers", *Comp. Comm. Rev.*, October 1994, pp. 234-244

[11] A. Iwata et al, "Scalable Routing Strategies for Ad Hoc Wireless Networks", *IEEE Journal on Selected Areas in Communications*, Special Issue on Ad-Hoc Networks, August 1999, pp. 1369-79

[12] S. Murthy and J.J. Garcia-Luna-Aceves, "An Efficient Routing Protocol for Wireless Networks", *ACM Mobile Networks and Applications Journal*, Special Issue on Routing in Mobile Communication Networks, October 1996, pp. 183-97

[13] C. E. Perkins, E. M. Royer, "Ad Hoc On-demand Distance Vector Routing", *Proceedings of 2nd IEEE Workshop on Mobile Computing Systems and Applications,* February 1999

[14] D. B. Johnson and D. A. Maltz, "Dynamic Source Routing in Ad-Hoc Wireless Networks", *Mobile Computing,* Kluwer Academic Publishers, 1996, pp. 153-181.

[15] V. D. Park and M. S. Corson "A Highly Adaptive Distributed Routing Algorithm for Mobile Wireless Networks", *Proceedings of INFOCOM '97*, April 1997.

[16] "IEEE 802.11 Working Group official Web site"
http://grouper.ieee.org/groups/802/11/

[17] "Wikipedia Online Encyclopedia, IEEE 802.11"
http://en.wikipedia.org/wiki/IEEE_802.11/

[18] A. Jamalipour, "The Wireless Mobile Internet: Architectures, Protocols and Services", Wiley, 2003

[19] G. Xylomenos, G. C. Polyzos, "TCP and UDP Performance over a Wireless LAN", *INFOCOM '99*, Proceedings of the IEEE, March 1999

[20] G. Xylomenos, G. C. Polyzos, "Wireless Link Layer Enhancements for TCP and UDP Applications", *Proceedings of the Parallel and Distributed Processing Symposium*, April 2003

[21] M. Gerla et al., "TCP over Wireless Multi-hop Protocols: Simulation and Experiments", *Proceedings of IEEE ICC'99*, June 1999

[22] E. Altman, T. Jimenez, "Improving TCP over Multihop Networks using Delayed ACK", *MADNET '03,* March 2003

[23] Z. Fu et al., "On TCP Performance in Multihop Wireless Networks"

[24] M. Miyoshi et al., "Performance Improvement of TCP on Wireless Cellular Networks by Adaptive FEC Combined with Explicit Loss Notification", *IEEE 55th Vehicular Technology Conference VTC*, Spring 2002

[25] M. Petrovic, M. Aboelaze, "Performance of TCP/UDP under Ad Hoc IEEE 802.11", *10th International Conference on Telecommunications ICT*, March 2003

[26] M. Mathis et al., "TCP Selective Acknowledgement Options", *IETF RFC 2018*, October 1996

[27] A. Bakre, B. R. Badrinath, "I-TCP: Indirect TCP for Mobile Hosts", *Proceedings of 15th International Conference on Distributed Computing Systems*, May 1995

[28] H. Balakrishnan et al., "Improving TCP/IP Performance over Wireless Networks", *Proceedings of 1st ACM International Conference on Mobile Computing and Networking (Mobicom),* November 95

[29] Network Simulator 2 (ns-2), available at:
http://www.isi.edu/nsnam/ns/

[30] REAL Network Simulator, available at:

http://www.cs.cornell.edu/skeshav/real/

[31] Tcl Developer Xchange: http://www.tcl.tk/

[32] K. Ramakrishnan, S. Floyed, D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", *IETF RFC 3168*, September 2001

[33] H. Balakrishnan, R. H. Katz, "Explicit Loss Notification and Wireless Web Performance", *IEEE Global Telecommunications Conference (Globecom '98)*, November 1998

[34] "The ns Manual" available at: http://www.isi.edu/nsnam/ns/ns-documentation.html

[35] M. Gerla, K. Tang, R. Bagrodia, "TCP Performance in Wireless Multi-hop Networks", *Proceedings of IEEE WMCSA '99*, February 1999.

[36] D. Allen, "Hidden Terminal Problems in Wireless LAN's", *IEEE 802.11 Working Group paper 802.11/93-xx*

[37] V. Jacobson, "Congestion avoidance and control", *ACM SIGCOMM '88: Symposium on Communications Architectures and Protocols*, August 1988, pp. 314-329

[38] V. Jacobson. "Modified TCP Congestion Avoidance Algorithm", *Technical report*, April 1990

[39] K. Fall and S. Floyd, "Simulation-based Comparisons of Tahoe, Reno, and SACK TCP", *ACM Computer Communication Review*, July 1996

[40] L.A.Grieco and S. Mascolo, "Performance evaluation and comparison of Westwood+, New Reno, and Vegas TCP congestion control", *ACM SIGCOMM Computer Communications Review*, April 2004, pp. 25-38

[41] S. Floyd and T. Henderson, "New Reno Modification to TCP's Fast Recovery", IETF *RFC 2582*, April 1999

[42] L. S. Brakmo et al, "TCP Vegas: End-to-end congestion avoidance on a global Internet", *IEEE Journal on Selected Areas in Communications (JSAC),* October 1995, pp. 1465-1480

[43] M. Mathis et al, "TCP Selective Acknowledgement Options", *IETF RFC 2018*, April 1996

[44] R. Braden, "Requirements for Internet Hosts - Communication Layers", *IETF RFC 1122*, October 1989

[45] N. Cardwell and B. Bak, "A TCP Vegas Implementation for Linux", available at: http://flophouse.com/~neal/uw/linux-vegas/, May 2004

[46] A. Leon-Garcia and I. Widjaja, "Communication Networks: Fundamental Concepts and Key Architectures", Boston McGraw-Hill, 2004

[47] Wi-Fi Planet Web site Technology Articles, at: http://www.wi-fiplanet.com

[48] Kamesh Medepalli, "Voice Capacity of IEEE 802.11b, 802.11a and 802.11g Wireless LANs", *Proceedings of IEEE GLOBECOM*, 2004

[49] R. Sanchez, J. Martinez, et al, "TCP/IP Performance over EGPRS network", *Proceedings of Vehicular Technology Conference*, IEEE 56th VTC, September 2002

[50] W. Stevens, "Internet RFC 2001", *Network Working Group Request for Comment 2001*, January 1997, available at: http://www.faqs.org/rfcs/rfc2001.html