

Investigation of Cross-layer approach:

*Interaction between MAC and Network
Layers*

Table of Contents

ABSTRACT	3
1. INTRODUCTION	3
2. WIRELESS NETWORK.....	4
2.1 <i>Infrastructure networks</i>	4
2.2 <i>Ad hoc networks</i>	5
3. ISSUES IN AD-HOC NETWORKS	5
3.1 <i>Medium Access Control Problem</i>	5
3.2 <i>CSMA/CA</i>	6
3.3 <i>The Routing Problem</i>	8
4. PROBLEM DESCRIPTION	8
5. ASSUMPTIONS:	8
6. SIMULATION ENVIRONMENT	8
7. RESEARCH APPROACH	9
8. SIMULATION RESULTS.....	10
8.1 <i>The Chain of Nodes</i>	10
8.2 <i>Lattice with Only Horizontal Flows</i>	13
8.3 <i>Lattice with Crossing (Horizontal and Vertical) Flows</i>	14
8.4 <i>Power Control</i>	15
9. FUTURE WORK	16
10. CONCLUSION	16
11. REFERENCE	17
APPENDIX	18
A.1 <i>NS2 Scripts</i>	18
A.2 <i>UNIX Shell Scripts – Extracting the Throughput Info from Trace Files</i>	19
A.3 <i>UNIX Shell Scripts – Obtain the Throughputs over a Set of Scenarios</i>	20

Abstract

Ad-hoc wireless networks enable new and exciting applications but also pose significant technical challenges due to the need for decentralized control, dynamic topology and the characteristics of the wireless channel. Cross-layer design is a promising method to satisfy the network requirements. In this research, the medium-access-control (MAC) protocol prescribed in the IEEE 802.11 standard, namely Carrier Sense Multiple Access/Collision Avoidance (CSMA/CA), is studied in detail. A simulation study of the throughput performance of CSMA/CA over various ad-hoc network topologies is proposed. To better understand the cross-layer issues involved in wireless ad-hoc network design, the effect of the choice of packet length on the performance of CSMA/CA will be studied. In addition, the influence that power control and minimum power routing have on achievable throughput will be investigated. It is expected that the results will help to understand how the MAC protocol interacts with the network layer and will aid in the development of medium access and routing protocols.

1. Introduction

The growth of the wireless industry was focused mainly on voice alone. It is clear that there will be increasing demand for wireless data services [1]. Adding data-oriented services in the form of Wireless LAN and Internet-based packet data to the cellular and PCS world will trigger yet another explosion. Various mechanisms have been proposed and recently deployed to support data traffic over wireless media. An ad hoc wireless network is one such mechanism. It is a collection of wireless mobile nodes that self-configure to form a network without the aid of any established infrastructure. Ad hoc wireless networks are highly appealing for many reasons. They have the advantage of rapid deployment and reconfiguration. It is easy to tailor the network to specific applications. They are highly robust due to their distributed nature, and the lack of single point of failure.

While ad hoc networks exhibit much promise, they also pose some significant design challenges. The challenges are due to their lack of established infrastructure, the need for decentralized control, dynamic topology, and wireless channel characteristics. Despite the challenges posed, the various network requirements need to be met. As a promising solution for efficient networking, we address the issue of cross-layer design of network protocols. Previous approaches interpreted and implemented the network as a hierarchy of layers that are independent and non-cooperating. Thus, they were unable to take advantage of the interactions between the layers. The approaches were optimizing each layer by itself when a joint optimization across the various layers of the network would have resulted in greater performance. The inflexibility and sub-optimality of this paradigm results in poor performance of ad hoc wireless networks in general, especially when energy is a constraint or the application has high bandwidth need and/or stringent delay constraints. To meet these requirements a cross-layer protocol design that supports adaptivity and optimization across layers of the protocol is required. In an adaptive cross-layer protocol stack, the link layer can adapt rate, power, and coding to meet the requirements of the application given current channel and network conditions. The Medium Access Control (MAC) layer can adapt on underlying link and interference conditions as well as delay constraints and priorities. Adaptive routing protocols can be developed based on current link, network, and traffic conditions. It has been shown that cross-layer networking can improve performance by as much as a factor of two [2] [9].

Improvement of the MAC layer in ad-hoc wireless networks is essential if we want to maximize the network efficiency. The MAC protocol dictates how different users share the

available channel. This project will focus on Carrier Sense Multiple Access/Collision Avoidance (CSMA/CA), the MAC protocol prescribed in the IEEE. We will elaborate the actual workings of CSMA/CA in the subsequent chapters.

In this research, we study how different network layer parameters influence the performance of CSMA/CA. The performance of CSMA/CA in terms of the throughput of the network and average throughput will be studied. CSMA/CA alone would give us some performance metric, which we can easily analyze and simulate but in this project, we will investigate how the power control scheme affects the performance of CSMA/CA. Additionally, how various topologies and how interference of other users influences the throughput of CSMA/CA will be studied in details.

2. Wireless network

Wireless networking is an emerging technology that allows users to access information and services electronically, regardless of their geographic position. One method of classifying wireless networks is as follows:

2.1 Infrastructure networks

As shown in the figure 1, an infrastructure network consists of a network with fixed and wired gateways. A mobile host communicates with a bridge in the network (called base station) within its communication radius. The mobile unit can move geographically while it is communicating. When it goes out of range of one base station, it connects with new base station and starts communicating through it. This operation is called handoff. In this approach the base stations are fixed.

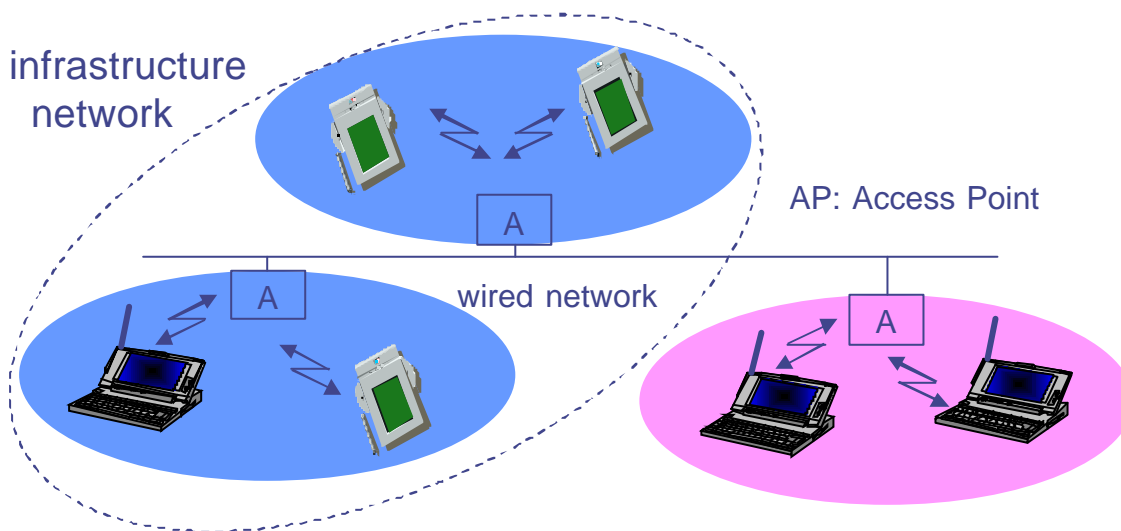


Figure 1: Infrastructure network

2.2 Ad hoc networks.

ad-hoc network

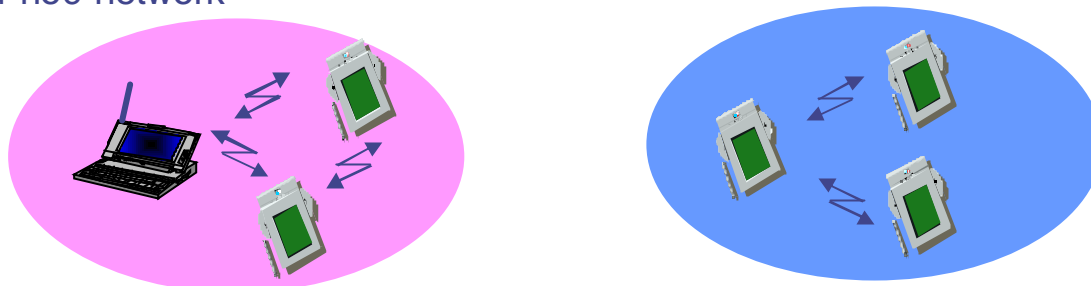


Figure 2: Ad hoc network

In ad hoc networks, all nodes are mobile and can connect dynamically to other nodes within range. All nodes in these networks behave as routers and take part in discovery and maintenance of routes to other nodes in the network. Ad hoc networks are very useful in emergency search-and-rescue operations, meetings or conventions in which persons wish to quickly share information, and data acquisition operations in inhospitable terrain.

3. Issues in Ad-hoc networks

In an ad hoc network, packets are relayed from node to node until they reach their final destination. This is because; in an ad hoc wireless network, each node is connected to only a few nodes, which are in its vicinity. Nodes may be mobile, changing their location over time. In addition, nodes may switch themselves off from time to time.

The objective of operating such a network is to transfer packets from their sources to their destinations reliably and efficiently. Nodes must choose the power levels at which they broadcast, since that influences the range, and delay experienced by their transmissions. Nodes must also cooperate in relaying each other's packets.

The wireless medium is an unreliable medium. Transmissions are subject to obstacles, reflections, multi-path effects, and fading, all of which affect the quality of the received signal. Transmission can moreover interfere with each other. All these conspire to make packet reception much more unreliable than in wired media.

Given the characteristics of the wireless medium, how would we design a system that can reliably and efficiently transport packets from sources to destinations? What are the issues confronted in designing ad hoc networks? The major issues are explained in the following sections in details.

3.1 Medium Access Control Problem

For a receiver to intelligibly receive the packet, other nearby transmitters should refrain from broadcasting. Otherwise, one has the problem of “collisions” where packets destructively interfere with each other. This is the basis of the Medium Access Control Problem. How should nodes schedule their transmissions in order to have their packets received intelligibly by their intended receivers?

Ad hoc networks have additional problems compared to the traditional wired case, since not all nodes can hear all transmissions. The goal is to spatially reuse the radio frequency spectrum. That is, by restricting the range of transmission, one can limit the interference of one's transmission, thus allowing a distant transmitter-receiver pair to carry on conversation on the same frequency at the same time. Thus, the MAC protocol needs to spatio-temporally schedule transmissions. One scheme that has been proposed for 802.11 is the Carrier Sensing Multiple Access with Collision Avoidance (CSMA/CA). In our research, we will be focusing on the performance of CSMA/CA. The reason behind focusing on IEEE 802.11 lies in the fact that it is a widely accepted WLAN standard and it is also used extensively in ad hoc network test-beds. Additionally, it is the most widely used standard of the few that support multi-hop transmission in its DCF (Distributed Coordination Function) mode.

3.2 CSMA/CA

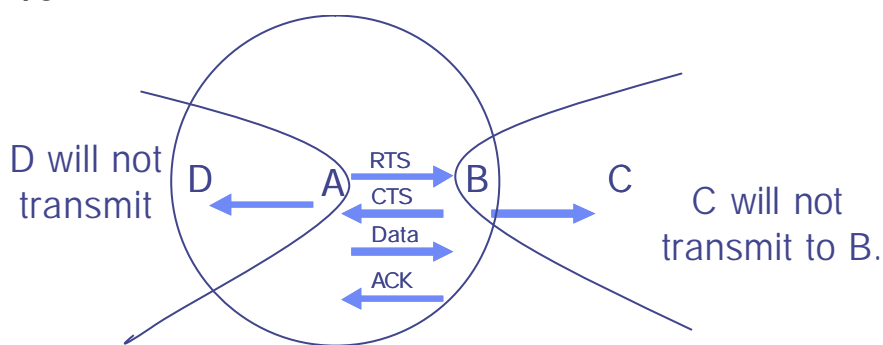


Figure 3: CSMA/CA

The basic operation of CSMA is as follows [3]. The situation is described in Figure 3. When a node A has a packet to send to a node B, it first sends out a “request to send” packets, called RTS. This transmission can be heard by all the neighbors of A, including B. Suppose, as in Figure 3, that A has one additional neighbor, D, and B has one additional neighbor, C.

If D is not currently in range of any other transmission, then it can hear the RTS from A. It should then refrain from transmitting for a while. “Silencing” of node D is achieved by setting the Network Allocation Value, NAV, of node D to some value, which corresponds to the time that will take for the whole transaction, assuming everything goes well. The particular NAV value is specified in each RTS packet. Nodes in the range of A after setting NAV will keep on decreasing the value until it hits zero. Those nodes will remain silent as long as NAV has a positive value. Similarly, if B is not currently in range of any other transmissions, then it too can hear A’s RTS. If it is not currently under an order to remain silent, it sends a “clear-to-send” CTS packet to T. If C is not in range of any other transmission, it will hear this CTS packet. Again, C is prevented from transmitting for the duration of the exchange by setting the NAV to the value specified by the CTS packet. This CTS from B is heard by A since D was previously silenced, and does not therefore cause interference by transmitting. Then A sends its data packet to B. B receives this successfully since its neighbor C has been silenced. Then R ends back an “acknowledgment” packet ACK to A. At this point, C is released from silence. Node D also is released from silence when it hears the end of A’s data packet, after an obligatory pause to allow node A to receive the subsequent ACK from node B. From these procedures, the “hidden terminal problem” is solved, leading to fewer collisions and ultimately increasing the throughput achieved by the MAC protocol.

This four-way handshake namely, RTS-CTS-DATA-ACK, needs to be carried out for each data packet on each hop.

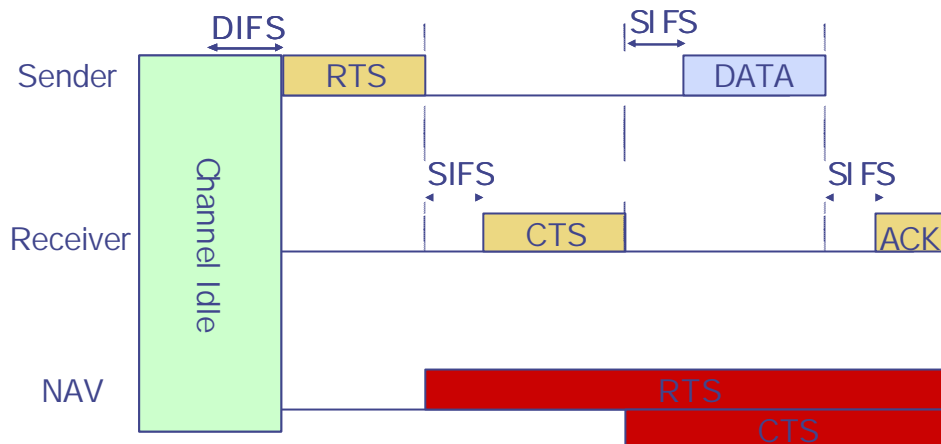


Figure 4: Operation of CSMA/CA

In case the channel was found to be idle the first time it is sensed by the transmitter and stays idle for a period equaling the DCF inter-frame spacing (DIFS) the transmitter can go ahead and send the RTS. But, if the channel was sensed to be busy, the transmitter waits until the channel reaches the idle state and stays idle for a period equaling DIFS, and then selects a random number between zero and a number called the contention window (CW) and starts counting down from this number. When the counter reaches zero the transmitter can go ahead with its RTS transmission. If another node transmits in the time during which the node is counting down, the node freezes its current counter state and during the next contention window, starts counting down from there. This ensures that no node ends up waiting indefinitely in order to send a packet due to sheer bad luck. In the event of a collision (e.g. not receiving a CTS) then the node exponentially increases the value of CW. This ensures that a node experiencing a lot of contention contends less frequently for the channel. The basic operations are illustrated in Figures 4 and 5.

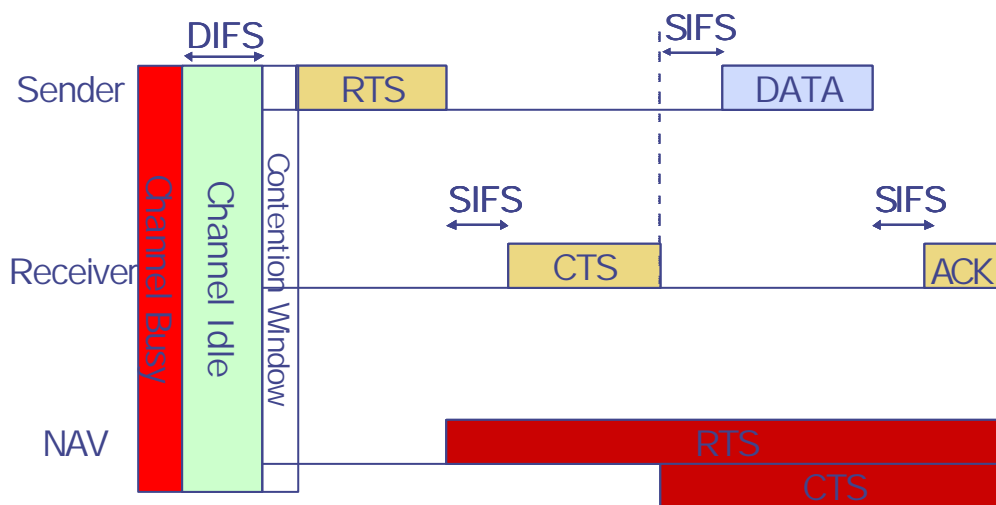


Figure 5: Operation of CSMA/CA: when the channel is busy

3.3 The Routing Problem

In addition to the already difficult routing problem in traditional wired network, the nodes in ad hoc network are no longer stationary. Moreover, the address of the node does not indicate the position of the node or how to reach it. The central problem is to design an efficient adaptive distributed routing algorithm. It necessarily must be adaptive due to the volatile and mobile nature of the ad hoc network, and it needs to be distributed since the nodes are neither spatially co-located nor is there any centrally available information.

From different routing protocols in ad hoc network, we chose to simulate the Ad-hoc on-demand distance vector routing protocol (AODV). AODV is an on-demand routing protocol. These protocols are used more extensively in the ad-hoc network domain compared to the table-based protocols. The simulations can be carried out using any on-demand protocol.

4. Problem Description

We study the performance of IEEE 802.11's MAC protocol under various conditions – CSMA/CA using achieved “throughput” as the metric. Throughput is defined as the number of bits of data successfully received at the destination per unit time. We investigate the impact of using different network topologies with varying number of interfering flows on achieved throughput and the effect packet size has on MAC layer performance by using three different packet sizes on each of the topologies. Finally, we also study CSMA/CA performance when transmit power is controlled. This research is intended to quantify the effect that network layer and physical layer parameters have on the performance of IEEE802.11's MAC layer.

5. Assumptions:

- Traffic Model: Constant Bit rate traffic (CBR)
- Bit Rate: 2 Mbps (max data rate in the original 802.11 specifications)
- Routing Protocol: Ad-hoc on-demand distance vector routing (AODV)

We choose to use CBR traffic since we would like to identify the reasons for any flaws found in CSMA/CA. The use of other traffic models like TCP traffic would have introduced correlations across time, making an analysis of the results very difficult.

We choose an on-demand routing protocol, since these are widely used in the ad-hoc network domain compared to the table-based protocols. The simulations can be carried out using any on-demand protocol.

6. Simulation Environment

NS2 (Network Simulator) [4] is a discrete event simulator developed by the University of California at Berkeley. In 1998, Carnegie Mellon University added a wireless extension, known as Monarch [5]. It enables NS2 to simulate mobile nodes connected by wireless network interfaces. It can also simulate multi-hop routings over wireless ad hoc networks.

Table 1: Features of the Monarch wireless extension.

Layer	Model	Details
Physical	Radio propagation	The Friis free-space model with $1/r^2$ attenuation and the two-ray ground reflection model with $1/r^4$ attenuation
	Antenna	A unity gain omni-directional antenna.
	Shared media	A shared media model that subjects to collisions and the propagation model is implemented. Each node can overhear packets transmitted by the others as long as the senders are within the radio range of the receive node.
MAC	IEEE 802.11	The distributed coordination function (DCF)
Network	DSR	Dynamic source routing
	AODV	Ad-hoc on-demand distance vector
	DSDV	Destination Sequence Distance Vector
	TORA	Temporally Ordered Routing Algorithm

Table 1 briefly presents the models that the wireless extension of NS provides. For the simulation, we choose the two-ray ground reflection model, IEEE 802.11, and AODV at the physical, MAC, and network layers respectively.

NS2 provides several traffic models such as exponential distribution, Pareto distribution, and constant bit rate. The constant bit rate (CBR) model is chosen for this simulation.

7. Research Approach

In an ad hoc network with multi-hop routing, a packet moves from the source to destination along a chain of intermediate nodes. The successive packets sent by a single source to a destination may interfere with each other as they move along the chain, thus causing contention among nodes. To understand the mechanics of CSMA/CA, we first investigate a topology consisting of a chain of nodes. However, this is not a true reflection of a real-world ad hoc network where there may be a number of intersecting traffic-flows taking place simultaneously between different source-destination pairs. Hence, we further study the throughput of CSMA/CA on a lattice topology [6]. The lattice topology gives us a more complicated scenario that allows us to study the interactions among several traffic-flows. This topology is also simple enough and is amenable to an analysis based on first principles to estimate the throughput. With the lattice topology, two sets of flows are considered: horizontal flows only and crossing flows (i.e. both horizontal and vertical).

The type of traffic flow affects the ability of CSMA/CA to schedule packets, and the traffic flow in turn depends on the type of application. A video application, for example, will generate a larger data rate when the scene is changing rapidly than when it is still. In this project, we want to see how CSMA/CA works in general to schedule packets. We believe that the investigation of constant-bit-rate (CBR) traffic can give us enough insight about the scheduling ability of CSMA/CA. It will be a part of our future work to study more complicated traffic types. The bit rate of the CBR traffic is chosen to be 2Mbps because NS2 uses the basic rate 2Mbps for the default bandwidth of IEEE 802.11 protocol. We choose three packet sizes 64, 500, and 1500 bytes. An RTS packet is 40 bytes, an ACK packet is 39 bytes, and the MAC header of a data packet is 47 bytes long. Every packet suffers an overhead of 126 bytes due to the file header and the RTS/CTS/ACK contention. We choose to study the cases where the overhead is comparable or greater than the data (64-byte packets), one where the overhead is negligible

(1500-byte packets) and an intermediate case (500-byte packets). In addition, we study the impact of power control on CSMA/CA performance. We choose routes between the source and the destination that use the shortest hops and modulate the transmission power, so that the node transmits just the power required to reach its next hop neighbor. Furthermore, we run the simulation for a period sufficient for the transmission of 100,000 packets. We choose 100,000 packets so that we can observe the long-term behavior of the network and allow enough time for the short-term transients to die out.

8. Simulation Results

8.1 The Chain of Nodes

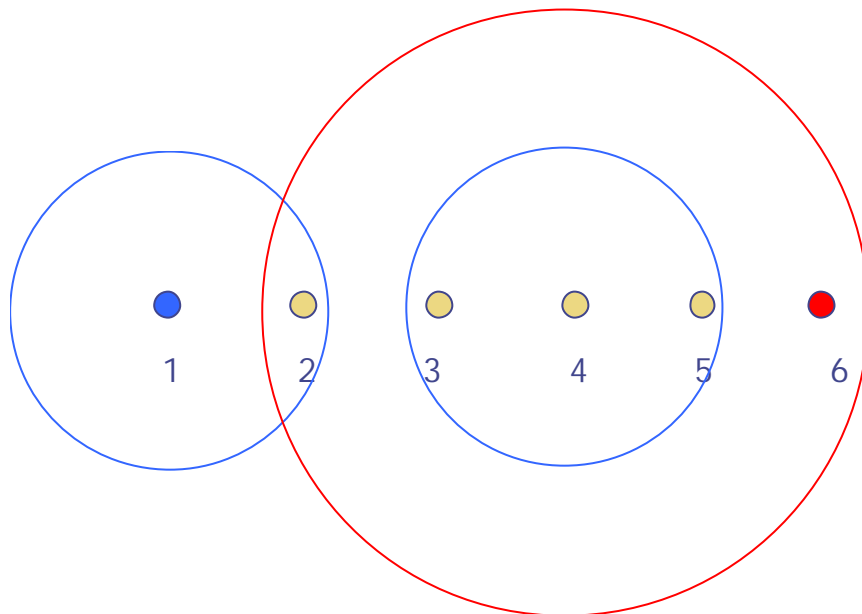


Figure 6: MAC interference in a chain of nodes. The blue circle denotes the valid transmission range of a node. The red circle denotes the interference range of node 4. The transmissions of node 4 corrupt the transmissions from 1 to 2. The inter-nodal distance is 200m

First, we simulate a chain of nodes. Consider the example network shown in the figure 6, where node 1 is the source and the last node 6 is the sink. The blue circle is the communication range of a node, that is, it can communicate successfully with any node within this range. The red circle is the interfering range of a node. In the space between the blue and red circles, the node is not able to communicate effectively but causes interference. We use the default values that NS2 provides: a node can correctly receive packets from 250 meters away but can interfere at 550 meters. Therefore, the packet transmission from node 4 will interfere with the RTS packets sent from 1 to 2, preventing 2 from successfully receiving the RTS from 1 or sending the corresponding CTS. Because of the 550-meter range of interference, node 1 and 4 cannot transmit at the same time. However, since nodes 2 and 5 are 4 nodes ($> 550\text{m}$) away, node 5 can transmit without interfering with successful reception at node 2. This leads to a channel utilization of $\frac{1}{4}$

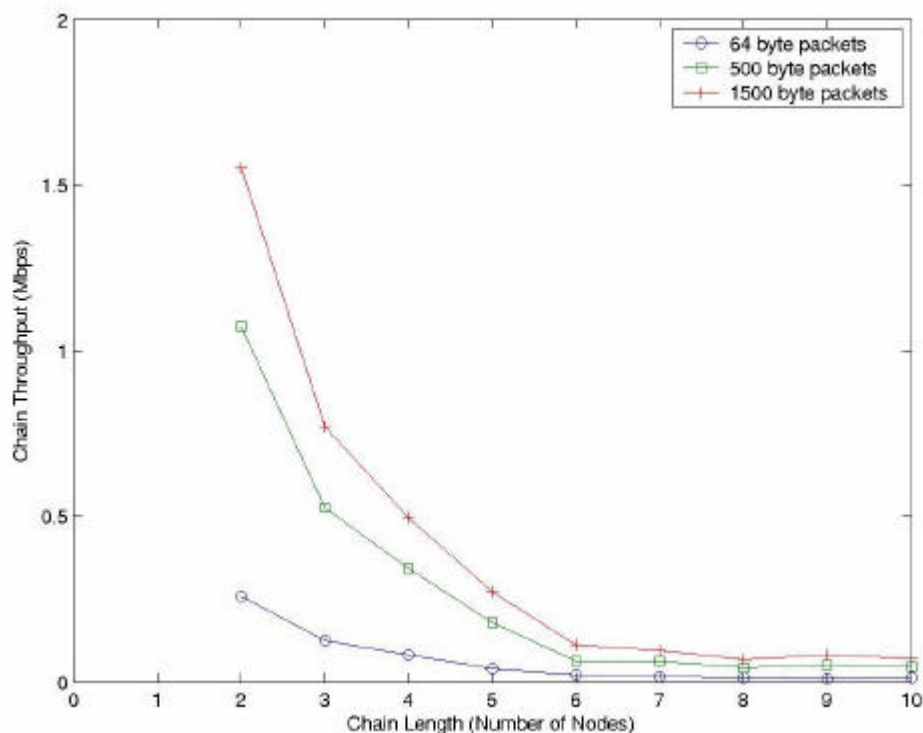


Figure 7: The throughput achieved on a chain of nodes as a function of chain length.

Figure 7 presents the simulation results for a single chain. The achieved throughput is plotted as a function of chain length. There is only one stream of traffic, where node 1 is the source and the last node is the destination. The two-node chain with 1500-byte packets has the largest throughput among all the cases. The achieved throughput in this case is 1.6 Mbps. Since there are no other interferers for the transmission of a two-node chain, we expect that the throughput can equal 2Mbps, which is the default data rate of 802.11 set up in the simulator. Nevertheless, the 2Mbps raw rate consists of actual data payload and the overhead caused by file headers and RTS/CTS/ACK contentions. The achieved rate 1.6 Mbps is reasonable if only the data payload is taken into account. From the analysis presented in the previous paragraph, we expect a channel utilization of $\frac{1}{4}$ when the chain gets longer, which corresponds to an achieved throughput of $1.6 / 4 = 0.4$ Mbps. However, the achieved throughput we obtain in Figure 7 is about 0.1 Mbps. It is natural to come up with the question: Why does CSMA/CA perform worse than our estimate? To answer this, we found it necessary to perform the following simulation.

Packets are transmitted at different controlled rates over a 6-node chain using 1500-byte packets. Figure 8 presents the results. When the offered load is light, CSMA/CA can schedule packets well, thus providing a linear increase in throughput with increase in offered load. However, once the offered load exceeds 0.3 Mbps, CSMA/CA performance drops drastically and the throughput obtained reduces. The peak of 0.3 Mbps is about the rate 0.37 Mbps, which we expect from the $\frac{1}{4}$ rule of thumb. We thus conclude that CSMA/CA is capable of sending at a rate very close to the optimal rate, but does not discover the optimum schedule on its own. Only when the sender transmits the exact amount of packets that CSMA/CA can handle appropriately, the optimal rate can be achieved. This observation is important for many applications because researchers and wireless users all want to know the limitation of the wireless LAN. The result we find is for the ad hoc mode of 802.11. There are also some researchers looking at the

infrastructure mode of 802.11. On carefully observing the simulations above, we found two phenomena that result in degraded performance when offered load is high. For the purpose of this paper, we call them injection and back-off problems, and the following paragraphs will explain them in detail.

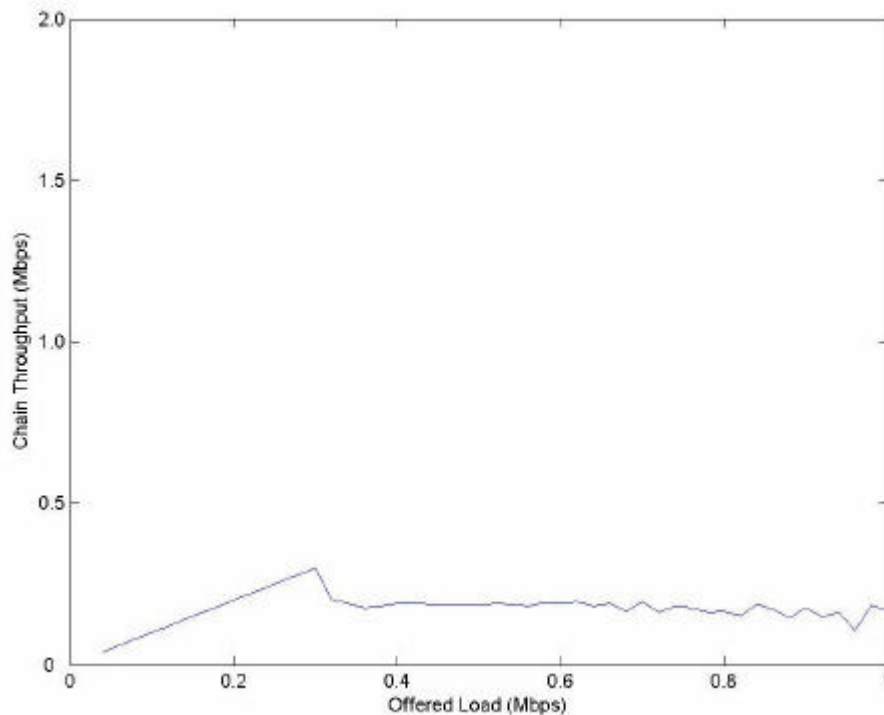


Figure 8: Achieved throughput as a function of different offer loads on an 8-node chain, using 1500-byte packets.

For instance, node 3 in the 6-node chain can send nothing if any of the five other nodes are transmitting, so node 3 has five interferers, whereas node 1 has only three interferers. Since node 1 has fewer interferers compared to node 3, it gets more opportunities to send its packets. This results in the *injection phenomenon*: node 1 injects more packets into the network than the subsequent nodes can forward. Therefore, the queues with limited sizes at the subsequent nodes cannot hold all the incoming packets but are forced to drop them. Some amount of time that node 1 spends on transmission is meaningless because subsequent nodes eventually drop a fraction of the packets. A better way of scheduling is to slow down the injection by the source node and allow time for the downstream nodes to handle the packets. Because CSMA/CA does not propose a scheme for subsequent nodes to send congestion feedback to the source node, the source node has no way of knowing what happens on the multi-hop route and the source node swamps the downstream nodes. If we appropriately incorporate congestion-control schemes into the current MAC protocol, the injection problem may vanish.

In addition to the inefficient bandwidth utilization caused by the injection problem, the back-off mechanism of CSMA/CA works unfavorably for multi-hop routing. Consider the case when node 4 is transmitting data to node 5 and node 1 tries to send an RTS to node 2. The interference from node 4 damages the RTS at node 2, thereby forcing node 1 to back off: node 1 doubles its contention window and waits longer before it retries. For the duration of transmission of node 4, all RTS attempts from node 1 fail. This causes the exponential back-off timer to migrate to very high values causing the *back-off phenomenon*. When node 4 completes

its transmission and has no more packets to send, node 1 may still remain backed-off and keep the wireless medium idle for longer than necessary, which we consider as wasted time. Table 2 shows the amount of time wasted on a 6-node chain with 1500-byte packets, where the ratio of the summation of all wasted time segments to the whole simulation period gives the percentage of wasted time.

Table 2: Wasted time of a 6-chain.

Node	1	2	3	4	5	6
Wasted time (%)	12.6	2.4	0.3	0.1	0	0

In the following two subsections, we will show the results for the lattice topology, where the injection and back-off problems are still present.

8.2 Lattice with Only Horizontal Flows

The previous section presents how the consecutive packets influence each other in a single chain. To study the effect of inter-flow interference on throughput, we consider a lattice topology. Actually, a random distribution can also be considered for the investigation of inter-flow interference, but we favor the lattice due to its regularity, which simplifies the computation of analytical results. As in the case of the chain, we assume each node is 200 meters away from its east, south, west, and north neighbors. The topology is plotted in figure 9. In this case, every third flow can operate without interference from other flows since the inter-flow distance (600 meters) is greater than the interference range of 550 meters. This simple observation leads to 1/3 of the throughput of the chain topology. As the lattice gets larger, we thus expect the channel utilization to be 1/12 of the channel bandwidth, or $1.6 / 12 = 0.12$ Mbps. However, the simulation results in figure 10 show that the throughput per flow settles at about 0.025 Mbps as the network grows large. This achieved rate is less than our estimated value. The inefficiencies of 802.11 that we find in the chain scenarios are still present. The nodes in the beginning feel less contention and hence inject more packets than those can be handled in the later part of the chain. Moreover, the back-off phenomenon also exists for the same reason we explain in the chain scenario.

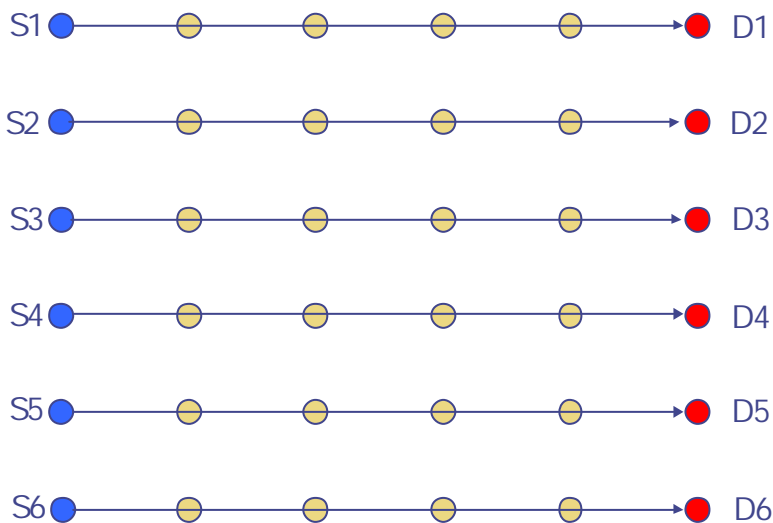


Figure 9: Lattice network topology. Horizontal flows are moving from left to the right.

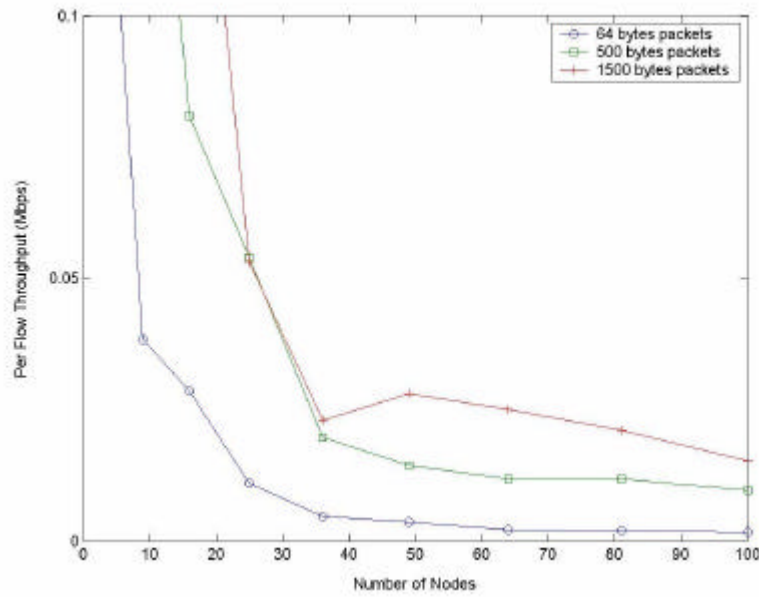


Figure 10: Average throughput per flow in lattice network with only horizontal data streams.

8.3 Lattice with Crossing (Horizontal and Vertical) Flows

Consider the lattice network where both horizontal and vertical flows are present. As shown in Figure 11, the vertical flows are moving from the top to the bottom, and the horizontal flows are moving from the left to the right. To estimate the throughput in this case, we can think in the following way. Ideally, the MAC can schedule a time slot for all vertical flows of the entire network to operate, and the next time slot for all horizontal flows to operate. Since we use the time-division scheme, the schedule will give each flow half as much throughput as in the previous section. Therefore, the estimated throughput is $1/24$ of the available channel bandwidth, that is, $1.6 / 24 = 0.067$ Mbps. The inefficiency of 802.11, however, still occurs in the simulation, so the throughput is mere 0.01 Mbps, as presented in Figure 12. Note that the achieved throughput 0.01 Mbps is about the half of 0.025 Mbps achieved in the lattice topology with only horizontal flows. This shows that CSMA/CA somewhat finds a reasonable way to interleave the two-direction flows but cannot figure out the optimal scheme.

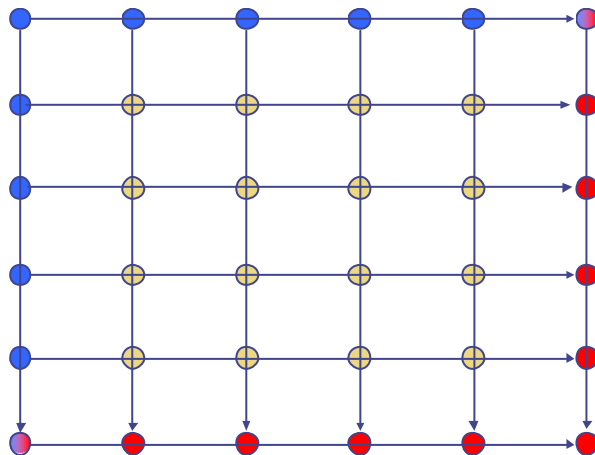


Figure 11: Lattice network topology. Both vertical and horizontal flows are present.

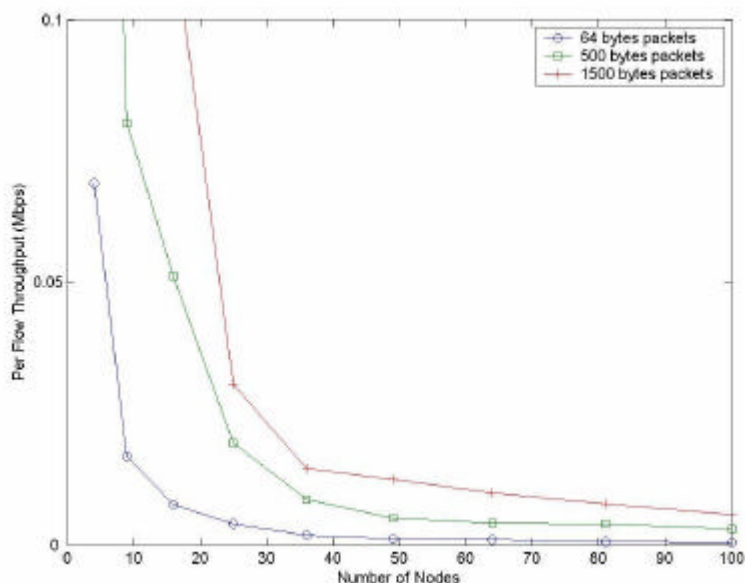


Figure 12: Average throughput per flow in lattice network with only horizontal data streams.

8.4 Power Control

For the previous sections, the inter-nodal distances are identical for the uniformly spaced nodes on a chain or a lattice; so all nodes can simply operate at a common power level. Power control schemes are not necessary for such regularly spaced network topology. In order to study the gain of power control, we consider a chain of nodes where inter-nodal distances are randomly distributed and the connectivity is maintained when the default communication range 250m is used. To obtain the gain of power control, both the schemes with and without power control are performed on the same network topology for comparison.

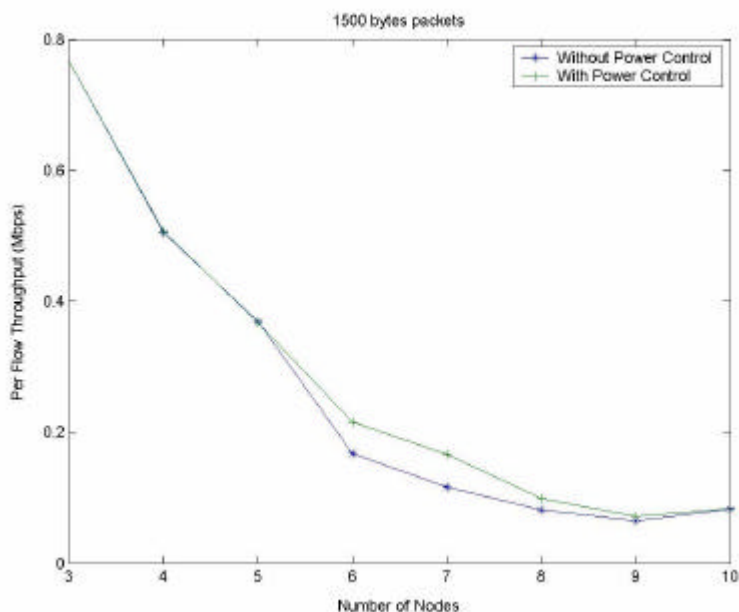


Figure 13: Throughputs of a chain with randomly distributed inter-nodal distances. The scheme of power control gives about 10% throughput gain over the one without power control.

When the power control is performed, a node transmits packets at the power level just enough to reach its neighbors. Figure 13 presents the throughputs of a chain of nodes with and without power control. For the case of 1500-byte packets, the gain that the power control has is about 10%. We also have obtained the throughput gain for the cases of 500-byte and 64-byte packets, and the trends of the curves are similar to the figure shown. However, researchers find that power control can provide up to 50% of the throughput gain [7] [8]. What we obtained is less than the gain in the literature. This is because the topology of our simulation is relatively simple. If a scenario with more intersecting flows is considered, the power control may have significant amount of throughput gain.

9. Future Work

This research of CSMA/CA can be extended to more general topologies that give a better model of real world ad hoc networks. Furthermore, the performance of CSMA can also be studied under more general traffic patterns. That is traffic models that reflect real life patterns and take into account locality of traffic can be used. It would also be instructive to study CSMA/CA assuming TCP flows. In addition, the gains obtained by using power control could be studied in larger networks with many intersecting flows to verify that it does indeed offer a substantial increase in capacity.

10. Conclusion

In this project, we examined the capacity wireless ad hoc networks and the performance of CSMA/CA under different conditions via simulations and analysis from first principles. In particular, we studied 802.11 MAC interactions with ad hoc forwarding and the effect of power control and various packet sizes on CSMA/CA performance.

The ideal capacity of a long chain of nodes in isolation is $\frac{1}{4}$ of the raw channel bandwidth obtainable from the radio. The simulated chain capacity that 802.11 MAC achieves is much less, because the source swamps the immediately downstream nodes where most of the packets are dropped resulting in the later nodes getting starved and operating far below their maximum potential. We find that this trend continues in the case of the lattice topology both in the cases where only horizontal flows are considered and in the case when we consider intersecting flows. We find that, in general, 802.11 does a reasonable job of interleaving intersecting flows, but still achieves a capacity smaller than the theoretical maximum that can be achieved. We also determined that the achieved capacity increases with packet size. This is because, with increased packet size the relative overhead incurred is reduced, as the overhead is a constant notwithstanding the size of the packets.

When a node's transmitted power is controlled to ensure that it transmits the bare minimum necessary to reach its next hop neighbor, we expect that throughput would increase, since a higher number of simultaneous transmissions can be allowed in the same space and they would not interfere since power is controlled. We find that the simulated capacity for a chain of nodes is indeed larger than in the case when power control is not performed, but the increase is marginal. We expect that a much larger gain in capacity would be achieved in more complex topologies and more generalized traffic models with a larger number of competing flows.

We conclude that 802.11 MAC does a reasonable job of scheduling packet transmissions in ad hoc networks. It is capable of transmitting at the optimal rate, but is not capable of finding the optimal schedule on its own. So, in a heavily loaded network it achieves a smaller capacity than that can be achieved, leaving much scope for improvement.

11. Reference

- [1] T. S. Rappaport, A. Annamalai, R. M. Beuhrer, and W. H. Trantor, "Wireless communications: Past events and a future perspective," *IEEE Communications Magazine*, vol. 40, pp. 148-161, May 2002
- [2] R. Knopp and P. Humblet, "Information capacity and power control in single-cell multiuser communications," in *Proceedings of IEEE ICC*, June 1995
- [3] IEEE Standards for Wireless LAN Media Access Control (MAC) and Physical Layer (PHY) Specifications, IEEE Standards 802.11, 1997
- [4] The Network Simulator NS-2. Accessible at <http://www.isi.edu/nsnam/ns>, current as of October 2002
- [5] The Monarch project: Wireless and mobile extensions to ns-2. Accessible at <ftp://ftp.monarch.cs.rice.edu/pub/monarch/wireless-sim/ns-cmu.ps>, current as of October 1998
- [6] J. Li, C. Blake, D. S. J. De Couto, H. I. Lee, and R. Morris, "Capacity of ad hoc wireless networks," *Proc. of ACM SIGMOBILE' 01*, pp. 61 – 69, July 2001
- [7] E. Jung and N. H. Vaidya, "A power control MAC protocol for ad hoc networks," *Proc. Of ACM MOBICOM' 02*, pp. 36 – 47, September 2002
- [8] S. Agarwal, R. H. Katz, S. V. Krishnamurthy, and S. K. Dao, "Distributed power control in ad-hoc wireless networks," *IEEE Symposium on Personal, Indoor, and Mobile Radio Communications*, pp. F59 – F-66 vol. 2, September 2001
- [9] Fang, Yue and McDonald, A.B., "Dynamic Codeword Routing (DCR): A Cross-Layer Approach for Performance Enhancement of General Ad Hoc Routing." To appear in *Proceedings of The First IEEE International Conference of Sensor and Ad Hoc Communications Networks (SECON 2004)*, Santa Clara, CA, October 47, 2004, pp. 255-263.

Appendix

A.1 NS2 Scripts

Our NS2 script files are written in OTcl. There are three scripts written to generate the scenarios of chain, lattice with horizontal flows, and lattice with crossing flows respectively. They are listed in Table A.1. All of them use the protocols and parameters specified in the Simulation Setup section. Each script takes three arguments and outputs a trace file. The three arguments are the name of the trace file, number of nodes on a chain, and packet size in bytes. For example, the following command runs the simulation of a chain of 6 nodes with 500-byte packets and stores the output to the trace file named ‘cap_6n_500.tr’:

```
ns cap.tcl cap_6n_500.tr 6 500
```

It is worth noting that the second argument that lattice scripts take is interpreted as the number of nodes N on a chain. The scripts will then generate a N -by- N lattice. The reason why we interpret this argument as the number of nodes on a chain is the ease of running a shell script to increase the size of the lattice. The shell script can simply use a loop to set the second argument as 2, 3, ..., 10 run the NS2 script with 2-by-2, 3-by-3, ..., 10-by-10 lattice. The following command, for instance, simulates a 5-by-5 lattice with horizontal flows using 1500-byte packets and outputs to the trace file ‘cap_lattice_5n_1500.tr’:

```
ns cap_lattice.tcl cap_lattice_5n_1500.tr 5 1500
```

Table A.1: NS2 Script Files

File Name	Explanation
cap.tcl	Chain of nodes
cap_lattice.tcl	Lattice with horizontal flows
cap_lattice2d.tcl	Lattice with crossing flows

Moreover, we have run the simulation of varying the load offered to a chain and measuring the throughput accordingly. We have also considered a chain with the inter-nodal distances to be randomly distributed. Those files are listed in Table A.2. The scripts with randomly distributed nodes take the same arguments mentioned in the previous paragraph. Note that the offered-load script takes one extra argument: the offered load in bits per second. The following command, for example, simulates a 6node chain using 64-byte packets with offered load 50,000 bps and outputs to the trace file ‘offer_6n_64_50000.tr’:

```
ns cap_offer.tcl offer_6n_64_50000.tr 6 64 50000
```

Table A.2: NS2 Script Files

File Name	Explanation
cap_offer.tcl	Offered load simulation
cap_random.tcl	Randomly distributed nodes on a chain with common power level
cap_random_power.tcl	Randomly distributed nodes on a chain with minimum power level

A.2 UNIX Shell Scripts – Extracting the Throughput Info from Trace Files

In the previous subsection, we have mentioned that NS2 scripts generate trace files. In this section, we will discuss how to get the throughput information from the trace files. The NS2 trace files record each individual packet as it arrives, departs, or is dropped at a link or a queue. It includes information about the time stamp, packet size, packet header, etc. Here is a segment of a trace file generated for a 6-node chain topology with 1500-byte packets. Since we run the simulation for the time duration enough for 100,000 packets, there are more than 100,000 lines in a trace file.

```
s 1.084000000 _0_ AGT --- 14 cbr 1500 [0 0 0 0] ----- [0:0 5:0 32 0] [14] 0 0
r 1.084905521 _5_ AGT --- 0 cbr 1520 [13a 5 4 800] ----- [0:0 5:0 26 5] [0] 5 0
s 1.090000000 _0_ AGT --- 15 cbr 1500 [0 0 0 0] ----- [0:0 5:0 32 0] [15] 0 0
s 1.096000000 _0_ AGT --- 16 cbr 1500 [0 0 0 0] ----- [0:0 5:0 32 0] [16] 0 0
s 1.102000000 _0_ AGT --- 17 cbr 1500 [0 0 0 0] ----- [0:0 5:0 32 0] [17] 0 0
```

As you can see, there are many entries in each line. We will explain only the entries that are useful for extracting the throughput information. The first entry shows the type of the event (‘s’ denotes packet sent and ‘r’ denotes packet received.) The simulation time (in seconds) at which the event occurs is listed in the second column. The third entry indicates the node to which the trace belongs. The fourth entry shows the layer of the event (AGT, denotes the agent, RTR denotes the routing, and MAC denotes the medium access control layer.) The seventh column (‘cbr’ in this example segment) denotes the type of packet seen. The eighth entry is the size of the packet. Following the entry of packet size, there are two sets of data placed in square brackets. Let us look at the ‘0:0 5:0’ in the second brackets. The two pairs of number spaced by the colons denote the IP address and port of the source and destination respectively. As seen in the segment, ‘0:0 5:0’ indicate that the source is at IP address 0 with port 0 and the destination is at IP address 5 with port 0.

In order to compute the throughput, we have to gather all the received events (denoted by ‘r’) at the last node (‘_5_’ in this example) within a certain time duration. We then sum up the sizes of all the received packets and obtain the throughput by dividing the sum by the time duration.

For the topology of a chain of nodes, there is only one traffic-flow. However, there are many traffic-flows for the lattice topology. The throughput we compute for the lattice case is the average over all traffic-flows. Because the number of flows differs for the three scenarios (chain, lattice with horizontal flows, and lattice with crossing flows), we have three scripts for each scenario. They are listed in Table A.3.

Table A.3: UNIX Shell Script Files – Extracting Throughput Info

File Name	Explanation
ns_thru.sh	Chain of nodes
ns_thru_lat.sh	Lattice with horizontal flows
ns_thru_lat2d.sh	Lattice with crossing flows

The scripts take three arguments: the name of the trace file, the number of nodes on a chain, and the packet size. One line of output will be printed on standard output, and the line includes three entries: the number of nodes, the packet size, and the average throughput (in bits per second.) The number of nodes and the packet size are printed along with the throughput.

The following command, for example, parses the trace file ‘lat_4n_500.tr’ to compute the throughput for a 4-by-4 lattice with 500-byte packets.

```
ns_thru_lat.sh lat_4n_500.tr 4 500
```

The output is shown as

```
16 500 80950.00
```

Secondly, for the offered-load simulation and the scenario of the randomly distributed nodes on a chain, the shell scripts are listed in Table A.4. Note that the same script ‘ns_thru.sh’ can be used for all the scenarios with the chain topology whether nodes are randomly or uniformly distributed.

Table A.4: UNIX Shell Script Files – Extracting Throughput Info

File Name	Explanation
ns_offerthru.sh	Offered load simulation
ns_thru.sh	Randomly distributed nodes on a chain with common power level
ns_thru.sh	Randomly distributed nodes on a chain with minimum power level

The script for varying the offered-load takes one extra argument: the offered load in bits per second. An example of running the script is as follows. The load of 50,000 bps is offered to a 6-node chain with 64-byte packets.

```
ns_offerthru.sh offer_6n_1500_50000.tr 6 1500 50000
```

The output has two entries, which denote the offered load and the obtain throughput.

```
50000 49800.00
```

With the knowledge of the sections of NS2 scripts and UNIX shell scripts, one can obtain the throughput by running the shell script after getting the trace file from NS2 simulation. However, it would be a tedious work if one has to manually run NS2 over all the scenarios and then run the shell scripts over all trace files. Therefore, we compose the shell scripts that can automatically run a large set of scenarios and compute the corresponding throughputs. The details are explained in the next section.

A.3 UNIX Shell Scripts – Obtain the Throughputs over a Set of Scenarios

Suppose we want to plot the curve of throughput as a function of the number of nodes. We also want to have three such curves that show the throughput for the flows with packet sizes 64, 500, and 1500 bytes. We have to run the simulation over 2, 3, ..., 10 nodes and three different packet sizes. Totally, there are $9 * 3 = 27$ runs. The work is tedious if all things need to be done manually. Hence we compose the shell scripts that automatically compute the throughputs over a specified set of scenarios. There are three scripts written corresponding to the three network topologies we have.

Table A.5: UNIX Shell Script Files – run a set of scenarios

File Name	Explanation
ns_run_thru.sh	Chain of nodes
ns_run_thru_lat.sh	Lattice with horizontal flows
ns_run_thru_lat2d.sh	Lattice with crossing flows

The scripts take five arguments: the file name of the NS2 script, the starting number, the ending number, the step size, the output file name of the throughput info. The starting number S , ending E , and the step size I work as the for-loop in the programming language.

```
for ( N=S; N<=E, N+=I) {
  run the simulation with the number of nodes on a chain equal to N
}
```

The script runs the NS2 simulation with the number of nodes on a chain equal to N , and takes three different packet sizes 64, 500, and 1500 bytes. The following command, for instance, runs the simulation over a chain of 2, 3, ..., 10 nodes with three different packet sizes:

```
ns_run_thru.sh cap.tcl 2 10 1 resultchain.txt
```

The script has for-loops and runs the NS2 script and the ‘ns_thru.sh’ script inside the loop. The file consisting of the throughput information is thus generated. Basically, each line in the throughput file has the same format as specified in the previous section for the script ‘ns_thru.sh’.

```
2 64 258214.40
2 500 1073300.00
2 1500 1553100.00
3 64 123161.60
3 500 524400.00
3 1500 768600.00
.
.
.
9 64 8960.00
9 500 51000.00
9 1500 79200.00
10 64 12083.20
10 500 46000.00
10 1500 68400.00
```

For the simulation where we vary the offered-load, we want to see the throughput as a function of the offer load given the number of nodes on a chain and the packet size. Hence, we come up with a script ‘ns_offerload.sh’ that takes seven arguments: the file name of the NS2 script, the starting offered load, the ending offered load, the step size of the load, the number of nodes, the packet size, and the output file name of the throughput info. Note that the offered load has to be given in bits per seconds and the packet size to be in bytes. The following command simulates a 8node chain with 1500-byte packets and the offered load ranges from 50,000 bps to 1,000,000 bps with the step size equal to 50,000.

```
ns_offerload.sh cap_offer.tcl 50000 1000000 50000 8 1500 resultofferload.txt
```

The content of the output file ‘resultofferload.txt’ is as follows. Each line is generated by ‘ns_offferthru.sh’, which is explained in the previous section.

```
50000 49800.00
100000 100200.00
150000 150000.00
200000 200400.00
250000 249600.00
300000 83400.00
350000 96000.00
400000 88200.00
450000 93600.00
500000 90600.00
550000 90600.00
600000 76200.00
650000 94800.00
700000 94800.00
750000 105600.00
800000 94800.00
850000 97800.00
900000 101400.00
950000 91200.00
1000000 118200.00
```

Finally, for the randomly distributed nodes on a chain with or without power control, the script ‘ns_run_thru.sh’ is used with the corresponding NS2 script. For example, one can execute the following command to simulate the randomly distributed nodes with power control.

```
ns_run_thru.sh cap_random_power.tcl 2 10 1 resultranpow.txt
```

The content of the ‘resultranpow.txt’ is shown below.

```
2 64 258406.40
2 500 1073000.00
2 1500 1552200.00
3 64 124032.00
3 500 527400.00
3 1500 767400.00
.
.
.
9 64 11187.20
9 500 48600.00
9 1500 72000.00
10 64 10547.20
10 500 50400.00
10 1500 83400.00
```

Using the scripts introduced in this section, the desired throughput information can be obtained very easily. In the following pages, all the files discussed in the appendix will be attached.

cap.tcl (NS2 Script)

```
if { $argc < 3 } {
    puts "Usage:"
    puts "    ns $argv0 trace_file_name number_of_node packet_size"
}
#####
# Define options
#####
set val(chan)          Channel/WirelessChannel    ;# channel type
set val(prop)          Propagation/TwoRayGround   ;# radio-propagationmodel
set val(netif)         Phy/WirelessPhy           ;# network interface type
set val(mac)           Mac/802_11                ;# MAC type
set val(ifq)           Queue/DropTail/PriQueue   ;# interface queue type
set val(ll)            LL                         ;# link layer type
set val(ant)           Antenna/OmniAntenna       ;# antenna model
set val(ifqlen)        500                       ;# max packet in ifq
set val(nn)            [expr [lindex $argv 1] - 1] ;# number of mobilenodes -1
set val(rp)            AODV                       ;# routing protocol
set val(packet_size)   [lindex $argv 2]         ;# (bytes)
setval(cbr_rate)       2e6                       ;# the rate of the CBR (bps)
#set val(run_time)
#set val(end_time)     [expr $val(run_time)+100]
set val(filename)     [lindex $argv 0]

#set the maximum size of UDP packets
Agent/UDP set packetSize_2000

#set bandwidth and datarate for Mac 802.11
Phy/WirelessPhy set bandwidth_2e6
Mac/802_11 set bandwidth_2e6
Mac/802_11 set dataRate_2e6

set ns_                [new Simulator]
set tracefd            [open $val(filename) w]
$ns_ trace-all $tracefd

# set up topography object
set topo               [new Topography]

$topo load_flatgrid 5000 5000

#
# Create God
#
create-god [expr $val(nn) + 1]

proc Initchbr { c myrate packet_size } {
    # $c set rate_ val(cbr_rate)
    # $c set rate_ $myrate
    # puts [$c rate_]
    $c set packetSize_ $packet_size
    $c set interval_ [expr $packet_size*8/$myrate]
}

#
# Create the specified number of mobilenodes [$val(nn)] and "attach" them
# to the channel.
# Here two nodes are created : node(0) and node(1)

# configure node

set c1_ [new $val(chan)]

    $ns_ node-config -adhocRouting $val(rp) \
                    -llType $val(ll) \
                    -macType $val(mac) \
                    -ifqType $val(ifq) \
```

```
-ifqLen$val(ifqlen) \  
-antType$val(ant) \  
-propType$val(prop) \  
-phyType$val(netif) \  
-topoInstance$topo \  
-agentTraceON \  
-routerTraceON \  
-macTraceOFF \  
-movementTraceOFF \  
-channel$c1_  
  
#  
# Provide initial (X,Y, for now Z=0) co-ordinatesformobilenodes  
#  
for {set i 0} {$i <= $val(nn)} {incr i} {  
  set node_($i) [$ns_ node]  
  $node_($i) random-motion 0 ;# disable random motion  
  $node_($i) set X_ [expr $i * 200.0+200]  
  $node_($i) set Y_ 200.0  
  $node_($i) set Z_ 0.0  
  puts "node $i ([expr $i * 200.0+200], 200.0)"  
}  
  
  set sink_($val(nn)) [new Agent/LossMonitor]  
  $ns_ attach-agent $node_($val(nn)) $sink_($val(nn))  
  
  set cbr_(0:$val(nn)) [new Application/Traffic/CBR]  
  Initcbr $cbr_(0:$val(nn)) $val(cbr_rate) $val(packet_size)  
  set udp_(0:$val(nn)) [new Agent/UDP]  
  $ns_ attach-agent $node_(0) $udp_(0:$val(nn))  
  $cbr_(0:$val(nn)) attach-agent $udp_(0:$val(nn))  
  $ns_ connect $udp_(0:$val(nn)) $sink_($val(nn))  
  $ns_ at 1.0 "$cbr_(0:$val(nn)) start"  
  $ns_ at 30.0 "$cbr_(0:$val(nn)) stop"  
  
# Setup traffic flow between nodes  
# TCP connections between node_(0) and node_(1)  
  
#  
# Tell nodes when the simulation ends  
#  
#for {set i 0} {$i < $val(nn)} {incr i} {  
#  $ns_ at 56.9 "$node_($i) reset";  
#}  
$ns_ at 57.0 "stop"  
$ns_ at 57.0 "puts \"NSEXITING...\" ; $ns_ halt"  
proc stop {} {  
  global ns_ tracefd  
  $ns_ flush-trace  
  close $tracefd  
}  
  
puts "Starting Simulation..."  
$ns_ run
```

cap_lattice.tcl (NS2 Script)

```

if { $argc < 3 } {
    puts "Usage:"
    puts "  ns $argv0 trace_file_name number_of_node_on_a_chain packet_size"
}
#####
# Define options
#####
set val(chan)      Channel/WirelessChannel    ;# channel type
set val(prop)      Propagation/TwoRayGround   ;# radio-propagationmodel
set val(netif)     Phy/WirelessPhy           ;# network interface type
set val(mac)       Mac/802_11                ;# MAC type
set val(ifq)       Queue/DropTail/PriQueue   ;# interface queue type
set val(ll)        LL                        ;# link layer type
set val(ant)       Antenna/OmniAntenna       ;# antenna model
set val(ifqlen)    500                      ;# max packet in ifq
set val(nn)        [expr [lindex $argv 1] - 1] ;# number of mobilenodes on a chain - 1
set val(rp)        AODV                      ;# routing protocol
set val(packet_size) [lindex $argv 2]        ;# (bytes)
set val(cbr_rate)  2e6                      ;# the rate of the CBR (bps)
#set val(run_time)
#set val(end_time) [expr $val(run_time)+100]
set val(filename) [lindex $argv 0]

#set the maximum size of UDP packets
Agent/UDP set packetSize_ 2000

#set bandwidth and datarate for Mac 802.11
Phy/WirelessPhy set bandwidth_ 2e6
Mac/802_11 set bandwidth_ 2e6
Mac/802_11 set dataRate_ 2e6

#
# Initialize Global Variables
#
set ns_ [new Simulator]
set tracefd [open $val(filename) w]
$ns_ trace-all $tracefd

# set up topography object
set topo [new Topography]

$topo load_flatgrid 5000 5000

#
# Create God
#
create-god [expr ($val(nn) + 1)*($val(nn) + 1)]

proc Initscbr { c myrate packet_size } {
    # $c set rate_ val(cbr_rate)
    # $c set rate_ $myrate
    # puts [$c rate_]
    $c set packetSize_ $packet_size
    $c set interval_ [expr $packet_size*8/$myrate]
}

#
# Create the specified number of mobilenodes [$val(nn)] and "attach" them
# to the channel.
# Here two nodes are created : node(0) and node(1)

# configure node

set c1_ [new $val(chan)]

$ns_ node-config -adhocRouting $val(rp) \
                -llType $val(ll) \

```

```
-macType$val(mac) \  
-ifqType$val(ifq) \  
-ifqLen$val(ifqlen) \  
-antType$val(ant) \  
-propType$val(prop) \  
-phyType$val(netif) \  
-topoInstance$topo \  
-agentTrace ON \  
-routerTraceON \  
-macTraceOFF \  
-movementTraceOFF \  
-channel$c1_  
  
#  
# Provide initial (X,Y, for now Z=0) co-ordinatesformobilenodes  
#  
  
for {set i 0} {$i <= $val(nn)} {incr i} {  
  for {set k 0} {$k <= $val(nn)} {incr k} {  
    set node_($i:$k) [$ns_node]  
    $node_($i:$k) random-motion 0      ;# disable random motion  
    $node_($i:$k) set X_ [expr $i * 200.0+200]  
    $node_($i:$k) set Y_ [expr $k * 200.0+200]  
    $node_($i:$k) set Z_ 0.0  
    puts "node_($i:$k) ([expr $i * 200.0+200], [expr $k * 200.0+200])"  
  }  
}  
  
for {set row 0} {$row <= $val(nn)} {incr row} {  
  setsink_($row) [newAgent/LossMonitor]  
  $ns_attach-agent$node_($val(nn):$row)$sink_($row)  
  
  setcbr_($row) [newApplication/Traffic/CBR]  
  Initcbr$cbr_($row)$val(cbr_rate)$val(packet_size)  
  setudp_($row) [newAgent/UDP]  
  $ns_attach-agent$node_(0:$row)$udp_($row)  
  $cbr_($row) attach-agent$udp_($row)  
  $ns_connect$udp_($row)$sink_($row)  
  $ns_at 1.0 "$cbr_($row) start"  
  $ns_at 11.0 "$cbr_($row) stop"  
}  
  
# Setup traffic flow between nodes  
# TCP connections between node_(0) and node_(1)  
  
#  
# Tell nodes when the simulation ends  
#  
#for {set i 0} {$i < $val(nn)} {incr i} {  
#  $ns_at 56.9 "$node_($i) reset";  
#}  
$ns_at 12.0 "stop"  
$ns_at 12.0 "puts \"NSEXITING...\" ; $ns_halt"  
proc stop {} {  
  global ns_tracefd  
  $ns_flush-trace  
  close$tracefd  
}  
  
puts "StartingSimulation..."  
$ns_run
```

cap_lattice2d.tcl (NS2 Script)

```

if { $argc < 3 } {
    puts "Usage:"
    puts "    ns $argv0 trace_file_name number_of_node_on_a_chain packet_size"
}
#####
# Define options
#####
set val(chan)          Channel/WirelessChannel    ;# channel type
set val(prop)          Propagation/TwoRayGround   ;# radio-propagationmodel
set val(netif)         Phy/WirelessPhy           ;# network interface type
set val(mac)           Mac/802_11                ;# MAC type
set val(ifq)           Queue/DropTail/PriQueue   ;# interface queue type
set val(ll)           LL                          ;# link layer type
set val(ant)           Antenna/OmniAntenna       ;# antenna model
set val(ifqlen)        500                       ;# max packet in ifq
set val(nn)            [expr [lindex $argv 1] - 1] ;# number of mobilenodes on a chain - 1
set val(rp)           AODV                       ;# routing protocol
set val(packet_size)   [lindex $argv 2]         ;# (bytes)
set val(cbr_rate)      2e6                       ;# the rate of the CBR (bps)
#set val(run_time)
#set val(end_time)     [expr $val(run_time)+100]
set val(filename)     [lindex $argv 0]

#set the maximum size of UDP packets
Agent/UDPsetpacketSize_ 2000

#set bandwidth and datarate for Mac 802.11
Phy/WirelessPhysetbandwidth_ 2e6
Mac/802_11 set bandwidth_ 2e6
Mac/802_11 set dataRate_ 2e6

#
# Initialize Global Variables
#
set ns_ [new Simulator]
set tracefd [open $val(filename) w]
$ns_trace-all $tracefd

# set up topography object
set topo [new Topography]

$topo load_flatgrid 5000 5000

#
# Create God
#
create-god [expr ($val(nn) + 1)*($val(nn) + 1)]

proc Initchbr { c myrate packet_size } {
    # $c set rate_ val(cbr_rate)
    # $c set rate_ $myrate
    # puts [$c rate_]
    $c set packetSize_ $packet_size
    $c set interval_ [expr $packet_size*8/$myrate]
}

#
# Create the specified number of mobilenodes [$val(nn)] and "attach" them
# to the channel.
# Here two nodes are created : node(0) and node(1)

# configure node

set c1_ [new $val(chan)]

    $ns_node-config -adhocRouting$val(rp) \
                    -llType$val(ll) \
                    -macType$val(mac) \
                    -ifqType$val(ifq) \
                    -ifqLen$val(ifqlen) \

```

```

        -antType$val(ant) \
        -propType$val(prop) \
        -phyType$val(netif) \
        -topoInstance$topo \
        -agentTraceON \
        -routerTraceON \
        -macTraceON \
        -movementTraceOFF \
        -channel$c1_

#generate NODES and set their coordinates
for {set i 0} {$i <= $val(nn)} {incr i} {
    for {set k 0} {$k <= $val(nn)} {incr k} {
        set node_($i:$k) [$ns_node]
        $node_($i:$k) random-motion 0 ;# disable random motion
        $node_($i:$k) set X_ [expr $i * 200+200]
        $node_($i:$k) set Y_ [expr $k * 200 + 200]
        $node_($i:$k) set Z_ 0.0
        puts "node_($i:$k) ($$ranX_($i), $$ranY_($k))"
    }
}

#
# Provide initial (X,Y, for now Z=0) co-ordinatesformobilenodes

for {set row 0} {$row <= $val(nn)} {incr row} {
    setsink_($row) [newAgent/LossMonitor]
    $ns_attach-agent $node_($val(nn):$row)$sink_($row)

    setcbr_($row) [newApplication/Traffic/CBR]
    Initcbr$cbr_($row) $val(cbr_rate) $val(packet_size)
    setudp_($row) [newAgent/UDP]
    $ns_attach-agent $node_(0:$row)$udp_($row)
    $cbr_($row) attach-agent $udp_($row)
    $ns_connect $udp_($row) $sink_($row)
    $ns_at 1.0 "$cbr_($row) start"
    $ns_at 11.0 "$cbr_($row) stop"
}

for {set col 0} {$col <= $val(nn)} {incr col} {
    setsink2_($col) [newAgent/LossMonitor]
    $ns_attach-agent $node_($col:$val(nn))$sink2_($col)

    setcbr2_($col) [newApplication/Traffic/CBR]
    Initcbr$cbr2_($col) $val(cbr_rate) $val(packet_size)
    setudp2_($col) [newAgent/UDP]
    $ns_attach-agent $node_($col:0)$udp2_($col)
    $cbr2_($col) attach-agent $udp2_($col)
    $ns_connect $udp2_($col) $sink2_($col)
    $ns_at 1.0 "$cbr2_($col) start"
    $ns_at 11.0 "$cbr2_($col) stop"
}

$ns_at 12.0 "stop"
$ns_at 12.0 "puts \"NSEXITING...\" ; $ns_halt"
proc stop {} {
    global ns_tracefd
    $ns_flush-trace
    close$tracefd
}

puts "Starting Simulation..."
$ns_run

```

cap_offer.tcl (NS2 Script)

```

if { $argc < 3 } {
    puts "Usage:"
    puts "  ns $argv0 trace_file_name number_of_node packet_size offered_load(bps)"
}
#####
# Defineoptions
#####
set val(chan)          Channel/WirelessChannel    ;# channel type
set val(prop)          Propagation/TwoRayGround   ;# radio-propagationmodel
set val(netif)         Phy/WirelessPhy           ;# network interface type
set val(mac)           Mac/802_11                ;# MAC type
set val(ifq)           Queue/DropTail/PriQueue   ;# interface queue type
set val(ll)           LL                          ;# link layer type
set val(ant)           Antenna/OmniAntenna       ;# antenna model
set val(ifqlen)        500                       ;# max packet in ifq
set val(nn)            [expr [lindex $argv 1] - 1] ;# number of mobilenodes -1
set val(rp)            AODV                       ;# routing protocol
set val(packet_size)   [lindex $argv 2]         ;# (bytes)
set val(cbr_rate)      [lindex $argv 3]         ;# the rate of the CBR (bps)
#set val(run_time)
#set val(end_time)     [expr $val(run_time)+100]
set val(filename)      [lindex $argv 0]

#set the maximum size of UDP packets
Agent/UDP set packetSize_ 2000

#set bandwidth and datarate for Mac 802.11
Phy/WirelessPhy set bandwidth_ 2e6
Mac/802_11 set bandwidth_ 2e6
Mac/802_11 set dataRate_ 2e6

set ns_                [new Simulator]
set tracefd            [open $val(filename) w]
$ns_ trace-all $tracefd

# set up topography object
set topo               [new Topography]

$topo load_flatgrid 5000 5000

#
# Create God
#
create-god [expr $val(nn) + 1]

proc Initscbr { c myrate packet_size } {
    # $c set rate_ val(cbr_rate)
    # $c set rate_ $myrate
    # puts [$c rate_]
    $c set packetSize_ $packet_size
    $c set interval_ [expr $packet_size*8.0/$myrate]
}

#
# Create the specified number of mobilenodes [$val(nn)] and "attach" them
# to the channel.
# Here two nodes are created : node(0) and node(1)

# configure node

set c1_ [new $val(chan)]

    $ns_ node-config -adhocRouting $val(rp) \
                    -llType $val(ll) \
                    -macType $val(mac) \
                    -ifqType $val(ifq) \
                    -ifqLen $val(ifqlen) \

```

```
-antType$val(ant) \  
-propType$val(prop) \  
-phyType$val(netif) \  
-topoInstance$topo \  
-agentTraceON \  
-routerTraceOFF \  
-macTraceOFF \  
-movementTraceOFF \  
-channel$c1_  
  
#  
# Provide initial (X,Y, for now Z=0) co-ordinatesformobilenodes  
#  
for {set i 0} {$i <= $val(nn)} {incr i} {  
  set node_($i) [$ns_ node]  
  $node_($i) random-motion 0          ;# disable random motion  
  $node_($i) set X_ [expr $i * 200.0+200]  
  $node_($i) set Y_ 200.0  
  $node_($i) set Z_ 0.0  
  puts "node $i ([expr $i * 200.0+200], 200.0)"  
}  
  
  setsink_($val(nn))[newAgent/LossMonitor]  
  $ns_ attach-agent $node_($val(nn)) $sink_($val(nn))  
  
  setcbr_(0:$val(nn))[newApplication/Traffic/CBR]  
  Initcbr$cbr_(0:$val(nn)) $val(cbr_rate) $val(packet_size)  
  setudp_(0:$val(nn))[newAgent/UDP]  
  $ns_ attach-agent $node_(0) $udp_(0:$val(nn))  
  $cbr_(0:$val(nn)) attach-agent $udp_(0:$val(nn))  
  $ns_ connect $udp_(0:$val(nn)) $sink_($val(nn))  
  $ns_ at 1.0 "$cbr_(0:$val(nn)) start"  
  $ns_ at 30.0 "$cbr_(0:$val(nn)) stop"  
  
# Setup traffic flow between nodes  
# TCP connections between node_(0) and node_(1)  
  
#  
# Tell nodes when the simulation ends  
#  
#for {set i 0} {$i < $val(nn)} {incr i} {  
#  $ns_ at 56.9 "$node_($i) reset";  
#}  
$ns_ at 57.0 "stop"  
$ns_ at 57.0 "puts \"NSEXITING...\" ; $ns_ halt"  
proc stop {} {  
  global ns_ tracefd  
  $ns_ flush-trace  
  close$tracefd  
}  
  
puts "StartingSimulation..."  
$ns_ run
```

cap_random.tcl (NS2 Script)

```

if { $argc < 3 } {
    puts "Usage:"
    puts "    ns $argv0 trace_file_name number_of_node packet_size"
}
#####
# Define options
#####
set val(chan)           Channel/WirelessChannel    ;# channel type
set val(prop)           Propagation/TwoRayGround   ;# radio-propagationmodel
set val(netif)          Phy/WirelessPhy           ;# network interface type
set val(mac)            Mac/802_11                ;# MAC type
set val(ifq)            Queue/DropTail/PriQueue   ;# interface queue type
set val(ll)            LL                          ;# link layer type
set val(ant)            Antenna/OmniAntenna       ;# antenna model
set val(ifqlen)         500                       ;# max packet in ifq
set val(nn)             [expr [lindex $argv 1] - 1] ;# number of mobilenodes -1
set val(rp)            AODV                       ;# routing protocol
set val(packet_size)    [lindex $argv 2]          ;# (bytes)
set val(cbr_rate)       2e6                       ;# the rate of the CBR (bps)
#set val(run_time)
#set val(end_time)      [expr $val(run_time)+100]
set val(filename)       [lindex $argv 0]
set val(tx_dist)        400

#set the maximum size of UDP packets
Agent/UDP set packetSize_2000

#set bandwidth and datarate for Mac 802.11
Phy/WirelessPhy set bandwidth_2e6
Mac/802_11 set bandwidth_2e6
Mac/802_11 set dataRate_2e6

set ns_                 [new Simulator]
set tracefd             [open $val(filename) w]
$ns_trace-all $tracefd

# set up topography object
set topo                [new Topography]

$topo load_flatgrid 5000 5000

#
# Create God
#
create-god [expr $val(nn) + 1]

proc Initscbr { c myrate packet_size } {
    $c set packetSize_ $packet_size
    $c set interval_ [expr $packet_size*8/$myrate]
}

proc CalTxPow { dist node } {
    set distance [expr $dist + 5]
    set phyObj [ $node set netif_(0)]
    $phyObj set Pt_ [expr 71.6049e-12 * $distance * $distance * $distance * $distance]
}

#
# Create the specified number of mobilenodes [$val(nn)] and "attach" them
# to the channel.
# Here two nodes are created : node(0) and node(1)

# configure node

set c1_ [new $val(chan)]

    $ns_node-config -adhocRouting $val(rp) \
                    -llType $val(ll) \
                    -macType $val(mac) \
                    -ifqType $val(ifq) \
                    -ifqLen $val(ifqlen) \

```

```
-antType$val(ant) \  
-propType$val(prop) \  
-phyType$val(netif) \  
-topoInstance$topo \  
-agentTraceON \  
-routerTraceON \  
-macTraceOFF \  
-movementTraceOFF \  
-channel$c1_  
  
globaldefaultRNG  
$defaultRNGseed9999  
  
set ranX_(0) 0.0  
set ranX_(1) [$defaultRNGuniform100 $val(tx_dist)]  
for {set i 2} {$i <= $val(nn)} {incr i} {  
  set first [expr $i - 2]  
  set second [expr $i - 1]  
  set low_bound [expr $val(tx_dist)-($ranX_($second) - $ranX_($first))]  
  if { $low_bound < 100 } { set low_bound 100 }  
  set ranX_($i) [expr $ranX_($second) + [$defaultRNGuniform$low_bound $val(tx_dist)]]  
}  
  
for {set i 0} {$i <= $val(nn)} {incr i} {  
  set node_($i) [ $ns_node]  
  $node_($i) random-motion 0           ;# disable random motion  
  $node_($i) set X_ $ranX_($i)  
  $node_($i) set Y_ 200.0  
  $node_($i) set Z_ 0.0  
  puts "node_($i) - ($ranX_($i), 200)"  
  CalTxPow$val(tx_dist) $node_($i)  
}  
  
set sink_($val(nn)) [newAgent/LossMonitor]  
$ns_attach-agent $node_($val(nn)) $sink_($val(nn))  
  
set cbr_(0:$val(nn)) [newApplication/Traffic/CBR]  
Initcbr $cbr_(0:$val(nn)) $val(cbr_rate) $val(packet_size)  
set udp_(0:$val(nn)) [newAgent/UDP]  
$ns_attach-agent $node_(0) $udp_(0:$val(nn))  
$cbr_(0:$val(nn)) attach-agent $udp_(0:$val(nn))  
$ns_connect $udp_(0:$val(nn)) $sink_($val(nn))  
$ns_at 1.0 "$cbr_(0:$val(nn)) start"  
$ns_at 15.0 "$cbr_(0:$val(nn)) stop"  
  
$ns_at 57.0 "stop"  
$ns_at 57.0 "puts \"NSEXITING...\" ; $ns_halt"  
proc stop {} {  
  global ns_tracefd  
  $ns_flush-trace  
  close $tracefd  
}  
  
puts "StartingSimulation..."  
$ns_run
```

cap_random_power.tcl (NS2 Script)

```

if { $argc < 3 } {
    puts "Usage:"
    puts "    ns $argv0 trace_file_name number_of_node packet_size"
}
#####
# Define options
#####
set val(chan)           Channel/WirelessChannel    ;# channel type
set val(prop)           Propagation/TwoRayGround   ;# radio-propagationmodel
set val(netif)          Phy/WirelessPhy           ;# network interface type
set val(mac)            Mac/802_11                ;# MAC type
setval(ifq)             Queue/DropTail/PriQueue    ;# interface queue type
set val(ll)            LL                          ;# link layer type
set val(ant)            Antenna/OmniAntenna       ;# antenna model
set val(ifqlen)         500                       ;# max packet in ifq
set val(nn)             [expr [lindex $argv 1] - 1] ;# number of mobilenodes -1
set val(rp)            AODV                       ;# routing protocol
set val(packet_size)    [lindex $argv 2]         ;# (bytes)
set val(cbr_rate)       1e6                       ;# the rate of the CBR (bps)
#set val(run_time)
#set val(end_time)     [expr $val(run_time)+100]
set val(filename)      [lindex $argv 0]
set val(tx_dist)       400

#set the maximum size of UDP packets
Agent/UDP set packetSize_2000

#set bandwidth and datarate for Mac 802.11
Phy/WirelessPhy set bandwidth_2e6
Mac/802_11 set bandwidth_2e6
Mac/802_11 set dataRate_2e6

set ns_                [new Simulator]
set tracefd            [open $val(filename) w]
$ns_trace-all $tracefd

# set up topography object
set topo               [new Topography]

$topo load_flatgrid 5000 5000

#
# Create God
#
create-god [expr $val(nn) + 1]

proc Inittcbr { c myrate packet_size } {
    $c set packetSize_ $packet_size
    $c set interval_ [expr $packet_size*8/$myrate]
}

proc CalTxPow { distance1 distance2 node } {
    if { $distance1 > $distance2 } {
        set maxdist $distance1
    } else {
        set maxdist $distance2
    }
    set dist_ [expr $maxdist + 5]
    set phyObj [ $node set netif_(0) ]
    set tmpPt_ [expr 71.6049e-12 * $dist_ * $dist_ * $dist_ * $dist_]
    $phyObj set Pt_ $tmpPt_
    puts "Pt_ of $node is $tmpPt_"
}

proc CalTxPow1 { d1 node } {
    CalTxPow $d1 $d1 $node
}

# configure node

```

```

set c1_ [new $val(chan)]

    $ns_node-config -adhocRouting$val(rp) \
                    -llType$val(ll) \
                    -macType $val(mac) \
                    -ifqType$val(ifq) \
                    -ifqLen$val(ifqlen) \
                    -antType$val(ant) \
                    -propType$val(prop) \
                    -phyType$val(netif) \
                    -topoInstance$topo \
                    -agentTraceON \
                    -routerTraceON \
                    -macTraceOFF \
                    -movementTraceOFF \
                    -channel$c1_

globaldefaultRNG
$defaultRNGseed9999

set ranX_(0) 0.0
set ranX_(1) [$defaultRNGuniform100 $val(tx_dist)]
for {set i 2} {$i <= $val(nn)} {incr i} {
    set first [expr $i - 2]
    set second [expr $i - 1]
    set low_bound [expr $val(tx_dist)-($ranX_($second) - $ranX_($first))]
    if { $low_bound < 100 } { set low_bound 100 }
    set ranX_($i) [expr $ranX_($second) + [$defaultRNGuniform$low_bound $val(tx_dist)]]
}

for {set i 0} {$i <= $val(nn)} {incr i} {
    set node_($i) [$ns_node]
    $node_($i) random-motion 0    ;# disable random motion
    $node_($i) set X_ $ranX_($i)
    $node_($i) set Y_ 200.0
    $node_($i) set Z_ 0.0
    puts "node_($i) - ($ranX_($i), 200)"
}
#the following lines are for the power control
CalTxPow1[expr $ranX_(1) - $ranX_(0)]$node_(0)
CalTxPow1[expr $ranX_($val(nn)) - $ranX_([expr $val(nn)-1])]$node_($val(nn))
for {set i 1} {$i < $val(nn)} {incr i} {
    CalTxPow [expr $ranX_([expr $i + 1]) - $ranX_($i)] [expr $ranX_($i) - $ranX_([expr $i - 1])] $node_($i)
}

    setsink_($val(nn))[newAgent/LossMonitor]
    $ns_attach-agent$node_($val(nn))$sink_($val(nn))

    setcbr_(0:$val(nn))[newApplication/Traffic/CBR]
    Initcbr$cbr_(0:$val(nn))$val(cbr_rate)$val(packet_size)
    set udp_(0:$val(nn))[newAgent/UDP]
    $ns_attach-agent$node_(0)$udp_(0:$val(nn))
    $cbr_(0:$val(nn))attach-agent$udp_(0:$val(nn))
    $ns_connect$udp_(0:$val(nn))$sink_($val(nn))
    $ns_at 1.0 "$cbr_(0:$val(nn)) start"
    $ns_at 15.0 "$cbr_(0:$val(nn)) stop"

$ns_at 57.0 "stop"
$ns_at 57.0 "puts \"NSEXITING...\" ; $ns_halt"
proc stop {} {
    global ns_tracefd
    $ns_flush-trace
    close$tracefd
}

puts "StartingSimulation..."
$ns_run

```

ns_thru.sh (UNIX Shell Script)

```
# Input:
# $1 : trace filename,
# $2 : number of nodes (node's number go from 0 to $2 - 1 ),
# $3 : packet size
# Output:
# Number_of_nodes Packet_size Throughput_at_the_last_node
lastnode=`echo $2 | awk '{ printf "%d", $1 - 1 ; }'`
n_recv_packet=`grep AGT $1 | grep " $lastnode:" | grep '^r' | awk '{if ($2 > 4.0 && $2 < 14.0) print $0;}' | grep "_$lastnode" | wc -l`
echo $2 $3 `echo $n_recv_packet $3 | awk '{ printf "%.2f", $1 * $2 * 8.0 / 10.0; }'`
```

ns_thru_lat.sh (UNIX Shell Script)

```
# Input:
# $1 : trace filename,
# $2 : number of nodes on a chain (node's number go from 0 to $2 - 1 ),
# $3 : packet size
# Output:
# Number_of_nodes Packet_size Throughput_at_the_last_node
for r in `echo $2 | awk '{ for(i=0; i< $1 ; i++) printf "%d ", i ; }'`
do
flowsrc=$r
flowdst=`echo $r $2 | awk '{ printf "%d", ($2)*($2 - 1) + $1 ; }'`
#lastnode=`echo $2 | awk '{ printf "%d", $1 - 1 ; }'`
n_recv_packet=`grep AGT $1 | grep "[[]$flowsrc:. $flowdst:" | grep '^r' | awk '{if ($2 > 3.0 && $2 < 9.0) print $0;}' | grep "_$flowdst" | wc -l`
total_thru=`echo $n_recv_packet $3 $total_thru | awk '{ printf "%.2f", $1 * $2 * 8.0 / 6.0 + $3 ; }'`
#Use the following two lines to show extra information
#
#echo "flowsrc=$flowsrc, flowdst=$flowdst, n_recv_packet=$n_recv_packet"
#echo `echo $n_recv_packet $3 | awk '{ printf "%.2f", $1 * $2 * 8.0 / 20.0; }'`
done
echo `echo $total_thru $2 $3 | awk '{printf "%d %d %.2f", ($2)*($2), $3, $1 / $2; }'`
```

ns_thru_lat2d.sh (UNIX Shell Script)

```
# Input:
# $1 : trace file name,
# $2 : number of nodes on a chain (node's number go from 0 to $2 - 1 ),
# $3 : packet size
# Output:
# Number_of_nodes Packet_size Throughput_at_the_last_node
for r in `echo $2 | awk '{ for(i=0; i < $1 ; i++) printf "%d ", i ; }'`
do
flowsrc=$r
flowdst=`echo $r $2 | awk '{ printf "%d", ($2)*($2 - 1) + $1 ; }'`
#lastnode=`echo $2 | awk '{ printf "%d", $1 - 1 ; }'`
n_recv_packet=`grep AGT $1 | grep "[[]$flowsrc:. $flowdst:" | grep '^r' | awk '{if ($2 >
2.0 && $2 < 4.5) print $0;}' | grep "_$flowdst" | wc -l`
total_thru=`echo $n_recv_packet $3 $total_thru | awk '{ printf "%.2f", $1 * $2 * 8.0 / 2.5
+ $3 ; }'`
#Use the following two lines to show extra information
#
#echo "flowsrc=$flowsrc, flowdst=$flowdst, n_recv_packet=$n_recv_packet"
#echo `echo $n_recv_packet $3 | awk '{ printf "%.2f", $1 * $2 * 8.0 / 20.0; }'`
done

for r in `echo $2 | awk '{ for(i=0; i < $1 ; i++) printf "%d ", i ; }'`
do
flowsrc=`echo $r $2 | awk '{ printf "%d", $1 * $2 ; }'`
flowdst=`echo $flowsrc $2 | awk '{ printf "%d", $1 + $2 - 1 ; }'`
n_recv_packet=`grep AGT $1 | grep "[[]$flowsrc:. $flowdst:" | grep '^r' | awk '{if ($2 >
2.0 && $2 < 4.5) print $0;}' | grep "_$flowdst" | wc -l`
total_thru=`echo $n_recv_packet $3 $total_thru | awk '{ printf "%.2f", $1 * $2 * 8.0 / 2.5
+ $3 ; }'`
done

echo `echo $total_thru $2 $3 | awk '{printf "%d %d %.2f", ($2)*($2), $3, $1 / ($2 *
2); }'`
```

ns_offerthru.sh (UNIX Shell Script)

```
# Input:
# $1 : trace file name, $2 : number of nodes (node's number go from 0 to $2 - 1 ), $3 :
packet size, $4 : offered load(bps)
# Output:
# Offered_load Throughput_at_the_last_node
lastnode=`echo $2 | awk '{ printf "%d", $1 - 1 ; }'`
n_recv_packet=`grep AGT $1 | grep "[[]0:. $lastnode:" | grep '^r' | awk '{if ($2 > 5.0 &&
$2 < 25.0) print $0;}' | grep "_$lastnode" | wc -l`
echo $4 `echo $n_recv_packet $3 | awk '{ printf "%.2f", $1 * $2 * 8.0 / 20.0; }'`
```

ns_run_thru.sh (UNIX Shell Script)

```
#Input
# $1 : tcl file name, $2 : starting number, $3 : ending number, $4 : step of number,
# $5 : thruput file name
#Output
# Save the output to $5, all lines have the same format. It is described as follows:
#Number_of_NodesPacket_size_in_bytesThroughput_at_the_last_node(bps)

#erase the thruput file if it exists
#if test -f $5
#then
# rm -f $5
#fi

#run over all number of nodes & 3 scenarios of packet sizes
for i in `echo $2 $3 $4 | awk '{ for(i=$1; i<=$2; i+=3) printf "%d ",i; }' `
do
  for k in 64 150 1500
  do
    t1=cap_$i
    t2=n_$k
    t3=B.tr
    trfile=$t1$t2$t3
    ns $1 $trfile $i $k
    ./ns_thru.sh $trfile $i $k >> $5
    rm -f $trfile
  done
done
```

ns_run_thru_lat.sh (UNIX Shell Script)

```
#Input
# $1 : tcl file name, $2 : starting number, $3 : ending number, $4 : step of number,
# (number of nodes on a chain)
# $5 : thruput file name
#Output
# Save the output to $5, all lines have the same format. It is described as follows:
#Number_of_NodesPacket_size_in_bytesThroughput_at_the_last_node(bps)

#erase the thruput file if it exists
#if test -f $5
#then
# rm -f $5
#fi

#run over all number of nodes & 3 scenarios of packet sizes
for i in `echo $2 $3 $4 | awk '{ for(i=$1; i<=$2; i+=3) printf "%d ",i; }' `
do
  for k in 64 500 1500
  do
    t1=cap_$i
    t2=n_$k
    t3=B_lat.tr
    trfile=$t1$t2$t3
    ns $1 $trfile $i $k
    ./ns_thru_lat.sh $trfile $i $k >> $5
    rm -f $trfile
  done
done
```

ns_run_thru_lat2d.sh (UNIX Shell Script)

```
#Input
# $1 : tcl file name, $2 : starting number, $3 : ending number, $4 : step of number,
#      (number of nodes on a chain)
# $5 : thruput file name
#Output
# Save the output to $5, all lines have the same format. It is described as follows:
#Number_of_NodesPacket_size_in_bytesThroughput_at_the_last_node(bps)

#erase the thruput file if it exists
#if test -f $5
#then
# rm -f $5
#fi

#run over all number of nodes & 3 scenarios of packet sizes
for i in `echo $2 $3 $4 | awk '{ for(i=$1; i<=$2; i+=$3) printf "%d ",i; }' `
do
# for k in 64 500 1500
  for k in 1500
  do
    t1=cap_$i
    t2=n_$k
    t3=B_lat2.tr
    trfile=$t1$t2$t3
    ns $1 $trfile $i $k
    ./ns_thru_lat2.sh $trfile $i $k >> $5
    rm -f $trfile
  done
done
```

ns_offerload.sh (UNIX Shell Script)

```
#Input
# $1 : tcl file name,
# $2 : starting load(bps),
# $3 : ending load(bps),
# $4 : stepsize of load(bps),
# $5 : number of nodes
# $6 : packet size (bytes)
# $7 : thruput file name
#Output
# Save the output to $7, all lines have the same format. It is described as follows:
#Offered_load_at_the_first_node(bps)Throughput_at_the_last_node(bps)

#erase the thruput file if it exists
#if test -f $5
#then
# rm -f $5
#fi

#run over all offered load
for i in `echo $2 $3 $4 | awk '{ for(i=$1; i<=$2; i+=$3) printf "%d ",i; }' `
do
  t1=offer$i
  t2=_$5
  t3=n_$6
  t4=B.tr
  trfile=$t1$t2$t3$t4
  ns $1 $trfile $5 $6 $i
  ./ns_offerthru.sh $trfile $5 $6 $i >> $7
  rm -f $trfile
done
```