DOKUZ EYLÜL UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

COMPARING RED AND BLUE ALGORITHMS IN NS2

by
Delgermaa KHISHGEE

March, 2013
İZMİR

COMPARING RED AND BLUE ALGORITHMS IN NS2

A Thesis Submitted to the
Graduate School of Natural and Applied Sciences of Dokuz Eylül University
In Partial Fulfillment of the Requirements for the Degree of Master of Science
in Computer Engineering, Computer Engineering Program

by
Delgermaa KHISHGEE

March, 2013
İZMİR

M.Sc THESIS EXAMINATION RESULT FORM

We have read the thesis entitled **"COMPARING RED AND BLUE ALGORITHMS IN NS2"** completed by **DELGERMAA KHISHGEE** under supervision of **DR. M. KEMAL ŞİŞ** and we certify that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

[illegible]

Dr. M. KEMAL ŞİŞ

Supervisor

[illegible]

Asst. Prof. Dr. A. ALPKOÇAK

(Jury Member)

[illegible]

Prof. Dr. Haldun Karaca

(Jury Member)

[illegible]

Prof. Dr. Ayşe OKUR

Director

Graduate School of Natural and Applied Sciences

ACKNOWLEDGMENTS

I am really grateful to Dr. M. Kemal ŞİŞ, my research supervisor, for his valuable and constructive suggestions during the planning and development of this research work.

It would be harder without students of Ph.D. to complete the project who offered me the resources regarding to my project and showed me the shortest way.

Finally, I wish to thank my mother for her support and encouragement throughout my study.

Delgermaa KHISHGEE

COMPARING RED AND BLUE ALGORITHMS IN NS2

ABSTRACT

In this study, Blue Active Queue Management (AQM) is implemented and compared with Random Early Detection (RED) algorithm in NS2, which is an open source network simulator widely used for network researching area.

Nowadays, the rapid growth of the Internet requires supporting Quality of Service (QoS). Active Queue Management (AQM) is one of the most effective tools for all QoS measurements. Many AQMs are proposed and studied. One of the most known AQM is RED. RED uses queue length as the indicator of congestion. However, queue length gives us a very little information about congestion.

To overcome RED and other AQMs shortcomings basically different AQM Blue is proposed. In contrary to RED, Blue uses packet drop event and link idle events as the indicator of congestion. This work compares performances of Blue and RED queue algorithms in different network situations.

Keywords: Active queue management, congestion control, network simulator, network animator, xgraph, tracegraph.

NS2 ÜZERİNE RED VE BLUE ALGORİTMALARIN KIYASLANMASI

ÖZ

Bu çalışmada, Blue etkin kuyruk yönetimi (EKY) uygulandı ve RED EKY ile karşılaştırıldı. Uygulamalar ağ araştırma alanında geniş çapta kullanılan açık kodlu ağ simülatörü NS2'de yapılmıştır.

Bugünlerde, İnternetin hızlı büyümesi onu destekleyecek servis kaliteyi gerektiriyor. EKY servis kalitenin her alanında kullanılan en etkin araçlardan biridir. Çok sayıda EKY önerilmiş ve araştırılmıştır. En çok bilinen EKY algoritmalarından biri de RED. RED kuyruk uzunluğu tıkanıklığın işareti olarak kullanıyor. Oysaki, kuyruk uzunluğu bize tıkanıklık hakkında çok az bilgi vermektedir.

RED ve buna benzer EKY'lerin eksikliklerini gidermek üzere tamamen farklı EKY olan Blue önerildi. Blue RED-in aksine tıkanıklığın işareti olarak paket kaybı ve linkin boş durumunu kullanmaktadır. Bu çalışmada Blue ve RED kuyruk yönetimlerinin performanslarını farklı ağ koşullarda kıyaslanmaktadır.

Anahtar sözcükler: Etkin kuyruk yönetimi, tıkanıklık denetimi, ağ simülatör, ağ animatör, xgraph, tracegraph.

CONTENTS

LIST OF FIGURES

Page

LIST OF TABLES

Page

CHAPTER ONE
INTRODUCTION

1.1 Background

Use of internet is exponentially increasing day by day and it results in huge traffic in the Internet. Congestion is the problem that happens in the network when there is so much data traffic that the network cannot handle anymore.

Active Queue Management (AQM) is one of the several techniques that are used to control congestion (Nagle, 1984). Until now many AQM algorithms have been introduced. Blue is one of them and it tries to be superior even in heavily congested situations by using a different approach from the other AQMs (Feng, Kandlur, Saha & Shin, 1999). To see Blue's performance we compared Blue with RED, which is one of the oldest and the most known AQM algorithms (Floyd & Jacobson, 1993).

The system models are simulated using NS2 (Greis, n.db). In NS2, users can add their projects such as a new protocol, network elements and network scenarios to it (Greis, n.da). Blue AQM is not available in NS2 library by default. In this work, my aim is to add and implement Blue AQM into the NS2 and compare with RED, which is already in NS2 library, in aspects of packet loss rate, throughput and packet delay time or queue size.

1.2 Introduction of Chapters

The second chapter introduces the basic knowledge of AQM, and then the methods of Blue and RED AQMs are given in details. Chapter three is a chapter concentrates on NS2 simulator, which is used for our simulation. Chapter four describes the simulation topologies and used parameters of RED and Blue. In chapter five obtained results are presented. Chapter six compares results of RED and Blue AQM and chapter six. Finally, chapter seven concludes our work and discusses future work.

CHAPTER TWO
ACTIVE QUEUE MANAGEMENT ALGORITHMS

Queue management algorithms are used to prevent congestion and to increase the value of QoS in the network (Li & Wang, n.d). When several nodes transmit data to the bottleneck link congestion may occur. Congestion will cause packet delay and packet loss even congestion collapse and lockout. In this situation to restore network will take a long time (Hu, Ren & Chang, n.d).

Queue managements prevent congestion by dropping packets in different dropping methods. There are two basic sort of queue management. One is passive queue management which simply drops packets when queue overflows, the other is active queue management that starts to control the queue length before queue overflow by dropping packets using different dropping probability methods (Wang, 2012).

There are a large number of queue management algorithms to avoid congestion and to improve QoS in the network. The example of passive queue management DropTail is the most widely used passive queue scheme due to its simple implementation. Besides, RED, which is the most popular active queue management algorithm so far, is usually available in most routers but disabled by default.

There are many studies that prove AQMs perform better than passive queue managements. For this reason I worked with only AQMs. In this work, one of the newly proposed algorithms for congestion Blue is evaluated. To understand whether Blue performs good or not we compared with RED AQM method.

2.1 Explicit Congestion Notification

Standard TCP detects congestion after packet drop occurs (Antila, n.d). To avoid packet loss in TCP/IP network explicit congestion notification (ECN) is proposed (Floyd & Ramakrishan, n.d). ECN is a congestion avoidance mechanism and it intends to deliver congestion signals to end nodes without packet loss. To enable ECN both the routers and the end hosts have to support it. When congestion occurs, routers mark ECN capable packets after checking the packets are ECN capable or not. The router marks the ECN Capable Transport (ECT) with the Congestion Experienced (CE) bit. Senders interpret packets the same way as real packet loss if packet is marked with ECN (Floyd & Ramakrishan, n.d). Combination of ECN and active queue managements like RED or Blue are expected to achieve better performance like low packet loss. In this work we used Blue and RED AQM algorithms with ECN capable TCP.

2.2 Random Early Detection

Random Early Detection is an active queue management algorithm proposed by Floyd and Jacobson in 1993. The idea of RED is to control congestion before queue overflows by dropping or marking coming packets and to notify congestion to the sender, thus sender reduces its transmission rates (Floyd & Jacobson, 1993).

The indicator of congestion in RED is average queue length. Packets are dropped or marked with ECN when average queue length exceeds a minimum threshold. The dropping probability changes from minimum threshold to maximum threshold and when maximum threshold is triggered all coming packets are dropped or marked. The Figure 2.1 presents the dropping probability of RED.

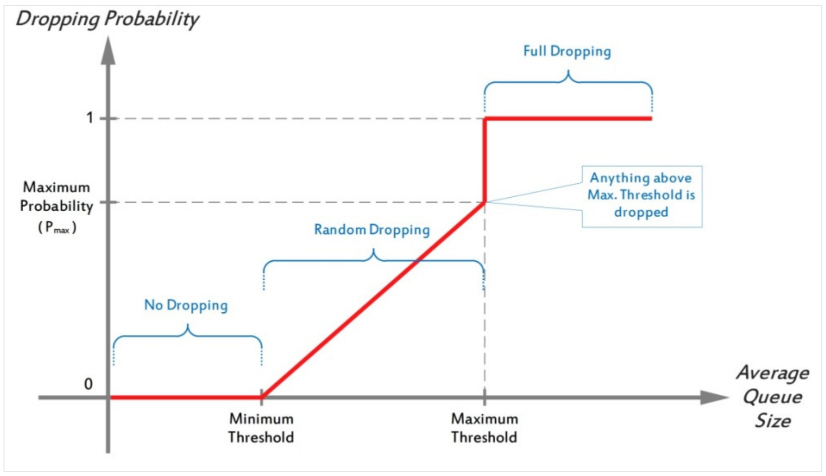


Figure 2.1 Dropping probability of RED.

Since RED detects congestion earlier, it overcomes the “Lockout” and “Full queue” problems in DropTail mechanism which is one of the widely used passive queue management algorithm (Dana & Malekloo, 2010). RED reduces packet delay by keeping queue size small. Parameters in RED are really important and must be selected correctly to get a good performance.

2.2.1 RED Algorithm

First of all, RED calculates the average queue size. The equation is:

$$avg(t) = (1 - Wq)\, avg(t - 1) + Wq*Q(t)$$

Where avg is the average queue size, Q is the current queue size, and Wq is the weight of the queue. The dropping probability of RED varies depending on the maximum probability, minimum threshold and maximum threshold. Once the average queue reaches the minimum threshold, the dropping probability is:

$$Pb = Pmax\, (avg - Tmin) / (Tmax - Tmin)$$

Where Pb is temporarily probability, Pmax is the maximum dropping probability, avg is the average queue size, Tmax and Tmin are the maximum and minimum thresholds respectively. While an arrived packet is marked as the dropping packet, the probability of packet dropping changes immediately. The dropping probability of the coming packet is:

$$Pa = Pb / (1 – C* Pb)$$

Where C is the total number of marked packets. The algorithm is:

```
For each packet arrival
    If the queue is not empty
        $avg(t) = (1 - Wq) * avg(t - 1) + Wq*Q(t)$
    Else using a table lookup
        $avg(t) = (1 - Wq) * (time-q\_time)/s * avg(t - 1)$
```

Where avg is the average queue size, Wq is time constant, s is typical transmission time, time is the current time and q_time is the beginning of the queue idle event.

```
For each packet arrival calculate the average queue size avg
    If $Tmin \le avg < Tmax$
        Calculate the probability Pa and mark or
        drop the arriving packet with probability Pa
    Else If $avg \ge Tmax$
        Mark or drop the arriving packet
    Else
        Accept the arriving packet (Dana & Malekloo, 2010).
```

2.3 Blue

Blue algorithm relies on packet loss and link utilization to perform congestion control. Thus Blue is basically different from almost all of the known AQM algorithms. The original Blue paper indicates that Blue can perform better than the other AQM, which uses queue length to determine congestion in the network (Feng, Kandlur, Saha & Shin, 1999).

Blue marks or drops packets with probability Pm. If packets are continually being dropped or marked due to buffer overflow, Pm is incremented. Conversely Pm is decreased if the queue is empty or link is idle. That means, the dropping probability of Blue does not increase until the buffer gets full, and does not decrease unless link becomes idle or queue is empty (Burri, 2004; Feng, Kandlur, Saha & Shin, 1999).

When large size of buffer is used, to prevent high packet loss and long delay time, we can set queue limit. Thus when queue length achieves the queue limit Blue will act like queue overflows.

Without ECN Blue cannot achieve good performance. Thus we have to set ECN enable in sender and receiver nodes.

2.3.1 Blue Algorithm

Blue uses some other parameters to decide that when and how Pm changes. The parameters are: freeze_time, d1 and d2. Freeze_time determines the minimum time interval of two successive updates of Pm. Pm increases by d1 when queue overflows and decreases d2 when queue is empty or link is idle (Bartok, 2001). The basic Blue algorithm is shown here:

```
Upon packet loss (or Q > L) event:
        If ((now – last_update) > freeze_time) then
                Pm = Pm + d1;
                Last_update = now;

Upon queue empty or link idle event:
        If ((now – last_update) > freeze_time) then
                Pm = Pm – d2;
                Last_update = now;
```

From the algorithm, we can see that Blue is simple in contrast to RED.

2.4 Integrating Blue into the NS2

Blue queue algorithm is firstly implemented in ns2.1b8 release of NS2. By default, the Blue algorithm is not integrated into NS2. But it can be integrated into the distributions of NS2 by following steps:

Step 1: Download the *zipped* file from *http://home.lanl.gov/sunil/ns/ns-blue.tar.gz.*
Step 2: Extract the downloaded *ns-blue.tar.gz* file.

Step 3: Copy the *blue.h* and *blue.cc* files from extracted ns-blue folder into the c++ directory.

Step 4: Add the following lines in *ip.h* file (inside the struct hdr_ip declaration):

```
int pmark_;   //Marker for unresponsive flows - used
```

Step 5: Add the following lines in ip.h file where the member access functions are defined:

```
int& pmark() { return (pmark_);
```

Step 6: Add the following lines to *ns-default.tcl* file:

```
Queue/Blue set drop_front_ false
Queue/Blue set bytes 0
Queue/Blue set setbit true ; #ECN support turned on
by default
Queue/Blue set decrement 0.00025
Queue/Blue set increment 0.0025
Queue/Blue set dhold-time 100ms
Queue/Blue set ihold-time 100ms
Queue/Blue set dalgorithm 0
Queue/Blue set ialgorithm 0
Queue/Blue set pmark 1
Queue/Blue set pktsize 1000
```

Step 7: Add the following lines into ns-lib.tcl in the *simple-link{}* procedure (look for Simulator instproc simple-link{...}):

```
#For Blue
if {$qtype == "Blue"} {
  $q link [$link_($sid:$did) set link_]
}
```

Step 8: Add the following line to the *Makefile* in the *"OBJ_CC = "* section:

```
blue.o  fairblue.o
```

Step 9: For compiling, type *"make"* at the command prompt. If dependency errors show up (unlikely), try the following:

```
make clean
make depend
make
```

2.5 TCP Congestion Control

Since we concentrate on the active queue managements with TCP protocol, we should understand TCP congestion control too. Besides the control of queue, TCP protocol starts to control congestion from transmitting rate. The processes are: slow start, congestion avoidance, fast retransmit and fast recovery (Stevens, 1997).

Slow start: The TCP protocol starts with a very low rate to ensure the success of transmission. Then the rate grows exponentially to reach the slow start ssthresh. The process ensures that the bandwidth is fully utilized.

Congestion avoidance: After reach the sstresh, the rate increases linearly, i.e. one full-sized segment each time, until one packet lost. The process avoids burst traffic and large amount of packet loss.

Fast retransmit: Receiver sends duplicate ACKs with the same segment number if one packet has lost and the sender keeps transmitting. If the sender receives three duplicate ACKs, it will retransmit the packet without waiting for RTT. Fast retransmit is applied for in both TCP Tahoe and TCP Reno.

Fast recovery: It is first used in TCP Reno. Unlike the ssthresh of TCP Tahoe, which restarts from one every time, the ssthresh of TCP Reno restarts from: max (Flight_size/2.2*MSS). And the congestion window equals to ssthresh+3. It is obvious that TCP Reno gains a better performance than TCP Tahoe.

CHAPTER THREE
NETWORK SIMULATOR 2

NS2 is discrete event simulator developed at the University of California at Berkley (Greis, n.db). NS2 is widely used for researching and studying areas. It includes wide range of network functions such as applications, transport layer protocols, network types, and traffic models. When using NS2 users can add their protocols, scenarios and algorithms since it provides flexibility to the users by its developable structure (Kasymaliev, 2004, Başdemir, 2012).

NS2 uses two scripting languages: OTcl and C++. User uses OTcl language to define network topologies, protocols and applications to simulate. On the other hand, detailed simulations of protocols are written in C++ because C++ handles a big volume of data in a short execute time. The languages are linked together using TclCL (Fall & Varadhan, 2011).

The main part of simulation is to analyze the output files. If user defines in the TCL script to trace simulation, NS2 produces out files which contain detailed information about simulation. By using these information we can interpret the results graphically in XGraph and even watch the flows in Network AniMator (NAM).

3.1 NS2 Languages

NS2 is written with combination of OTcl and C++. This is because these two languages have their advantages and disadvantages and NS2 benefits from them (Altman & Jimenez, 2003).

C++ allows us to procedure simulation efficiency and to reduce event processing time. However, this language is difficult and very slow when we have to change some parameters and rerun the simulation. On the other hand, OTcl is much slower while running but very convenient for changing parameters and rerunning (Chung & Claypool,n.d).

3.1.1 Linkage of C++ and OTcl languages

OTcl and C++ linkage is implemented using TclCL language. OTcl gets compiled C++ objects over OTcl linkage. This linkage provides OTcl objects for each of the C++ objects. The control of the C++ objects is given to OTcl. User can add some functions and variables to a C++ linked OTcl object. C++ objects don't have to be linked if they are not controlled in a simulation or used by another object (Chung & Claypool, n.d).

3.2 Agents and Applications

There are two classes of objects in NS2. To generate a traffic agent class and application class are used. To send or receive data, a node needs an agent and an application. Application runs on the top of the agent, and agent is attached to the node. The type of traffic is determined by application. In NS2, generally UDP and TCP agents are used.

3.2.1 TCP Agent

TCP is a dynamic reliable congestion control protocol. In TCP destination produces acknowledgment and send it back to sender to notify packet is well delivered. If packet loss occurs TCP detects as congestion occurs. Thus TCP uses duplex links to send acknowledgment to the sender. TCP sinks are active that they generate acknowledgement and send back to the senders to guarantee packet delivery. TCP has some parameters and user can change the default parameters by defining in TCL script. There are several types of TCP agent. Some of them are: Tahoe, Reno, Newreno, Vegas.TCP uses FTP, HTTP and Telnet applications (Issariyakul & Hossain, 2009). In my topology FTP is used.

3.2.2 UDP Agent

UDP is a transport layer protocol that does not need any connection setup to transfer data. UDP destination node does not generate acknowledgment since UDP does not guarantee data delivery. An application dispatches data to the UDP in variable size chunk, and UDP segments them if necessary. An application used with UDP can communicate with the network layer directly.

UDP uses four different applications: CBR, Exponential, Pareto, TrafficTrace. CBR is used for my topology. When CBR application is used packet size is constant and traffic will be generated in deterministic rate. User can define them in TCL script.

3.2.3 Traffic Sinks

UDP and TCP sources are connected to the destination node by link. UDP uses Null agent as a sink and we can define it in the class Agent/Null. TCP uses its own sink and it could be defined in the class Agent/TCPSink.

3.3 Visualizing Output Results

The most important task in simulation is analysis of output files. After the simulation NS2 produces output files with detailed information if the user writes certain commands to the TCL script to do so. From the output files visualizing tools NAM and XGraph display data graphically.

3.3.1 Network Animator

Network Animator (NAM) is a Tcl/TK based animation tool. If user gives a command into the TCL script to activate NAM trace, it will record all the events into the text file and by using this text file NAM will play back simulation in visual way. From the animation we can see packets traveling, dropping and being received.

NAM has many features. Some of them are: positioning nodes, writing a label to the nodes, giving a shape to the nodes, coloring nodes and links, monitoring specific queue (Delaney & Meeneghan, 2004). User can replay, stop or set forward, and slow down the simulation by using control buttons (Figure 3.1).

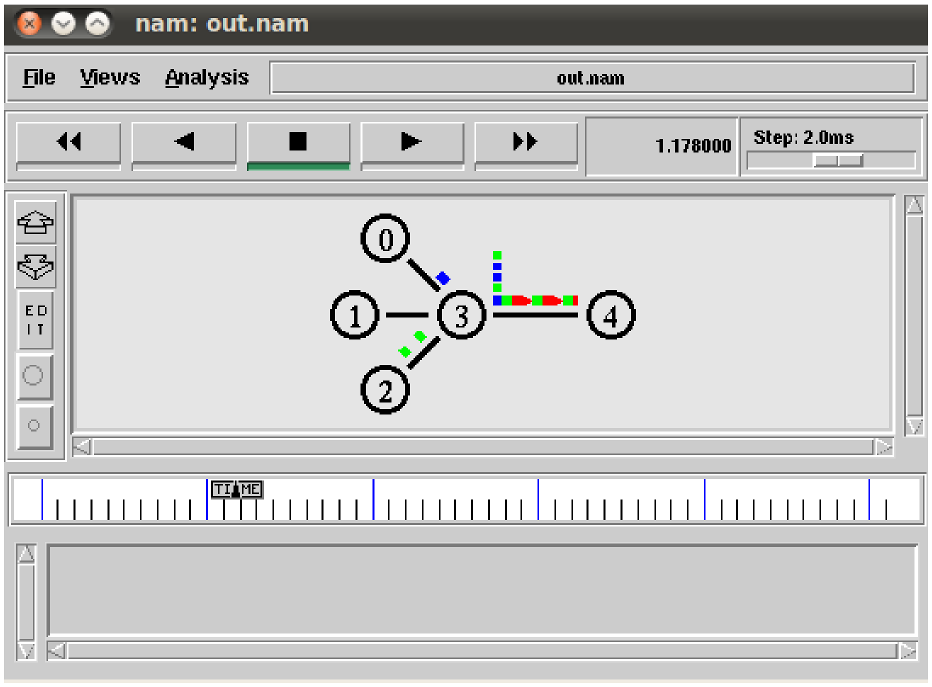


Figure 3.1 Network animator (NAM).

3.3.2 XGraph

The XGraph is an X -z Windows application draws a graph from the text file only has two fields. After simulation is done NS2 produces out files. To use XGraph we have to work on this out file since out file has twelve fields normally. For example it could be value vs. time schedule. We can give a title and labels to our graph and define the colors of graphs and background (Figure 3.2).

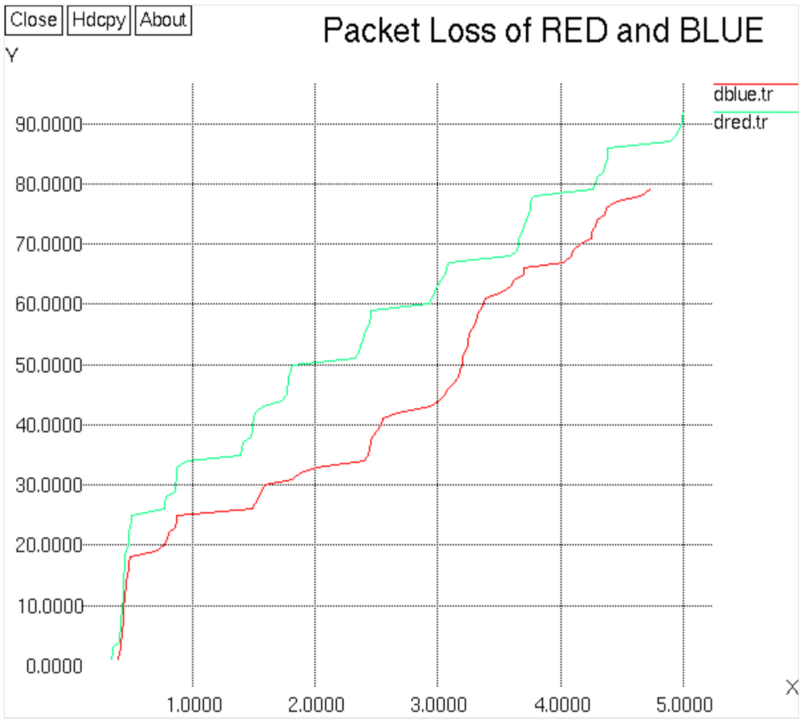


Figure 3.2 XGraph.

3.4 Tracing

3.4.1 Tracing Objects

NS2 produces trace file which contains information about simulation and events registered in the network. When we give a command to record all the events, NS2 inserts four events: EnqT, DeqT, RecvT and DrpT, as indicated in Figure 3.3.

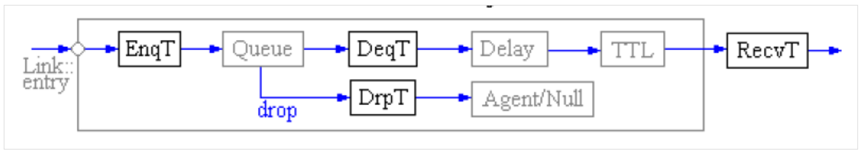


Figure 3.3 Trace objects.

EnqT records event that a packet arrives and queues at the input queue. If queue gets full and packet is dropped due to overflow then this event is recorded as DrpT.

DeqT records event that a packet leaves the queue. RecvT gives the information about that the packet has been received at the output of the link.

3.4.2 Design of the Trace File

From the trace file user can find all information needed such as packet arrivals, departures, drops and link failures. NS2 records all events into the files defined by the user. If user traces all the events, the data in this trace file should be as following:

```
r  0.371024 10   11 tcp 40 C------  5 8.0 9.0 0 1
+  0.371024 11   9  tcp 40 C------  5 8.0 9.0 0 1
-  0.371024 11   9  tcp 40 C------  5 8.0 9.0 0 1
r  0.372056 11   9  tcp 40 C------  5 8.0 9.0 0 1
r  0.470184 11   10 ack 40 C------  1 1.0 0.0 0 2
+  0.470184 10   0  ack 40 C------  1 1.0 0.0 0 2
```

The trace is organized in 12 fields as shown in Table 3.1 (Altan & Jimenez, 2003).

Table 3.1 Explanations of trace columns.

1	Event type.	"**r**" – received
		"**+**" – enqued
		"**-**" – dequed
		"**d**" – dropped
2	Event occurred time.	
3	Departure node.	
4	Arrival node.	
5	Packet type.	
6	Packet size.	
7	Some flags.	
8	Flow id.	
9	Packet source address.	
10	Packet destination address.	
11	Sequence number.	
12	Unique id of the packet.	

CHAPTER FOUR
DEFINING TEST TOPOLOGIES

This chapter describes topologies and parameters which used for simulation. We used ns-2.34 on Ubuntu 10.04 platform to simulate our system models. After simulation we can easily get the information of the simulation in details from out files. This information will help us to analyze the performances of RED and Blue. We analyzed RED and Blue queue managements by the following performance characteristics:

a) Queue size or End-to-end delay
b) Packet loss rate
c) Throughput

Since queue size is in direct proportion to end-to-end delay we do not have to use both of them in a simulation. Only in the first test network simulation, delay time is used.

First of all, we created a large topology and kept the bottleneck link very limited to see the queue performances in a heavily loaded situation. In this scenario only TCP sources are used and TCP packets are being sent to the end of the simulation time. There are two cases in this simulation. In case 1, the buffer size is 100 packets. In case 2, buffer size is 50 packets but all other parameters are same as in case 1.

Next, we used a smaller topology and traffic is not congested as the previous topology. We created four different topologies by changing source and destination node specifications but with same parameters. In these topologies we used two TCP agents and an UDP agent. To learn how long it takes to send the TCP packets when RED or Blue queue management used, we set the TCP file limited as 5000 and 8000 packets.

4.1 Test Network-1: Heavily Loaded Network

4.1.1 Topology

We chose a dump-bell topology for our simulation. The network topology consists of senders, receivers and two routers. The total number of senders and receivers are set to 40. Each sender transmits messages to the opposite receiver through the two routers. The topology of the model is shown in the Figure 4.1. Simulation screenshot in NAM is illustrated in Figure 4.2.

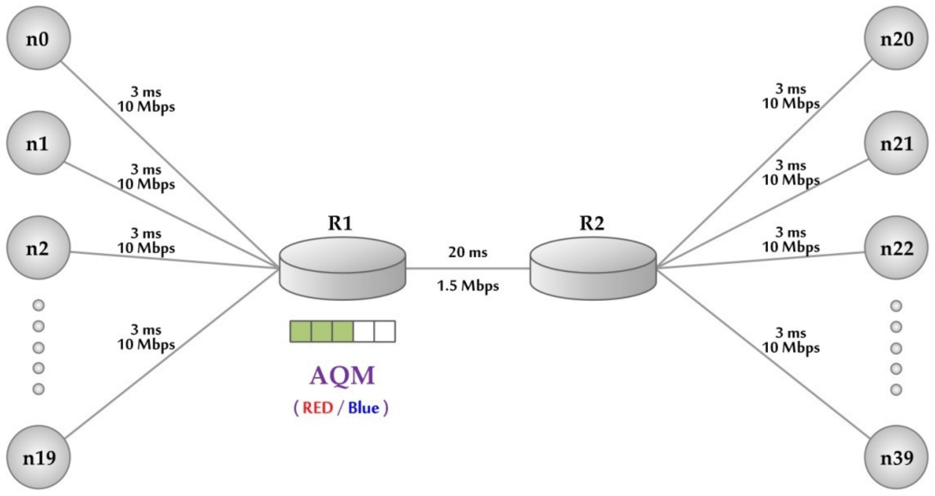


Figure 4.1 Test network-1 topology.

It is clear that the link between two routers is the bottleneck link of the network. The packets sent from the sources queue in the buffer of first router and wait for being transmitted. If the queue becomes full and the senders keep transmitting packets, congestion will happen. To manage the queue and to control congestion RED and Blue algorithms are working on the router one.

4.1.2 Parameters

The parameters of the test network-1are as the following:

- Maximum buffer size: set as 50 or 100;
- Maximum bound on TCP agent window size: 16;

- TCP packet size: 1500 bytes;
- RED minimum threshold: 25 packets for the buffer size of 50 packets; 50 packets for the buffer size of 100 packets;
- RED maximum threshold: equals to the maximum buffer size
- RED linterm_ :1;
- RED q_weight_ : 0.0002;
- Blue increment factor: 0.025;
- Blue decrement factor: 0.0025;
- Blue dhold-time: 100 ms;
- Blue ihold-time: 100 ms;
- Blue Pmark: 1;
- Simulation time: 100seconds

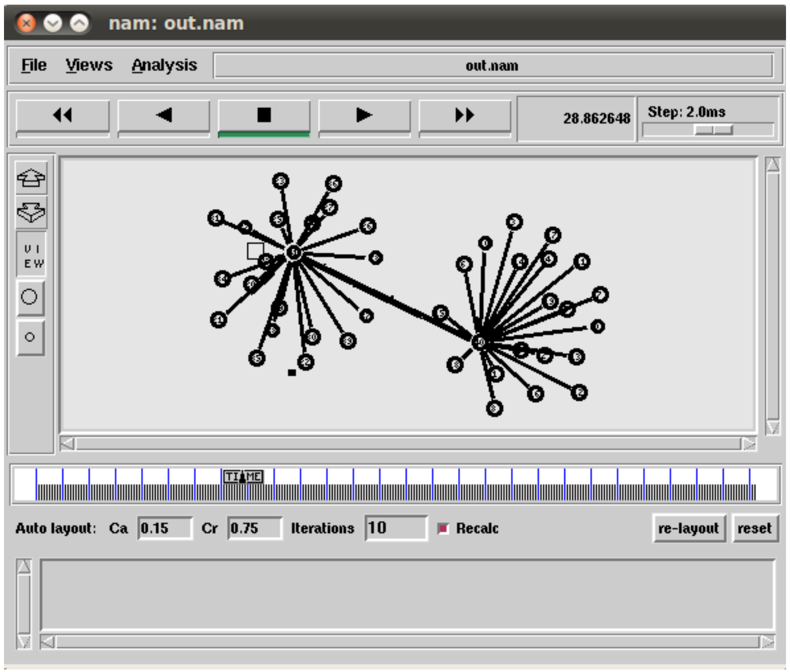


Figure 4.2 Test network-1 NAM screenshot.

Simulations are done with two different buffer sizes. According to the given parameters, maximum delay time for both situations can be calculated. Links from senders to receivers have total delay of 26 ms. By using parameters we can easily calculate maximum delay time of packets. Following is the maximum delay time for simulation with 100 packet buffer size:

L = 1500 bytes = 12000 bits

C = 1.5 Mbps

$X = L / C$ = 12000 bits / (1.5 * 106) bit/second = 8 ms

$T_{max} = X * B + D$ = 8 ms * 100 + 0.026 second = 0.826 second

where,

X – Maximum packet service time

L – Packet length

C – Sending rate of the service link

T_{max} – Maximum delay time

D – Total delay time

B – Buffer size

For simulation with 50 packets buffer size, the maximum delay time is:

T_{max} = 8 ms * 50 + 0.026 second = 0.426 second

4.2 Test Network-2: Wired Network with Three Sources and One Destination

4.2.1 Topology

The network has three sources and one destination. First two nodes are TCP agents and produce FTP packets and third node is UDP agent and produces CBR packets (Figure 4.3). All packets are transmitted to one main destination through the router. Simulation screenshot in NAM is illustrated in Figure 4.4.

The links between nodes and router have bandwidth of 1Mbps and the delay time of 1ms. The bottleneck link bandwidth is 2.5Mbps and 100ms is the delay time.

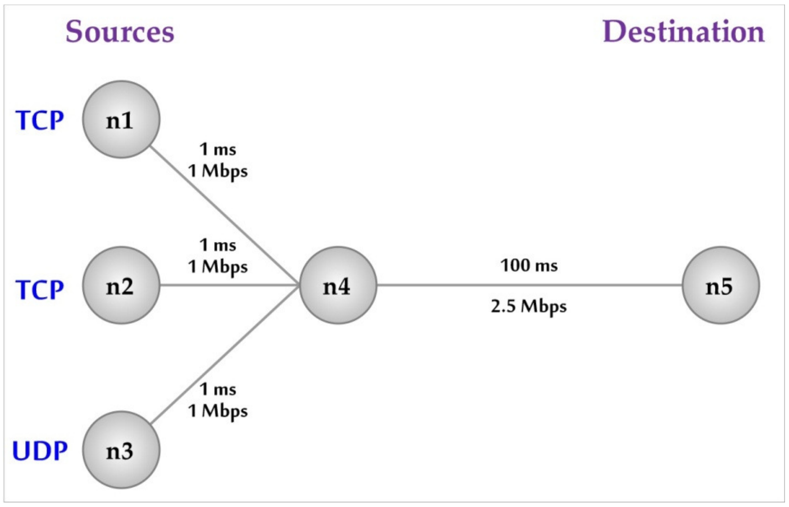


Figure 4.3 Test network-2 topology.

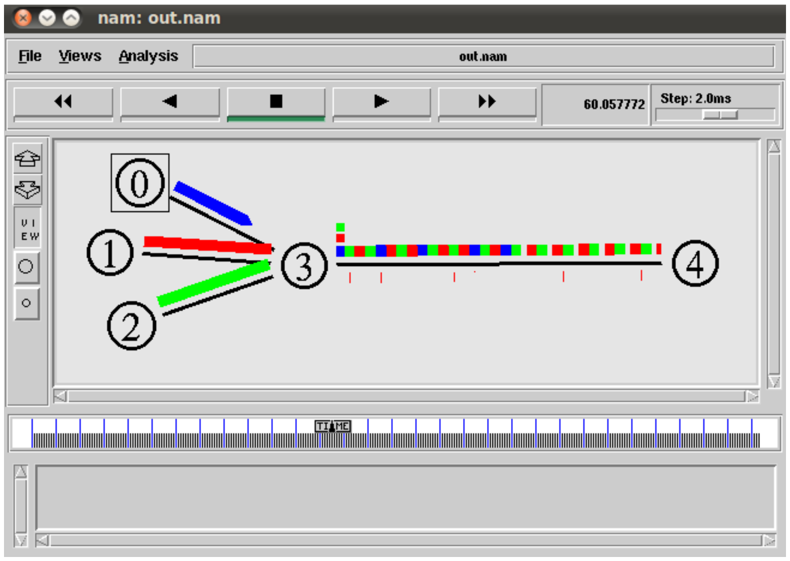


Figure 4.4 Test network-2 NAM screenshot.

4.2.2 Parameters

The parameters of the test network-3 are as the following:

- Maximum buffer size: 15 packets;
- Maximum bound on TCP agent window size: 32;
- TCP packet size: 1000 bytes;
- CBR packet size: 1000bytes;

- CBR rate : 1Mbps;
- RED minimum threshold: 5packets;
- RED maximum threshold: 10packets;
- RED q_weight_ : 0.0002;
- Blue increment factor: 0.025;
- Blue decrement factor: 0.0025;
- Blue dhold-time: 100 ms;
- Blue ihold-time: 100 ms;
- Blue Pmark: 1;
- First TCP produce: 5000packets;
- Second TCP produce: 8000packets;
- CBR start time: 10.0 second;
- Simulation stop time: 150.0 second;

By given parameters, TCP packet sending rate is as following:

$RTT = 1ms*4 + 100ms*2 \approx 200ms = 0.2s$

$(W * P * 8) / RTT = (32 * 1000 * 8) / 0.2 = 1.28Mbps$

where,

W - Window size of TCP

P - Packet size

RTT - Round trip time

Every node sends at least 1Mb data and uses their 1Mb bandwidths fully.

4.3 Test Network-3: Wired Network with Three Sources and Three Destinations

4.3.1 Topology

The topology consists of three senders including two TCP and one UDP and three receivers connected with senders one by one (Figure 4.5). Packets are transmitting to their destination by passing through two routers and in the first router congestion happens. Simulation screenshot in NAM is illustrated in Figure 4.6.

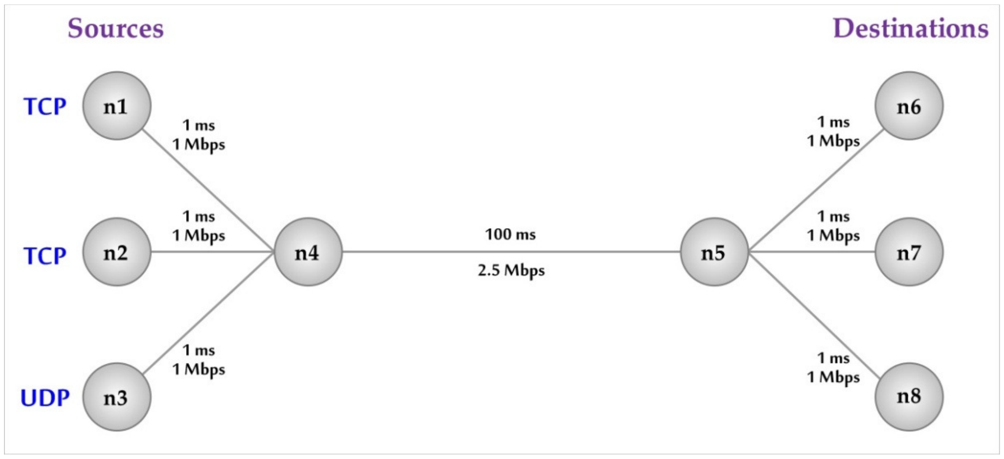
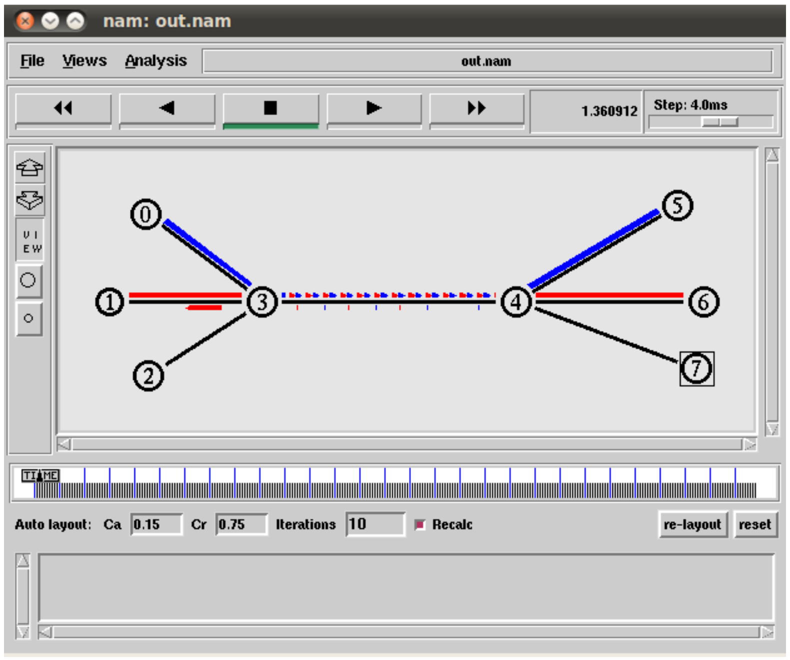


Figure 4.5 Test network-3 topology.

The bottleneck link has bandwidth of 2.5Mbps and other links between routers and nodes have 1Mbps bandwidth. The delay time of bottleneck link is 100ms and 1ms for other links.



Auto layout: Ca 0.15 Cr 0.75 Iterations 10 Recalc

Figure 4.6 Test network-3 NAM screenshot.

4.3.2 Parameters

The parameters of the test network-3 are as the following:

- Maximum buffer size: 15 packets;
- Maximum bound on TCP agent window size: 32;
- TCP packet size: 1000 bytes;
- CBR packet size: 1000bytes;
- CBR interval: 0.008second;
- First TCP produce: 5000packets;
- Second TCP produce: 8000packets;
- CBR start time: 10.0 second
- Simulation stop time: 150.0 second;

The parameters of the RED and Blue are the same to the previous model.

4.4 Test Network-4: Network with Sender Wireless Nodes

4.4.1 Topology

Topology includes three mobile nodes as sender and three wired nodes as receiver. To connect wireless and wired nodes a base station is used (Figure 4.7). Simulation screenshot in NAM is illustrated in Figure 4.8.

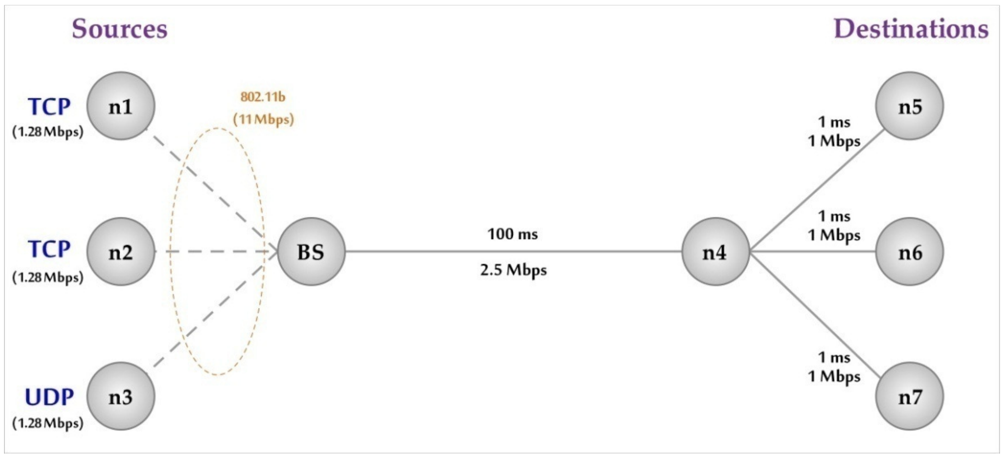


Figure 4.7 Test network-4 topology.

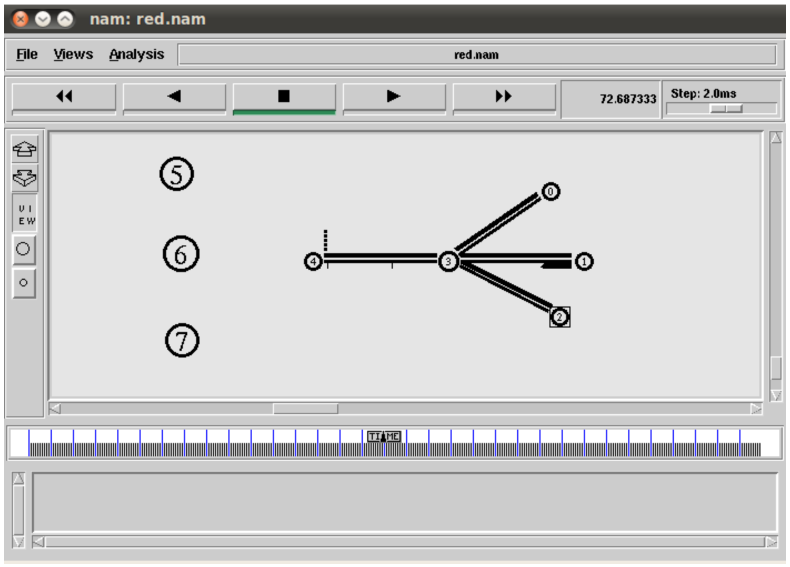


Figure 4.8 Test network-4 NAM screenshot.

4.4.2 Parameters

Since there are both wireless and wired nodes, we have to set parameters for wireless and wired network. For wireless network there must be some additional parameters such as MAC layer type and Physical layer type.

The parameters of the test network-4 are as the following:

- MAC layer type: Mac/802_11;
- Network interface type: Phy/WirelessPhy;

- Routing protocol: DSDV;
- Data rate: 11.0Mbps;
- Maximum buffer size: 15 packets;
- Maximum bound on TCP agent window size: 32;
- TCP packet size: 1000 bytes;
- CBR packet size: 1000bytes;
- CBR interval: 0.008second;
- First TCP produce: 5000packets;
- Second TCP produce: 8000packets;
- CBR start time: 10.0 second
- Simulation stop time: 150.0 second;

The parameters of the RED and Blue are the same to the previous model.

4.5 Test Network-5: Network with Receiver Wireless Nodes

4.5.1 Topology

Wired nodes are senders and wireless nodes are receivers in this topology. Packets are generated from wired nodes and transmitted by base station to reach the wireless destinations (Figure 4.9). Simulation screenshot in NAM is illustrated in Figure 4.10.

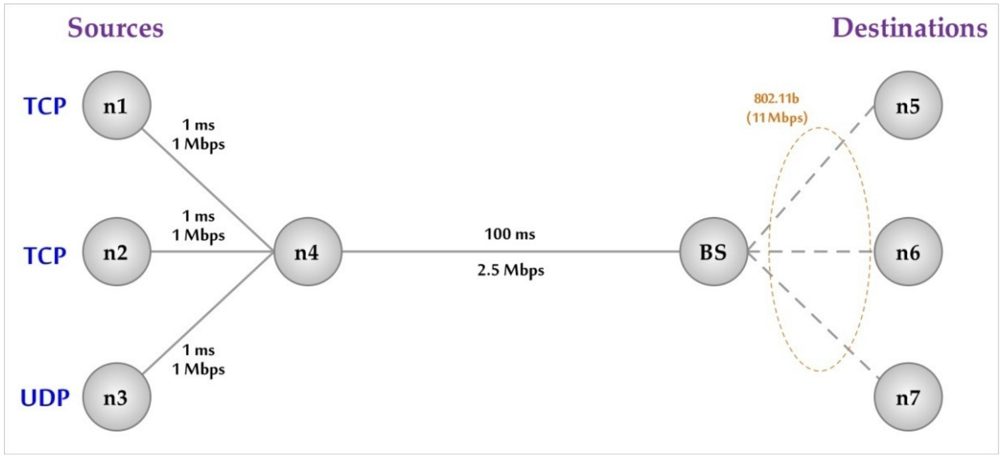


Figure 4.9 Test network-5 topology.

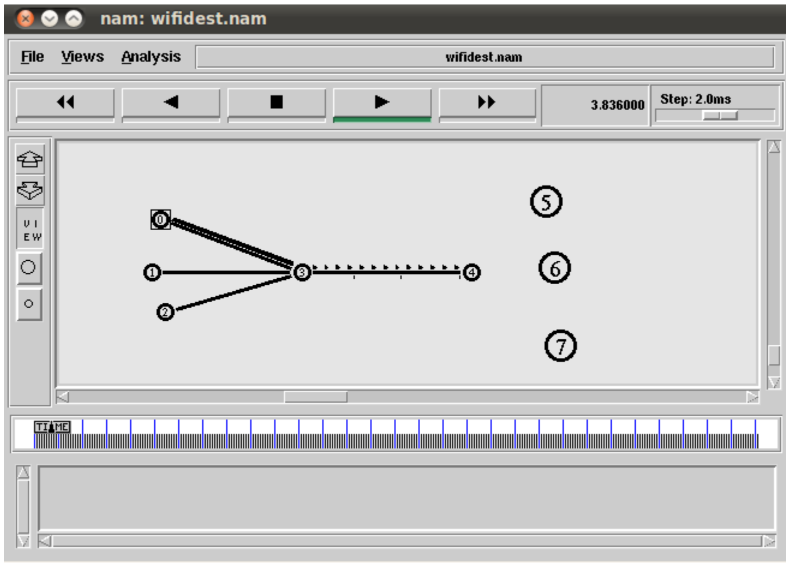


Figure 4.10 Test network-5 NAM screenshot.

The parameters are same as the previous one.

CHAPTER FIVE
SIMULATION RESULTS

We used different networks and get the results. However, the queue performances depends on their parameters thus the comparing them maybe complicated. To get the good comparing results we set the parameters recommended by their original papers.

After simulation we have to work on the output files. There are many ways to analyze output files. We used Tracegraph and Xgraph for our simulations.

5.1 Results of Test Network-1

Our first model has very limited bottleneck link and the parameters are set according to it. The purpose of simulating this network is to see how queue managements act when network is highly congested. We got the following graphs regarding to packet delay time, throughput and packet loss rate of RED and Blue. There are two different results that queue limit is changed 100 to 50 in order to see how queue limit effects simulation results.

5.1.1 End-to-End Delay (a)

Figure 5.1 and Figure 5.2 present the end to end delay time vs. simulation time with buffer size 100 packets and 50 packets respectively. As we calculated before in Chapter four, maximum delays of these topologies are 0.826 seconds and 0.426 seconds. This explains the boundary values of these figures.

RED has more delay time than Blue in both situations. When RED is used, the most of the simulation time queue size changes between minimum threshold and maximum threshold which equals to the buffer limit. We know that delay time is directly affected by queue size.

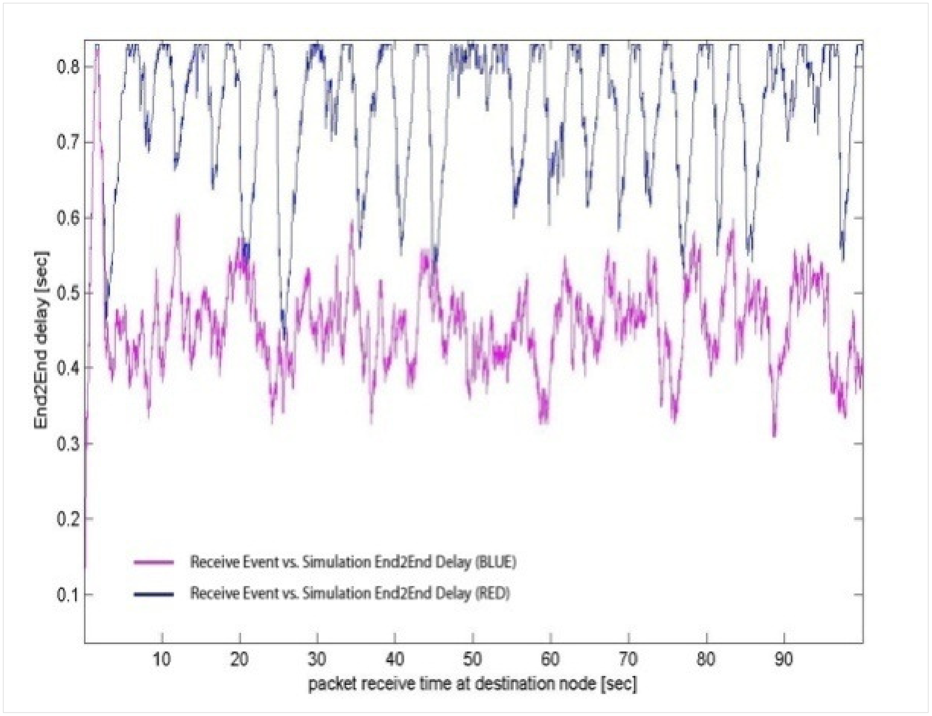


Figure 5.1 Delay (buffer size: 100 packets).

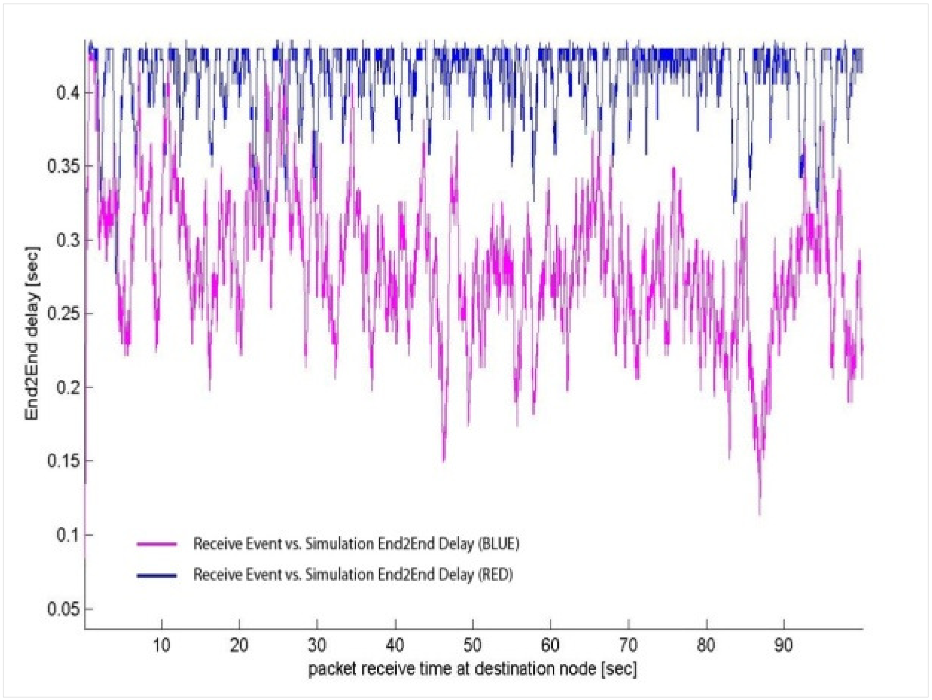


Figure 5.2 Delay (buffer size: 50 packets).

In figure 5.1, Blue buffer accepts packets until queue overflows and it results long delay time in early period of the simulation. After overflow Blue keeps the queue at half of the queue limit and as we can see from the graph, delay time becomes around the half of the maximum delay time.

In figure 5.2, queue limit is changed from 100 to 50 packets thus delay time is decreased. However in RED, most of the time delay tends to approach maximum delay. It is originated from that number of packets are same as previous simulation while queue limit is decreased to 50 thus RED cannot decrease queue size like before. Same as RED, Blue cannot keep queue size as low as previous simulation. Most time delay is more than half of the maximum delay.

5.1.2 Packet Loss Rate (b)

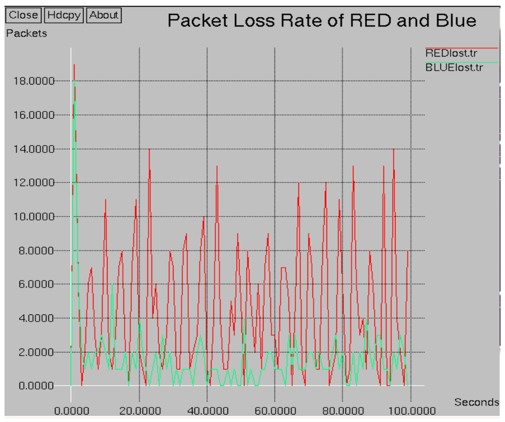


Figure 5.3 Loss rate(buffer size: 100 packets).

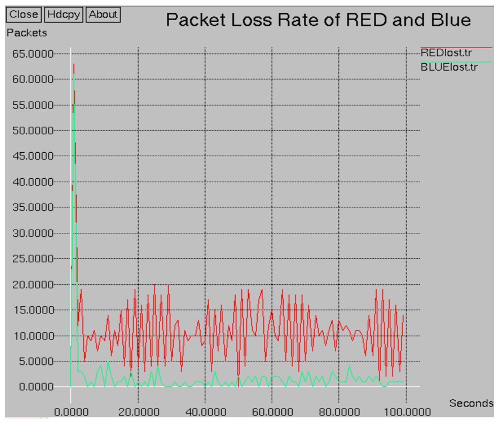


Figure 5.4 Loss rate (buffer size: 50 packets).

Figure 5.3 and Figure 5.4 show the packet drop rate vs. simulation time with 100 packets and 50 packets buffer size respectively. It is obvious that RED has lot more packet loss than Blue in both situations (Table 5.1). We can say here in this situation Blue shows considerably better performance than RED.

Table 5.1 Total lost packet of test network-1.

	Test-1 (Buffer: 100 packets)		**Test-1 (Buffer: 50 packets)**	
AQM	RED	BLUE	RED	BLUE
Total Dropped Packets	467	156	1116	179

5.1.1 Throughput (c)

Figure 5.5 and Figure 5.6 show the throughput of simulations. Bottleneck link has a bandwidth of 1.5 Mbps where 20 links, which has 10 Mbps bandwidth each, converged on it. It seems bottleneck link is very tight link. Thus throughput approached its maximum value in two cases with both AQM.

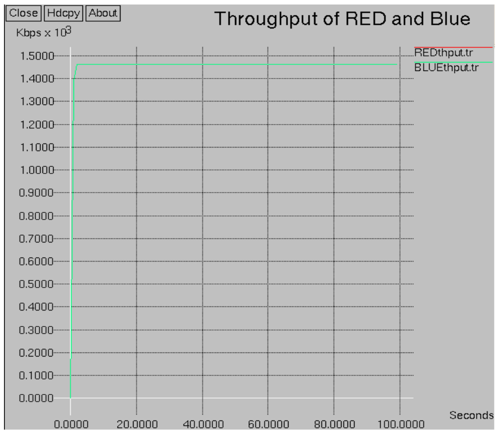


Figure 5.5 Throughput (buffer size: 100 packets).

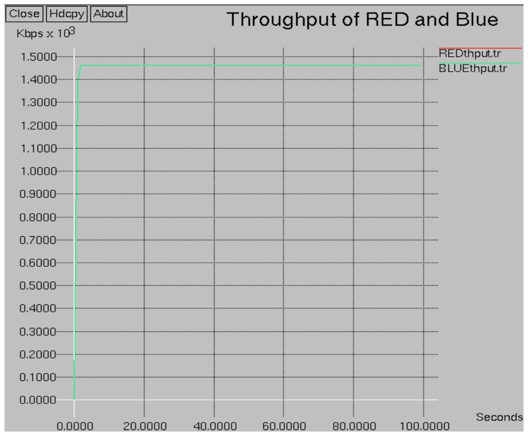


Figure 5.6 Throughput (buffer size: 50 packets).

5.2 Results of Test Network-2,3,4,5

In test network- 2, 3, 4, 5, topologies are similar to each other and all parameters are same thus we can compare them. Two TCP and one UDP connections are used and TCP sources have limited sources as 5000 packets and 8000 packets.

5.2.1 Queue Size (a)

From the Figures 5.7, 5.8, 5.9 and 5.10, we can see that results are similar. The queue size of RED is larger than Blue in all circumstances. This means the packet delay time of RED is longer than the delay time of Blue.

We set the maximum queue size as 15 packets.

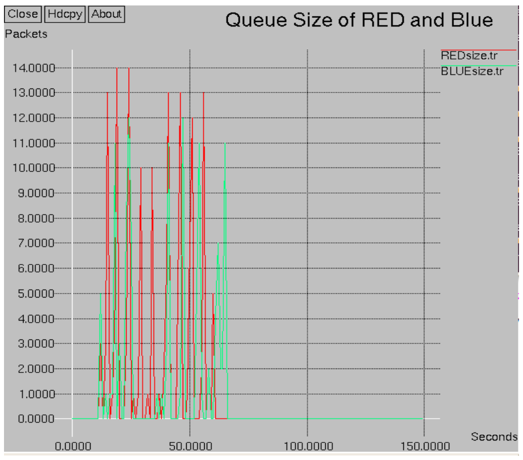


Figure 5.7 Test network-2.

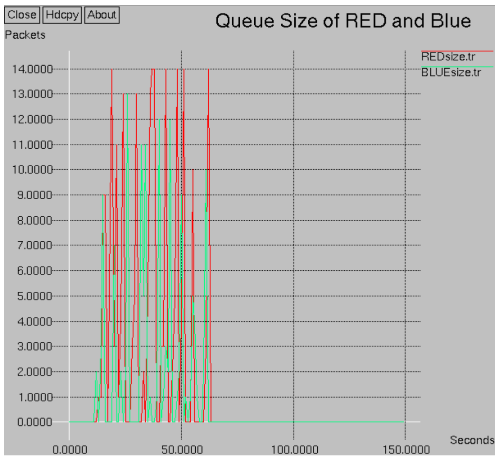


Figure 5.8 Test network-3.

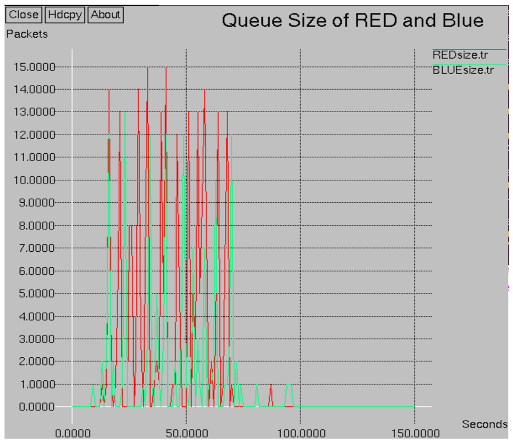


Figure 5.9 Test network-4.

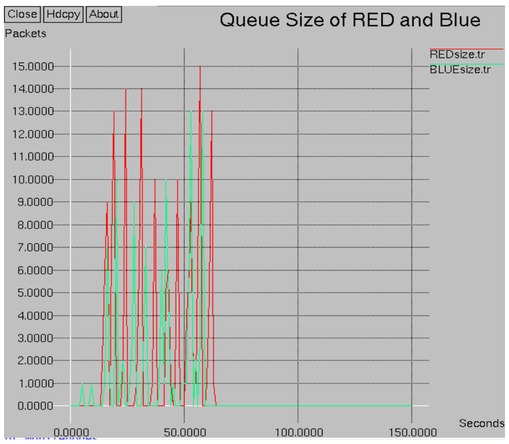


Figure 5.10 Test network-5.

5.2.2 Packet Loss Rate (b)

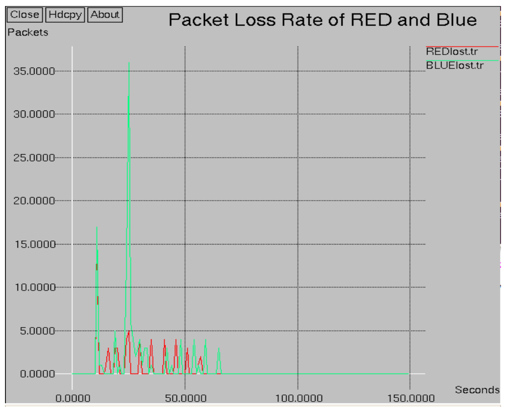


Figure 5.11 Test network-2.

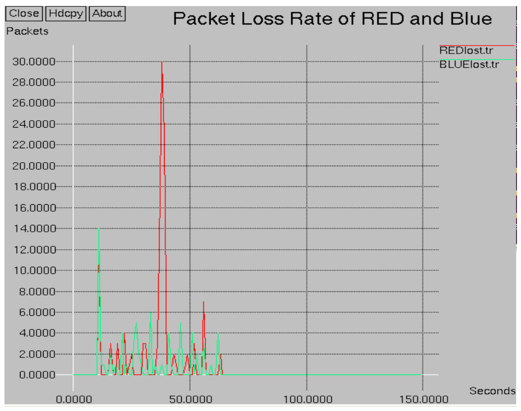


Figure 5.12 Test network-3.

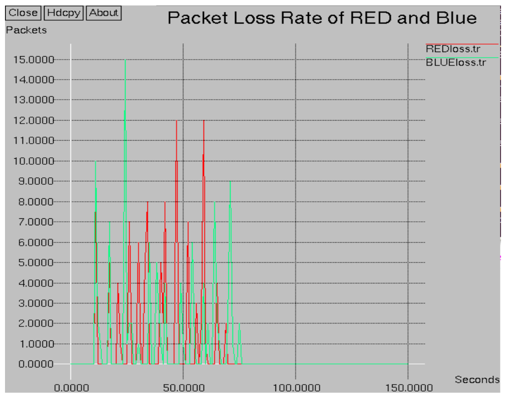


Figure 5.13 Test network-4.

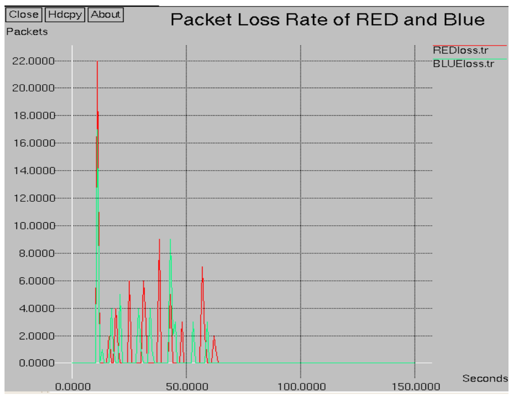


Figure 5.14 Test network-5.

Packet loss rates of test network-2,3,4,5 are illustrated in Figures 5.11, 5.12, 5.13 and 5.14. Table 5.2 shows total packet losses of aforementioned test networks which are calculated with AWK scripts.

Table 5.2 Total lost packet of test network-2,3,4,5.

	Test-2		Test-3		Test-4		Test-5	
AQM	RED	BLUE	RED	BLUE	RED	BLUE	RED	BLUE
Total Dropped Packets	57	121	118	67	102	103	77	58

5.2.3 Throughput (c)

The bottleneck link has 2.5 Mb bandwidth and as we can see from the Figures 5.15, 5.16, 5.17 and 5.18, throughputs are between 1.7 – 2.45 Mb when all three senders are active and TCP sources complete to send the all packets almost at the half of the simulation time.

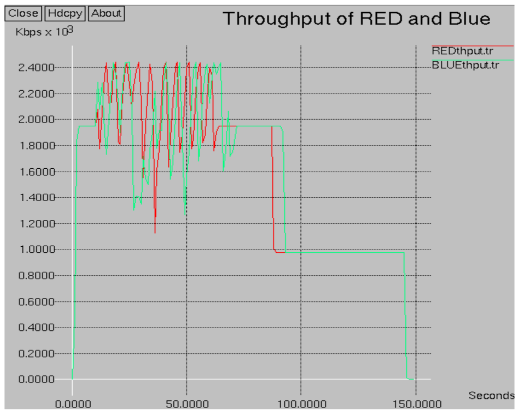


Figure 5.15 Test network-2.

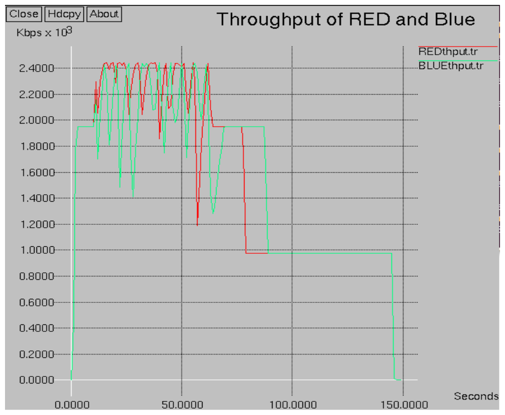


Figure 5.16 Test network-3.

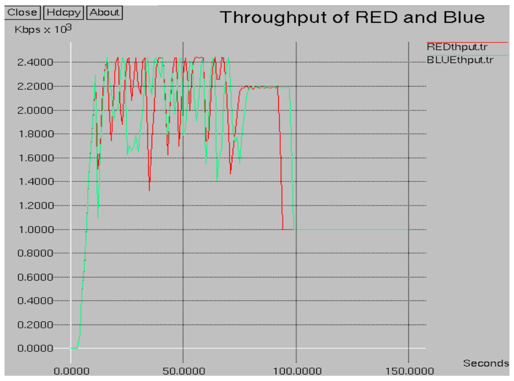


Figure 5.17 Test network-4.

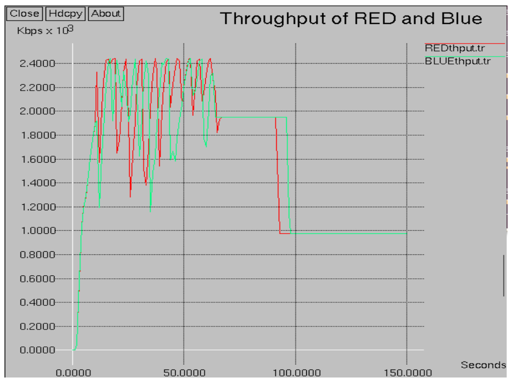


Figure 5.18 Test network-5.

To find out when exactly TCP packets are sent completely can be checked from out trace files. In Tables 5.3 and 5.4, there are columns of out trace files which has information about the last TCP packets. Time when the event occurs is colored.

Table 5.3 The columns related to the last packet of Node1.

	AQM	**1st TCP**
Test 2	*RED*	`r 60.9074 3 4 TCP 1000 ------N 1 0.0 4.0 5000 25922`
	BLUE	`r 68.34892 3 4 TCP 1000 ------N 1 0.0 4.0 5000 27574`
Test 3	*RED*	`r 62.34728 4 5 tcp 1000 ------N 1 0.0 5.0 5000 28670`
	BLUE	`r 62.68808 4 5 tcp 1000 ------N 1 0.0 5.0 5000 26999`
Test 4	*RED*	`r 70.105466 3 0 tcp 1020 ------N 1 1.0.1.0 0.0.0.0 5001 27282`
	BLUE	`r 73.597419 3 0 tcp 1020 ------N 1 1.0.1.0 0.0.0.0 5001 27170`
Test 5	*RED*	`r 63.682427761 _5_ AGT --- 25861 tcp 1000 [13a 1 0 800] ---- [0:0 4194305:0 29 4194305] [5000 0] 1 0`
	BLUE	`r 62.806767555 _5_ AGT --- 24289 tcp 1000 [13a 1 0 800] ---- [0:0 4194305:0 29 4194305] [5000 0] 1 0`

Table 5.4 The columns related to the last packet of Node2.

	AQM	**2nd TCP**
Test 2	*RED*	`r 87.1314 3 4 tcp 1000 ------N 2 1.0 4.1 8000 35696`
	BLUE	`r 92.0514 3 4 TCP 1000 ------N 2 1.0 4.1 8000 36308`
Test 3	*RED*	`r 77.9108 4 6 tcp 1000 ------N 2 1.0 6.0 8000 34522`
	BLUE	`r 87.766 4 6 tcp 1000 ------N 2 1.0 6.0 8000 35754`
Test 4	*RED*	`r 92.732323 3 1 tcp 1020 ------N 2 1.0.2.0 0.1.0.0 8001 36428`
	BLUE	`r 97.748599 3 1 tcp 1020 ------N 2 1.0.2.0 0.1.0.0 8001 37023`
Test 5	*RED*	`r 91.701640022 _6_ AGT --- 36320 tcp 1000 [13a 2 0 800] ---- [2048:0 4194306:0 29 4194306] [8000 0] 1 0`
	BLUE	`r 96.264551146 _6_ AGT --- 36854 tcp 1000 [13a 2 0 800] ---- [2048:0 4194306:0 29 4194306] [8000 0] 1 0`

CHAPTER SIX
COMPARING RESULTS

In this chapter, RED and Blue algorithms are compared based on results shown in chapter five. The most important quality of service measurements such as throughput, packet loss, delay time or queue size are used for our simulation scenarios. The simulation results show that RED and Blue queue managements can perform differently according to the simulation condition and congestion level. Furthermore, the queue parameters have a huge influence to their performance. To set the queue parameters we followed the way shown by the authors of RED and Blue algorithm's original papers. RED algorithm is complicated and function is not easy to understand. Configuring the parameters of RED is also challenging. RED parameters must be reconfigured under different network scenarios. Blue algorithm is really simple from the point of setting parameters besides one can use same parameters in different network scenarios.

We have five different test networks. However, queue size results are almost same for all the simulation models. RED always has bigger queue size than Blue (See Figure 5.7 and Figure 5.8). This is because RED uses average queue length and cannot be aware of coming packet rate so it takes a while the average queue size to reflect the current situation. In contrary to RED, Blue can realize the condition of load and react instantly according to the congestion situation. Queue size directly affects packet delay time thus RED always has more delay time.

In test network-1, delay time is shown, not queue size. For the most part of the simulation time, Blue keeps packet delay time as half of RED delay time in test network-1 when buffer is 100 packets (See Figure 5.1). Blue needs little time to stabilize queue size thus blue has more delay in the first few seconds of the simulation. When buffer is changed to 50 packets RED delay time is approximately between 0.35second to 0.43second which is almost the boundary delay time while Blue keeps delay time around 0.2 and 0.35 seconds (See Figure 5.2).

Test network-1 is a heavily loaded network therefore it is obvious that large number of packet loss is inevitable. From the Figures 5.3 and 5.4 packet loss rate in RED algorithm seems much more than Blue algorithm. Both queue managements have more packet loss at the start of the simulation and after a period of huge packet loss queue managements tries to minimize the loss. Blue can perform better even with small buffer in heavily loaded network. In table 5.1, total packet losses are shown. Blue has dropped 156 packets with 100 packets buffer and 179 packets with 50 packet buffer. We can say here, Blue can perform congestion control even with minimal amount of buffer size. In the same table, RED has 467 total dropped packets but 1116 packets are lost when buffer size is changed from 100 packets to 50 packets. It means RED needs larger buffer size to perform better. If the network is not congested that much the results become completely different. The gap is getting smaller between performances of RED and Blue from the point of packet loss. Even in some situation RED may have less packet loss (See Figure 5.11 and Table 5.2). The network has one CBR source in addition to two TCP sources and CBR packets are not affected by the queue managements. For that reason packets are still dropping even in uncongested network.

In test network-1, throughput is almost equal to link bandwidth due to loaded traffic sources thus both queue managements' link utilizations are high (See Figure 5.5 and Figure 5.6). From the figures we cannot see the difference between RED and Blue queue throughputs. In uncongested network like test network-2 and test network-3, the throughput of Blue is smaller and has more fluctuation while RED throughput tends to be equal to bandwidth in most of the time (See Figures 5.15 and 5.16).

Since FTP files are set as limited we can compare the ability of queue managements to send the FTP packets in a possible short time. In Table 5.1 and Table 5.2 in chapter five, the exact arrival time of last TCP packets are shown. FTP packets are sent earlier in RED queue management than Blue queue management. The differences between finish time of RED and Blue queue managements are not small, even in some case it reaches to 10 seconds.

For wireless network simulations, what we have found can be summarized as follows:

Queue size results in wireless network have a little difference from the wired network results (See Figure 5.9 and Figure 5.10). Queue size is larger than wired network but RED still has larger queue size than Blue algorithm.

In wireless network RED and Blue queue managements behave similar to each other in point of dropping packets. Total numbers of lost packets are almost equal when compared to wired network (See Table 5.2).

Throughputs of wireless networks are smaller compared to wired network. However, in wireless network RED and Blue throughputs become almost equal. In other words the queue managements act similar to each other in wireless network.

The FTP packets are sent later in wireless network (See Figure 5.17 and Figure 5.18). In Table 5.3 and Table 5.4 the finish times are shown. In the same way as wired network RED finishes TCP packets much earlier in wireless network.

CHAPTER SEVEN
CONCLUSION AND FUTURE WORK

In this work we compared RED and Blue AQM in several different network circumstances. The comparison is made in terms of queue size, delay time, packet loss and throughput. We have used NS2 simulator for our simulation.

From the results, we can say here both queue managements, RED and Blue, have advantages and disadvantages depend on the situations such as network topologies, packet sending application or agent, congestion level, etc. Thus one has to choose the right queue management which performs better for his/her expected values.

If the network is heavily congested, the Blue is definitely right choice. Blue can handle the congestion with less packet loss and short delay time. Blue may present almost same performance even with small buffer size in heavily loaded network since Blue keeps the queue size as a minimal amount. We cannot expect same consequences from RED algorithm which needs large buffer size to control the heavily loaded network.

It is obvious that if network has a delay sensitive traffic, Blue queue management could be recommended. Blue always keeps the queue size small. In RED, queue size is always bigger than Blue.

RED algorithm turns out to be good at transferring FTP files in a short time. Therefore in file transfer we would use RED algorithm. Besides, RED has almost equal packet loss when used in wireless network.

To get good results in RED algorithm, the parameters should be carefully configured and buffer size must be big enough. Blue parameters are really simple and working mechanism is also easy to understand.

There are many other queue managements. Most of the active queue managements are originated from RED algorithm. For future work other queue managements especially Orange queue management, proposed by Dr. Malik Kemal ŞİŞ and his student Aziz Kasymaliev, can be studied and compared with RED and Blue (Kasymaliev, 2004).

REFERENCES

Altman, E & Jimenez, T. (December 4, 2003). *NS simulator for beginners.* Retrieved May 25, 2012, from http://www-sop.inria.fr/members/Eitan.Altman/ns.htm.

Antila, J. (n.d). *TCP Performance Simulations Using NS2.* Retrieved July 15, 2012, fromhttp://web.mst.edu/~bckd2/CpE401/project/NS2%20simulations/special_study%20on%20TCP.pdf.

Bartok, I. (2001). *Implementation and Evaluation of the Blue Active Queue Management Algorithm.*Budapest University, Thesis of Master's Degree.

Başdemir, K. (2012). *A new queue management protocol and network simulation with NS2 (MOR).*Dokuz Eylul University, Thesis of Master's Degree.

Burri, S. (May 5, 2004). *Blue: Active Queue Management.* Retrieved May 11, 2012, from http://thefengs.com/wuchang/blue/burri04blue.pdf.

Chung, J & Claypool, M. (n.d). *NS by example.* Retrieved August 5, 2012, from http://nile.wpi.edu/NS/.

Dana, A., & Malekloo, A. (2010). *Performance comparison between active and passive queue management.* International Journal of Computer Science Issues, Vol. 7, Issue 3, No 5.

Delaney, D., & Meeneghan, P. (2004). *An introduction to NS, NAM and Otcl scripting.* Retrieved May 11, 2012, from http://ceit.aut.ac.ir/~bakhshis/ns-2/An%20Introduction%20to%20NS,%20Nam%20and%20OTcl%20scripting.pdf.

Fall, K & Varadhan, K. (October 4, 2011). *The ns Manual.* UC Berkeley, LBL, USC/ ISI,Xerox PARC. Retrieved May 1, 2012, from http://www.isi.edu/nsnam/ns/doc/ ns_doc.pdf.

Feng, W, Kandlur, D.D, Saha D, Shin, K.G. (April 1999). *Blue: A New Class of Active Queue Management Algorithms.* Retrieved July 10, 2012, from http://www.cs.ust.hk/faculty/bli/660h/feng99blue.pdf.

Floyd, S & Jacobson, V. (August 1993) *Random Early Detection gateways for Congestion Avoidance.* (V.1 N.4) (397-413).

Floyd, S & Ramakrishnan, (n.d). K.K. *ECN (Explicit Congestion Notification) in TCP/IP*. Retrieved May 11, 2012, from http://www.icir.org/floyd/ecn.html.

Greis, M. (n.d.a). *Tutorial for the Network Simulator.* Retrieved May 11, 2012, from http://www.isi.edu/nsnam/ns/tutorial/index.html.

Greis, M. (n.d.b). *The Network Simulator – ns – 2.* Retrieved May 11, 2012, from http://www.isi.edu/nsnam/ns/.

Hu, N., Ren, L & Chang J. (n.d). *Evaluation of Queue Management Algorithms.* Retrieved July 15, 2012, from http://www.cs.cmu.edu/afs/cs/user/hnn/www/cs744_project/744-report.htm.u

Issariyakul, T & Hossain, E. (2009). *Introduction to Network simulator NS2.* Springer. Retrieved May 12, 2012, from http://www.4shared.com/office/fj26P7s1/Introduction_to_Network_Simula.htm.

Li, M., & Wang, H. (n.d). *Study of active queue management algorithms.* Retrieved December 21, 2012, from http://web.eecs.utk.edu/~itamar/courses/Archieves/ECE-692%20-%20Spring%202004/Project_Papers/Paper5.pdf.

Nagle, J. (January, 1984). *Congestion Control in IP/TCP Internetworks.* Retrieved May 11, 2012, from http://tools.ietf.org/html/rfc896.

Stevens, W.R. (January, 1997). *TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms.* Retreived May 12, 2012, from http:// xml2rfc.tools.ietf.org/rfc/rfc2001.txt.

Kasymaliev, A. (February, 2004). *Installing NS2 & integrating a new queue algorithm (ORANGE).*Dokuz Eylul University, Thesis of Master's Degree.

Wang, M. (May, 2012). *Comparison between Droptail and AQM-RED in wireless network.* Retrieved December 21, 2012, from http://dspace.cc.tut.fi/dpub/bitstrea m/handle/123456789/21058/wang.pdf.

APPENDIX

TCL SCRIPTS

1. HEAVILY LOADED NETWORK

```
#==============================================================
# Define options
#==============================================================
set ns [new Simulator]
set nf [open out.nam w]
$ns namtrace-all $nf
set tracefd [open bout.tr w]
$ns trace-all $tracefd
#==============================================================
#set RED parameters
#==============================================================
Queue/RED set thresh_ 65
#When buffer size is 50 packets
#Queue/RED set thresh_ 25
Queue/RED set maxthresh_ 100
#When buffer size is 50 packets
#Queue/RED set thresh_ 50
Queue/RED set bytes_ false
Queue/RED set queue_in_bytes_ false
Queue/RED set linterm_ 1
Queue/RED set q_weight_ 0.00002
Queue/RED set drop_tail_ true
Queue/RED set setbit true
#==============================================================
#set BLUE parameters
#==============================================================
Queue/Blue set drop_front_ false
Queue/Blue set bytes 0
Queue/Blue set setbit true
Queue/Blue set decrement 0.0025
Queue/Blue set increment 0.025
Queue/Blue set dhold-time 100ms
Queue/Blue set ihold-time 100ms
Queue/Blue set Pmark 1

#==============================================================
# set nodes
#==============================================================
set number 20
for {set i 0} {$i < $number } {incr i} {
set n($i) [$ns node]
}
for {set i 0} {$i < $number } {incr i} {
set s($i) [$ns node]
}
set r1 [$ns node]
set r2 [$ns node]
for {set i 0} {$i < $number } {incr i} {
```

```
$ns duplex-link $n($i) $r1 10Mb 3ms DropTail
}
for {set i 0} {$i < $number } {incr i} {
$ns duplex-link $s($i) $r2 10Mb 3ms DropTail
}
$ns duplex-link $r1 $r2 1.5Mb 20ms Blue
#To use RED queue
#$ns duplex-link $r1 $r2 1.5Mb 20ms RED
$ns queue-limit $r1 $r2 100
#to set buffer size 50 packets
#$ns queue-limit $r1 $r2 50
$ns queue-limit $r2 $r1 100
#==============================================================
# set TCP
#==============================================================
for {set i 0} { $i < $number } {incr i} {
set tcp($i) [new Agent/TCP/Reno]
$tcp($i) set class_ $i
$tcp($i) set window_ 16
$tcp($i) set packetSize_ 1460
$tcp($i) set bytes_ false
$ns attach-agent $n($i) $tcp($i)
$tcp($i) set ecn_ 1
}
for { set i 0 } { $i < $number} { incr i } {
set sink($i) [new Agent/TCPSink]
$sink($i) set ecn_ 1
$ns attach-agent $s($i) $sink($i)
$ns connect $tcp($i) $sink($i)
set ftp($i) [$tcp($i) attach-source FTP]
}

# set time
#==============================================================
for { set i 0 } { $i < $number } { incr i } {
$ns at 0.0 "$ftp($i) start"
}
#$ns at 0.0 "record"
$ns at 100.0 "finish"
#==============================================================
# define finish
#==============================================================
proc finish {} {
global nf tracefd ns
close $nf
close $tracefd
exec nam out.nam &
exit 0
}
$ns run
```

2. NETWORK WITH THREE SOURCES AND ONE DESTINATION

```
#Create a simulator object
set ns [new Simulator]

#Define different colors for data flows (for NAM)
$ns color 1 Blue
$ns color 2 Red
$ns color 3 Green

#Open the nam trace file
set nf [open out.nam w]
$ns namtrace-all $nf

#Open the output files
set tr [open blue.tr w]
$ns trace-all $tr

#Define the finish procedure
proc finish {} {
global ns nf qsize qbw qlost
$ns flush-trace
#Close the trace file
close $nf
close $qsize
close $qbw
close $qlost
#Execute nam on the trace file
exec nam out.nam &
global tr
#Close the output files
close $tr
exec  xgraph  BLUEsize.tr  -geometry  800x400  -t  "Queue  size"  -x
"Seconds" -y "Packets" &
exec xgraph BLUEthput.tr -geometry 800x400 -t "Throughput" -x "secs"
-y "Kbps" -fg white &
exec xgraph BLUElost.tr -geometry 800x400 -t "Packet loss rate" -x
"Seconds" -y "Packets" &
exit 0
}

#Create nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]

#Define RED queue parameters
Queue/RED set thresh_ 5
Queue/RED set maxthresh_ 10
Queue/RED set bytes_ false
Queue/RED set queue_in_bytes_ false
Queue/RED set linterm_ 1
Queue/RED set q_weight_ 0.0002
Queue/RED set drop_tail_ true
Queue/RED set setbit_ true
```

```
#Define Blue queue parameters
Queue/Blue set drop_front_ false
Queue/Blue set bytes false
Queue/Blue set queue_in_bytes_ false
Queue/Blue set setbit_ true
Queue/Blue set decrement 0.00025
Queue/Blue set increment 0.0025
Queue/Blue set dhold-time 100ms
Queue/Blue set ihold-time 100ms
Queue/Blue set pmark 1

#Create links between nodes
$ns duplex-link $n0 $n3 1Mb 1ms DropTail
$ns duplex-link $n1 $n3 1Mb 1ms DropTail
$ns duplex-link $n2 $n3 1Mb 1ms DropTail
$ns simplex-link $n3 $n4 2.5Mb 100ms Blue
#$ns simplex-link $n3 $n4 2.5Mb 100ms RED
$ns simplex-link $n4 $n3 1Mb 100ms DropTail

#Define positions
$ns duplex-link-op $n0 $n3 orient right-down
$ns duplex-link-op $n1 $n3 orient right
$ns duplex-link-op $n2 $n3 orient right-up
$ns duplex-link-op $n3 $n4 orient right

#Queue monitor for NAM
$ns duplex-link-op $n3 $n4 queuePos 0.5

#Maximum number of packets for queue
$ns queue-limit $n3 $n4 15

#Setup TCP UDP connections
set tcp0 [new Agent/TCP/Reno]
$tcp0 set class_ 1
$tcp0 set window_ 32
$tcp0 set packetSize_ 960
$tcp0 set ecn_ 1
$ns attach-agent $n0 $tcp0

set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0

set tcp1 [new Agent/TCP/Reno]
$tcp1 set class_ 2
$tcp1 set window_ 32
$tcp1 set packetSize_ 960
$tcp1 set ecn_ 1
$ns attach-agent $n1 $tcp1

set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1

set udp2 [new Agent/UDP]
$udp2 set class_ 3
$ns attach-agent $n2 $udp2

set cbr2 [new Application/Traffic/CBR]
$cbr2 set packetSize_ 1000
```

```
$cbr2 set interval_ 0.008
$cbr2 attach-agent $udp2

set sink1 [new Agent/TCPSink]
$sink1 set ecn_ 1
$ns attach-agent $n4 $sink1

set sink2 [new Agent/TCPSink]
$sink2 set ecn_ 1
$ns attach-agent $n4 $sink2

set null3 [new Agent/Null]
$ns attach-agent $n4 $null3

$ns connect $tcp0 $sink1
$ns connect $tcp1 $sink2
$ns connect $udp2 $null3

#Tracing the queue
set qsize [open BLUEsize.tr w]
set qbw [open BLUEthput.tr w]
set qlost [open BLUElost.tr w]

set qfile [$ns monitor-queue $n3 $n4 [open queue.tr w] 0.05]
[$ns link $n3 $n4] queue-sample-timeout;

proc record {} {
global ns qfile qsize qbw qlost
set time 1
set now [$ns now]
$qfile instvar parrivals_ pdepartures_ bdrops_ bdepartures_ pdrops_
pkts_
puts $qsize "$now [expr $pkts_]"
puts $qbw "$now [expr $bdepartures_*8/1024/$time]"
set bdepartures_ 0
puts $qlost "$now [expr $pdrops_/$time]"
set pdrops_ 0
$ns at [expr $now+$time] "record"
}

#Set time
$ns at 0.0 "record"
$ns at 0.0 "$ftp0 produce 5000"
$ns at 0.0 "$ftp1 produce 8000"
$ns at 10 "$cbr2 start"
$ns at 145 "$cbr2 stop"

#Call the finish procedure
$ns at 150 "finish"

#Run the simulation
$ns run
```

3. NETWORK WITH THREE SENDERS AND THREE RECEIVERS

```
#Create a simulator object
set ns [new Simulator]

#Define different colors for data flows (for NAM)
$ns color 1 Blue
$ns color 2 Red
$ns color 3 Green

#Open the nam trace file
set nf [open out.nam w]
$ns namtrace-all $nf

#Open the output files
set tr [open blue.tr w]
$ns trace-all $tr

#Define the finish procedure
proc finish {} {
global ns nf qsize qbw qlost
$ns flush-trace
#Close the trace file
close $nf
close $qsize
close $qbw
close $qlost
#Execute nam on the trace file
exec nam out.nam &
global tr
#Close the output files
close $tr
exec  xgraph  BLUEsize.tr  -geometry  800x400  -t  "Queue  size"  -x
"Seconds" -y "Packets" &
exec xgraph BLUEthput.tr -geometry 800x400 -t "Throughput" -x "secs"
-y "Kbps" -fg white &
exec xgraph BLUElost.tr -geometry 800x400 -t "Packet loss rate" -x
"Seconds" -y "Packets" &
exit 0
}

#Create nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
set n6 [$ns node]
set n7 [$ns node]

#Define RED queue parameters
Queue/RED set thresh_ 5
Queue/RED set maxthresh_ 10
Queue/RED set bytes_ false
Queue/RED set queue_in_bytes_ false
Queue/RED set linterm_ 1
```

```
Queue/RED set q_weight_ 0.0002
Queue/RED set drop_tail_ true
Queue/RED set setbit_ true

#Define Blue queue parameters
Queue/Blue set drop_front_ false
Queue/Blue set bytes false
Queue/Blue set queue_in_bytes_ false
Queue/Blue set setbit_ true
Queue/Blue set decrement 0.00025
Queue/Blue set increment 0.0025
Queue/Blue set dhold-time 100ms
Queue/Blue set ihold-time 100ms
Queue/Blue set pmark 1

#Create links between nodes
$ns duplex-link $n0 $n3 1Mb 1ms DropTail
$ns duplex-link $n1 $n3 1Mb 1ms DropTail
$ns duplex-link $n2 $n3 1Mb 1ms DropTail
$ns simplex-link $n3 $n4 2.5Mb 100ms Blue
#$ns simplex-link $n3 $n4 2.5Mb 100ms RED
$ns simplex-link $n4 $n3 1Mb 100ms DropTail
$ns duplex-link $n4 $n5 1Mb 1ms DropTail
$ns duplex-link $n4 $n6 1Mb 1ms DropTail
$ns duplex-link $n4 $n7 1Mb 1ms DropTail
#Define positions
$ns duplex-link-op $n0 $n3 orient right-down
$ns duplex-link-op $n1 $n3 orient right
$ns duplex-link-op $n2 $n3 orient right-up
$ns duplex-link-op $n3 $n4 orient right

#Queue monitor for NAM
$ns duplex-link-op $n3 $n4 queuePos 0.5

#Maximum number of packets for queue
$ns queue-limit $n3 $n4 15

#Setup TCP UDP connections
set tcp0 [new Agent/TCP/Reno]
$tcp0 set class_ 1
$tcp0 set window_ 32
$tcp0 set packetSize_ 960
$tcp0 set ecn_ 1
$ns attach-agent $n0 $tcp0

set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0

set tcp1 [new Agent/TCP/Reno]
$tcp1 set class_ 2
$tcp1 set window_ 32
$tcp1 set packetSize_ 960
$tcp1 set ecn_ 1
$ns attach-agent $n1 $tcp1

set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1

set udp2 [new Agent/UDP]
```

```
$udp2 set class_ 3
$ns attach-agent $n2 $udp2

set cbr2 [new Application/Traffic/CBR]
$cbr2 set packetSize_ 1000
$cbr2 set rate_ 1Mb
$cbr2 attach-agent $udp2

set sink1 [new Agent/TCPSink]
$sink1 set ecn_ 1
$ns attach-agent $n5 $sink1

set sink2 [new Agent/TCPSink]
$sink2 set ecn_ 1
$ns attach-agent $n6 $sink2

set null3 [new Agent/Null]
$ns attach-agent $n7 $null3

$ns connect $tcp0 $sink1
$ns connect $tcp1 $sink2
$ns connect $udp2 $null3

#Tracing the queue
set qsize [open BLUEsize.tr w]
set qbw [open BLUEthput.tr w]
set qlost [open BLUElost.tr w]

set qfile [$ns monitor-queue $n3 $n4 [open queue.tr w] 0.05]
[$ns link $n3 $n4] queue-sample-timeout;

proc record {} {
global ns qfile qsize qbw qlost
set time 1
set now [$ns now]
$qfile instvar parrivals_ pdepartures_ bdrops_ bdepartures_ pdrops_
pkts_
puts $qsize "$now [expr $pkts_]"
puts $qbw "$now [expr $bdepartures_*8/1024/$time]"
set bdepartures_ 0
puts $qlost "$now [expr $pdrops_/$time]"
set pdrops_ 0
$ns at [expr $now+$time] "record"
}

#Set time
$ns at 0.0 "record"
$ns at 0.0 "$ftp0 produce 5000"
$ns at 0.0 "$ftp1 produce 8000"
$ns at 10 "$cbr2 start"
$ns at 145 "$cbr2 stop"

#Call the finish procedure
$ns at 150 "finish"

#Run the simulation
$ns run
```

4. NETWORK WITH SENDER WIRELESS NODES

```
#Global options
Mac/802_11 set dataRate_ 11.0Mb
set opt(chan)    Channel/WirelessChannel;  # channel type
set opt(prop)    Propagation/TwoRayGround; #  radio-propagation model
set opt(netif)   Phy/WirelessPhy;          # network interface type
set opt(mac)     Mac/802_11;               # MAC type
set opt(ifq)     Queue/DropTail/PriQueue;  # interface queue type
set opt(ll)      LL;                       # link layer type
set opt(ant)     Antenna/OmniAntenna;      # antenna model
set opt(ifqlen)  15;                       # max packet in ifq
set opt(nn)      3;                        # number of mobilenodes
set opt(adhocRouting)   DSDV;              # routing protocol
set opt(cp)      "";                       # connection pattern file
set opt(sc)      "";                       # node movement file
set opt(x)       1500;                # x coordinate of topology
set opt(y)       1500;                # y coordinate of topology
set opt(seed)    0.0;                 # seed for random number gen.
set opt(stop)    150;                 # time to stop simulation
set opt(ftp1-start)      0.0
set opt(ftp2-start)      0.0
set opt(cbr-start)       10.0
set num_wired_nodes      4
set num_bs_nodes         1

# check for boundary parameters and random seed
if { $opt(x) == 0 || $opt(y) == 0 } {
  puts "No X-Y boundary values given for wireless topology\n"
}
if {$opt(seed) > 0} {
  puts "Seeding Random number generator with $opt(seed)\n"
  ns-random $opt(seed)
}

# create simulator instance
set ns_  [new Simulator]

# set up for hierarchical routing
$ns_ node-config -addressType hierarchical
AddrParams set domain_num_ 2;    # number of domains
lappend cluster_num 4 1;         # number of clusters in each domain
AddrParams set cluster_num_ $cluster_num
lappend eilastlevel 1 1 1 1 4;   # number of nodes in each cluster
AddrParams set nodes_num_ $eilastlevel;        # of each domain
#Open the trace files
set tracefd  [open red.tr w]
set namtrace [open red.nam w]
$ns_ trace-all $tracefd
$ns_ namtrace-all-wireless $namtrace $opt(x) $opt(y)

# Create topography object
set topo   [new Topography]

# define topology
$topo load_flatgrid $opt(x) $opt(y)
```

```
# create God
create-god [expr $opt(nn) + $num_bs_nodes]

#create wired nodes
set temp {0.0.0 0.1.0 0.2.0 0.3.0};
# hierarchical addresses for wired domain
for {set i 0} {$i < $num_wired_nodes} {incr i} {
set W($i) [$ns_ node [lindex $temp $i]]
}

# configure for base-station node
$ns_ node-config -adhocRouting $opt(adhocRouting) \
                 -llType $opt(ll) \
                 -macType $opt(mac) \
                 -ifqType $opt(ifq) \
                 -ifqLen $opt(ifqlen) \
                 -antType $opt(ant) \
                 -propType $opt(prop) \
                 -phyType $opt(netif) \
                 -channelType $opt(chan) \
                 -topoInstance $topo \
                 -wiredRouting ON \
                 -agentTrace ON \
                 -routerTrace OFF \
                 -macTrace OFF

#create base-station node
set temp {1.0.0 1.0.1 1.0.2 1.0.3};
# hier address to be used for wireless
;# domain
set BS(0) [$ns_ node [lindex $temp 0]]
$BS(0) random-motion 0;                 # disable random motion

#provide some co-ord (fixed) to base station node
$BS(0) set X_ 1.0
$BS(0) set Y_ 2.0
$BS(0) set Z_ 0.0

#configure for mobilenodes
$ns_ node-config -wiredRouting OFF

for {set j 0} {$j < $opt(nn)} {incr j} {
    set node_($j) [ $ns_ node [lindex $temp \
    [expr $j+1]] ]
    $node_($j) base-station [AddrParams addr2id \
    [$BS(0) node-addr]]
}

# Define parameteres for RED queue
Queue/RED set thresh_ 5
Queue/RED set maxthresh_ 10
Queue/RED set bytes_ false
Queue/RED set queue_in_bytes_ false
Queue/RED set linterm_ 1
Queue/RED set q_weight_ 0.0002
Queue/RED set drop_tail_ true
Queue/RED set setbit_ true

#Define parameters for Blue queue
```

```
Queue/Blue set drop_front_ false
Queue/Blue set bytes false
Queue/Blue set queue_in_bytes_ false
Queue/Blue set setbit_ true
Queue/Blue set decrement 0.00025
Queue/Blue set increment 0.0025
Queue/Blue set dhold-time 100ms
Queue/Blue set ihold-time 100ms
Queue/Blue set pmark 1

#create links between wired and BS nodes
$ns_ duplex-link $BS(0) $W(3) 2.5Mb 100ms RED
#$ns_ duplex-link $BS(0) $W(3) 2.5Mb 100ms Blue
$ns_ duplex-link $W(3) $W(0) 1Mb 1ms DropTail
$ns_ duplex-link $W(3) $W(1) 1Mb 1ms DropTail
$ns_ duplex-link $W(3) $W(2) 1Mb 1ms DropTail

#Monitor queue for NAM
$ns_ duplex-link-op $BS(0) $W(3) queuePos 0.5

#Maximum number of packets in the queue
$ns_ queue-limit $BS(0) $W(3) 15

# setup TCP and UDP connections
set tcp1 [new Agent/TCP/Reno]
$tcp1 set class_ 1
$tcp1 set window_ 32
$tcp1 set packetSize_ 960
$tcp1 set ecn_ 1
set sink1 [new Agent/TCPSink]
$sink1 set ecn_ 1
$ns_ attach-agent $node_(0) $tcp1
$ns_ attach-agent $W(0) $sink1
$ns_ connect $tcp1 $sink1
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
$ns_ at $opt(ftp1-start) "$ftp1 produce 5000"

set tcp2 [new Agent/TCP/Reno]
$tcp2 set class_ 2
$tcp2 set window_ 32
$tcp2 set packetSize_ 960
$tcp2 set ecn_ 1
set sink2 [new Agent/TCPSink]
$sink2 set ecn_ 1
$ns_ attach-agent $node_(1) $tcp2
$ns_ attach-agent $W(1) $sink2
$ns_ connect $tcp2 $sink2
set ftp2 [new Application/FTP]
$ftp2 attach-agent $tcp2
$ns_ at $opt(ftp2-start) "$ftp2 produce 8000"

set udp [new Agent/UDP]
$udp set class_ 3
set null [new Agent/Null]
$ns_ attach-agent $node_(2) $udp
$ns_ attach-agent $W(2) $null
$ns_ connect $udp $null
set cbr [new Application/Traffic/CBR]
```

```
$cbr set packetSize_ 1000
$cbr set rate_ 1Mb
$cbr attach-agent $udp
$ns_ at $opt(cbr-start) "$cbr start"

#Tracing the queue
set qsize [open REDsize.tr w]
set qbw [open REDthput.tr w]
set qlost [open REDloss.tr w]

set qfile [$ns_ monitor-queue $BS(0) $W(3) [open queue.tr w] 0.05]
[$ns_ link $BS(0) $W(3)] queue-sample-timeout;

proc record {} {
global ns_ qfile qsize qbw qlost
set time 1
set now [$ns_ now]
$qfile instvar parrivals_ pdepartures_ bdrops_ bdepartures_ pdrops_
pkts_
puts $qsize "$now [expr $pkts_]"
puts $qbw "$now [expr $bdepartures_*8/1024/$time]"
set bdepartures_ 0
puts $qlost "$now [expr $pdrops_/$time]"
set pdrops_ 0
$ns_ at [expr $now+$time] "record"
}
$ns_ at 0.0 "record"


# source connection-pattern and node-movement scripts Mac/802_11 set
PLCPDataRate_  1.0e6
if { $opt(cp) == "" } {
  puts "*** NOTE: no connection pattern specified."
set opt(cp) "none"
} else {
  puts "Loading connection pattern..."
  source $opt(cp)
}
if { $opt(sc) == "" } {
  puts "*** NOTE: no scenario file specified."
set opt(sc) "none"
} else {
  puts "Loading scenario file..."
  source $opt(sc)
  puts "Load complete..."
}

# Define initial node position in nam

for {set i 0} {$i < $opt(nn)} {incr i} {

    $ns_ initial_node_pos $node_($i) 20
}

# Tell all nodes when the simulation ends
for {set i } {$i < $opt(nn) } {incr i} {
    $ns_ at $opt(stop).0 "$node_($i) reset";
}
$ns_ at $opt(stop).0 "$BS(0) reset";
```

```
$ns_ at $opt(stop).0002 "puts \"NS EXITING...\" ; $ns_ halt"
$ns_ at $opt(stop).0001 "stop"
proc stop {} {
global ns_ tracefd namtrace qsize qbw qlost
close $tracefd
close $namtrace
close $qsize
close $qbw
close $qlost
exec  xgraph  REDsize.tr  -geometry  800x400  -t  "Queue  size"  -x
"Seconds" -y "Packets" &
exec xgraph REDthput.tr -geometry 800x400 -t "Throughput" -x "secs"
-y "Kbps" -fg white &
exec xgraph REDloss.tr -geometry 800x400 -t "Packet loss rate" -x
"Seconds" -y "Packets" &
}

# informative headers for CMUTracefile
puts $tracefd "M 0.0 nn $opt(nn) x $opt(x) y $opt(y) rp \
$opt(adhocRouting)"
puts $tracefd "M 0.0 sc $opt(sc) cp $opt(cp) seed $opt(seed)"
puts $tracefd "M 0.0 prop $opt(prop) ant $opt(ant)"

puts "Starting Simulation..."
$ns_ run
```

5. NETWORK WITH WIRELESS RECEIVERS

```
#Global options
Mac/802_11 set dataRate_ 11.0Mb
set opt(chan)    Channel/WirelessChannel;  # channel type
set opt(prop)    Propagation/TwoRayGround; #  radio-propagation model
set opt(netif)   Phy/WirelessPhy;          # network interface type
set opt(mac)     Mac/802_11;               # MAC type
set opt(ifq)     Queue/DropTail/PriQueue;  # interface queue type
set opt(ll)      LL;                       # link layer type
set opt(ant)     Antenna/OmniAntenna;      # antenna model
set opt(ifqlen)  15;                       # max packet in ifq
set opt(nn)      3;                        # number of mobilenodes
set opt(adhocRouting)   DSDV;              # routing protocol
set opt(cp)      "";                       # connection pattern file
set opt(sc)      "";                       # node movement file
set opt(x)       1500;                # x coordinate of topology
set opt(y)       1500;                # y coordinate of topology
set opt(seed)    0.0;                 # seed for random number gen.
set opt(stop)    150;                 # time to stop simulation
set opt(ftp1-start)      0.0
set opt(ftp2-start)      0.0
set opt(cbr-start)       10.0
set num_wired_nodes      4
set num_bs_nodes         1

# check for boundary parameters and random seed
if { $opt(x) == 0 || $opt(y) == 0 } {
  puts "No X-Y boundary values given for wireless topology\n"
}
if {$opt(seed) > 0} {
  puts "Seeding Random number generator with $opt(seed)\n"
  ns-random $opt(seed)
}

# create simulator instance
set ns_  [new Simulator]

# set up for hierarchical routing
$ns_ node-config -addressType hierarchical
AddrParams set domain_num_ 2;    # number of domains
lappend cluster_num 4 1;         # number of clusters in each domain
AddrParams set cluster_num_ $cluster_num
lappend eilastlevel 1 1 1 1 4;   # number of nodes in each cluster
AddrParams set nodes_num_ $eilastlevel ;       # of each domain

#Open the trace files
set tracefd  [open red.tr w]
set namtrace [open wifidest.nam w]
$ns_ trace-all $tracefd
$ns_ namtrace-all-wireless $namtrace $opt(x) $opt(y)

# Create topography object
set topo   [new Topography]

# define topology
$topo load_flatgrid $opt(x) $opt(y)
```

```
# create God
create-god [expr $opt(nn) + $num_bs_nodes]

#create wired nodes
set temp {0.0.0 0.1.0 0.2.0 0.3.0};
# hierarchical addresses for wired domain
for {set i 0} {$i < $num_wired_nodes} {incr i} {
  set W($i) [$ns_ node [lindex $temp $i]]
}

# configure for base-station node
$ns_ node-config -adhocRouting $opt(adhocRouting) \
                 -llType $opt(ll) \
                 -macType $opt(mac) \
                 -ifqType $opt(ifq) \
                 -ifqLen $opt(ifqlen) \
                 -antType $opt(ant) \
                 -propType $opt(prop) \
                 -phyType $opt(netif) \
                 -channelType $opt(chan) \
                 -topoInstance $topo \
                 -wiredRouting ON \
                 -agentTrace ON \
                 -routerTrace OFF \
                 -macTrace OFF

#create base-station node
set temp {1.0.0 1.0.1 1.0.2 1.0.3};
# hier address to be used for wireless
;# domain
set BS(0) [$ns_ node [lindex $temp 0]]
$BS(0) random-motion 0;             # disable random motion

#provide some co-ord (fixed) to base station node
$BS(0) set X_ 1.0
$BS(0) set Y_ 2.0
$BS(0) set Z_ 0.0

#configure for mobilenodes
$ns_ node-config -wiredRouting OFF

for {set j 0} {$j < $opt(nn)} {incr j} {
    set node_($j) [ $ns_ node [lindex $temp \
    [expr $j+1]] ]
    $node_($j) base-station [AddrParams addr2id \
    [$BS(0) node-addr]]
}

# Define parameteres for RED queue
Queue/RED set thresh_ 5
Queue/RED set maxthresh_ 10
Queue/RED set bytes_ false
Queue/RED set queue_in_bytes_ false
Queue/RED set linterm_ 1
Queue/RED set q_weight_ 0.0002
Queue/RED set drop_tail_ true
Queue/RED set setbit_ true
```

```
#Define parameters for Blue queue
Queue/Blue set drop_front_ false
Queue/Blue set bytes false
Queue/Blue set queue_in_bytes_ false
Queue/Blue set setbit_ true
Queue/Blue set decrement 0.00025
Queue/Blue set increment 0.0025
Queue/Blue set dhold-time 100ms
Queue/Blue set ihold-time 100ms
Queue/Blue set pmark 1

#create links between wired and BS nodes
$ns_ duplex-link $W(3) $BS(0) 2.5Mb 100ms RED
#$ns_ duplex-link $W(3) $BS(0) 2.5Mb 100ms Blue
$ns_ duplex-link $W(0) $W(3) 1Mb 1ms DropTail
$ns_ duplex-link $W(1) $W(3) 1Mb 1ms DropTail
$ns_ duplex-link $W(2) $W(3) 1Mb 1ms DropTail

#Monitor queue for NAM
$ns_ duplex-link-op $W(3) $BS(0) queuePos 0.5

#Maximum number of packets in the queue
$ns_ queue-limit $W(3) $BS(0) 15

# setup TCP and UDP connections
set tcp1 [new Agent/TCP/Reno]
$tcp1 set class_ 1
$tcp1 set window_ 32
$tcp1 set packetSize_ 960
$tcp1 set ecn_ 1
set sink1 [new Agent/TCPSink]
$sink1 set ecn_ 1
$ns_ attach-agent $W(0) $tcp1
$ns_ attach-agent $node_(0) $sink1
$ns_ connect $tcp1 $sink1
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
$ns_ at $opt(ftp1-start) "$ftp1 produce 5000"

set tcp2 [new Agent/TCP/Reno]
$tcp2 set class_ 2
$tcp2 set window_ 32
$tcp2 set packetSize_ 960
$tcp2 set ecn_ 1
set sink2 [new Agent/TCPSink]
$sink2 set ecn_ 1
$ns_ attach-agent $W(1) $tcp2
$ns_ attach-agent $node_(1) $sink2
$ns_ connect $tcp2 $sink2
set ftp2 [new Application/FTP]
$ftp2 attach-agent $tcp2
$ns_ at $opt(ftp2-start) "$ftp2 produce 8000"

set udp [new Agent/UDP]
$udp set class_ 3
set null [new Agent/Null]
$ns_ attach-agent $W(2) $udp
$ns_ attach-agent $node_(2) $null
$ns_ connect $udp $null
```

```
set cbr [new Application/Traffic/CBR]
$cbr set packetSize_ 1000
$cbr set interval_ 0.008
$cbr attach-agent $udp
$ns_ at $opt(cbr-start) "$cbr start"

#Tracing the queue
set qsize [open REDsize.tr w]
set qbw [open REDthput.tr w]
set qlost [open REDloss.tr w]

set qfile [$ns_ monitor-queue $W(3) $BS(0) [open queue.tr w] 0.05]
[$ns_ link $W(3) $BS(0)] queue-sample-timeout;

proc record {} {
global ns_ qfile qsize qbw qlost
set time 1
set now [$ns_ now]
$qfile instvar parrivals_ pdepartures_ bdrops_ bdepartures_ pdrops_
pkts_
puts $qsize "$now [expr $pkts_]"
puts $qbw "$now [expr $bdepartures_*8/1024/$time]"
set bdepartures_ 0
puts $qlost "$now [expr $pdrops_/$time]"
set pdrops_ 0
$ns_ at [expr $now+$time] "record"
}
$ns_ at 0.0 "record"
#$ns_ at 0.0 "ftp1 produce 1000"

# source connection-pattern and node-movement scripts Mac/802_11 set
PLCPDataRate_  1.0e6
if { $opt(cp) == "" } {
  puts "*** NOTE: no connection pattern specified."
set opt(cp) "none"
} else {
  puts "Loading connection pattern..."
  source $opt(cp)
}
if { $opt(sc) == "" } {
  puts "*** NOTE: no scenario file specified."
set opt(sc) "none"
} else {
  puts "Loading scenario file..."
  source $opt(sc)
  puts "Load complete..."
}

# Define initial node position in nam

for {set i 0} {$i < $opt(nn)} {incr i} {

    $ns_ initial_node_pos $node_($i) 20
}

# Tell all nodes when the simulation ends
for {set i } {$i < $opt(nn) } {incr i} {
    $ns_ at $opt(stop).0 "$node_($i) reset";
}
```

```
$ns_ at $opt(stop).0 "$BS(0) reset";

$ns_ at $opt(stop).0002 "puts \"NS EXITING...\" ; $ns_ halt"
$ns_ at $opt(stop).0001 "stop"
proc stop {} {
global ns_ tracefd namtrace qsize qbw qlost
close $tracefd
close $namtrace
close $qsize
close $qbw
close $qlost
exec  xgraph  REDsize.tr  -geometry  800x400  -t  "Queue  size"  -x
"Seconds" -y "Packets" &
exec xgraph REDthput.tr -geometry 800x400 -t "Throughput" -x "secs"
-y "Kbps" -fg white &
exec xgraph REDloss.tr -geometry 800x400 -t "Packet loss rate" -x
"Seconds" -y "Packets" &
}

# informative headers for CMUTracefile
puts $tracefd "M 0.0 nn $opt(nn) x $opt(x) y $opt(y) rp \
$opt(adhocRouting)"
puts $tracefd "M 0.0 sc $opt(sc) cp $opt(cp) seed $opt(seed)"
puts $tracefd "M 0.0 prop $opt(prop) ant $opt(ant)"

puts "Starting Simulation..."
$ns_ run
```