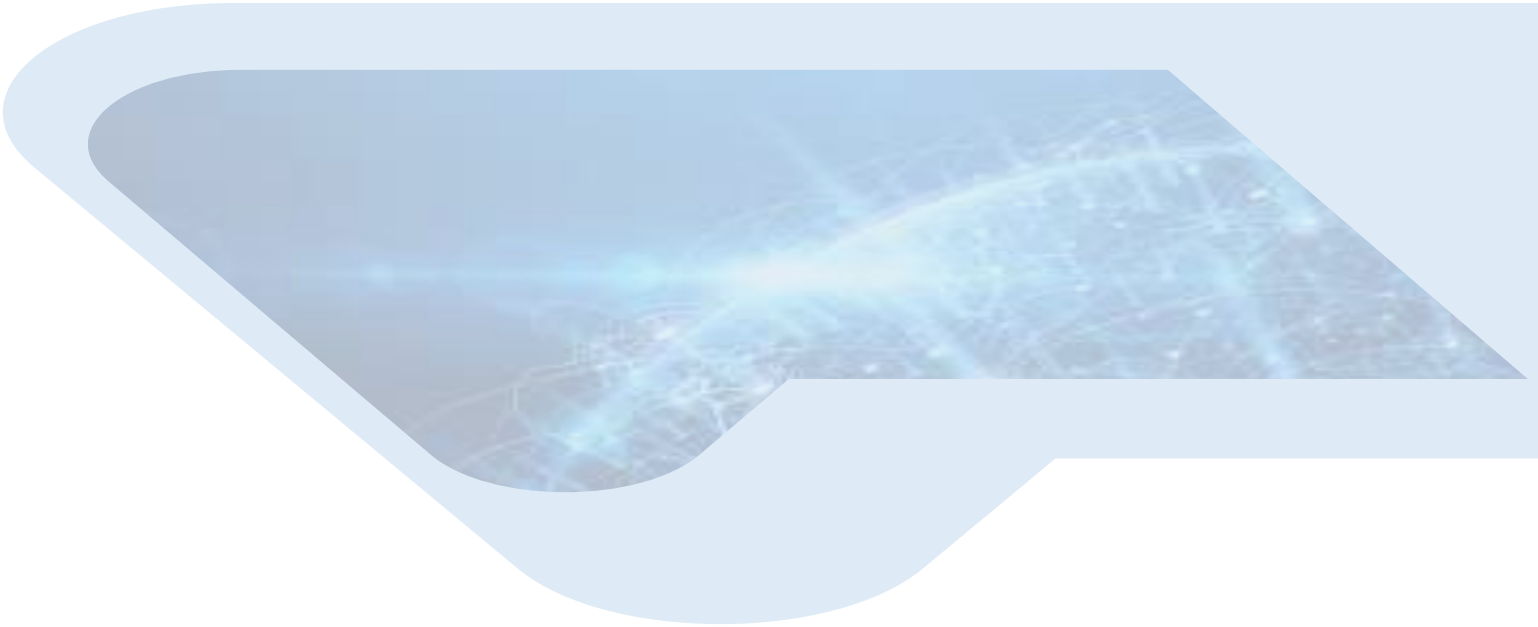
ITEC5100: Planning and Design of Computer Networks
School of Information Technology
Carleton University



Master Project Report on

Designing and Implementation of Heterogeneous Network for Supporting Quality of Service Communications

Participants:
Rana Abou Khamis
Sam Kayyali

Supervisor:
Prof. Jun Steed Huang

Date of Submission:
March 23, 2020

Table of Contents

Introduction

Business Case Overview

In different applications, communication between wired networks and ad hoc wireless networks is required. Connections can be made through bridging both networks (wired-cum-wireless topology) to form a heterogeneous network that achieves reliable communication and best performance (QoS).

In our proposal, in the 1st stage, we are going to build both types of networks separately, carry out trials to select the best routing protocol (Distance Vector, Link state, or hybrid of both). We are taking into consideration the existence of network bottlenecks, create proper backup plans for link restoration, and achieve the desirable network performance by avoiding network link failures. In the 2nd stage, we are going to bridge both networks and simulate the performance of the heterogeneous (wired-cum-wireless) system.

We are going to simulate Mobile Wireless Sensor Networks (MWSN) with Alert Emergency System (AES) that support two-way feedback as needed using different communication layer. MWSN connectivity relies on the principles of ad-hoc networks with a collection of mobile sensor devices that collect information related to temperature humidity, pressure, etc. MWSN sensors can be deployed without fixed infrastructure. AES provides rapid alert across wired and wireless sensors devices. The AES captures sensor events from physical security systems such as Access Control, Fire Systems Gas panel, and Gunshots Detections.

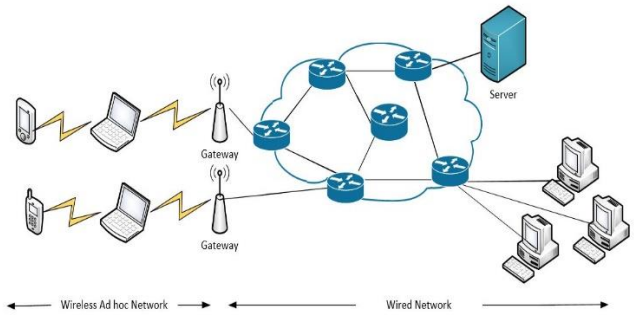


Figure 1 Heterogeneous Network (Wired-cum-Wireless)

Scope

The scope of this project is to simulate the performance of network protocols over MWSN and Wired Network and build an architecture that supports the transmission of data through air interface between wired and wireless networks. In this project, we will assess and evaluate the metrics generated from the simulation, so that in the end we will have an analyzable model, which approximates a real system to some level of accuracy and reality.

To implement the proposed system, we need to:

- Compare Distance Vector & Link State routing protocols in a wired IP network.
- Compare ad hoc On-Demand Distance Vector (AODV) and Dynamic Source Routing Protocol (DSDV) in ad hoc networks.
- Consider the differences between the two types of networks in the network layer (routing protocols) when bridging ad hoc networks with wired IP networks to achieve efficient connectivity.
- Integrate the two architectures wired IP networks and ad hoc networks
- Analyze the model when building two types of systems, taking into consideration the existence of a network bottleneck.

Problem Statement

Different applications require a combination of different methodologies to achieve the ultimate goal. The Alert Emergency Systems with 2-way feedback could be one of these applications with wireless sensors designed to collect some information and forward it to a central server and, in turn, send feedback to the sensors to take action, if needed. The fact that wireless networks classified into infrastructure networks and ad-hoc networks; therefore, wireless sensors could be fixed or mobile nodes.

The objective of this project is to simulate the performance of Wireless Sensor Network with the AES system that needs to communicate with the remote server, which plays the role of decision-maker in response to the data received from WSN and all information coming from the proposed terminology.

Therefore, we need to study and simulate data transmission from a wireless sensor network to a wired network and vice versa through gateways. Also, the simulation provides insights into the performance of the wired network using different algorithms, with the existence of bottlenecks, and with data transmission from various sources, taking into consideration the routing issues between the wired and wireless nodes. The main challenge in this type of topology is to select efficient routing protocol at the Network layer and meet the QoS at Transport Layer. In this project, we will mainly focus on how to exchange packets between wired and wireless nodes and analyze the impacts of network elements failures.

Methodology

- Build a wired network and analyze the impacts of network element failures.
- Build an Ad hoc network and analyze the impacts of network element failures.
- Use hierarchical routing to integrate Ad hoc network and wired network and analyze the impacts of network element failures.
- Use Automation to run the simulation multiple time.

Tools, Programming, and Machine Features

- OS: Linux UBUNTU 16.04
- Memory: RAM 8 GB
- Processor: Intel Core i5
- Simulator: Network Simulator (NS2)
- Programming Language: TCL, C++, awk, GNUPLOT, Perl, and shell script.

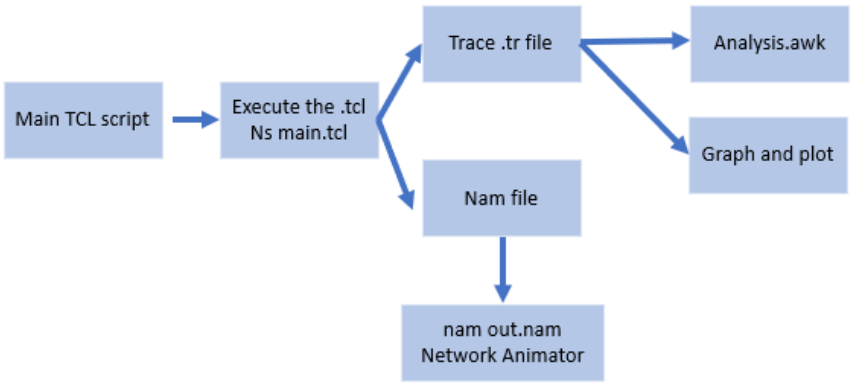


Table 1 Simulation process using NS2

Task, File and SC Management

We use Github to share our project.

Terminology

- Mobile Wireless Sensor Networks (MWSN)
- Destination Sequenced distance vector (DSDV)
- Ad- Hoc On-Demand Distance Vector (AODV)
- Quality of Service (QoS)

Simulation Overview

This study is applied to many applications based on heterogeneous networks that seek to get better Quality of Service (QoS). In our simulation, we link wired nodes through links, while the wireless nodes, we connect them using a fixed Base Station node as a gateway for the wired topology. This Base Station will deliver the packet into and out of the wireless topology. The mobile sensor can move out of the base station range, and it should connect to another base station and consider it as a gateway to the wired network. If there is no route to the Base Station, the packet will be dropped. We compare two routing protocols in this project: DSDV and AODV to consider the performance in the wire-cum-wireless network.

Our aim in choosing the parameters was not to obtain necessarily an optimal performance but rather to create conditions that allow us to study the effect of a link failure, and the impact on TCP performance (delay, throughput). For that reason, we chose the same parameters for the two priority levels (this will be explained below). To build this network, we use Hierarchical routing and two domains of addresses, one for the wired nodes and one for wireless nodes. Each node has one hierarchical address.

Proposed Approach and Scenarios

Topology

We will briefly describe the wired and wireless topology. In Ns2, the topology has to be created first. The wireless topology consists of one Base Station (node 10) and six wireless nodes (11,12,13,14,15 and 16). While the wired network consists of 10 wired nodes and links between them using symmetric (duplex) links with specified bandwidth and delay. We use the Link down and link up commands to change the link status to up or down. Also, we set equal link cost for all connections between nodes.

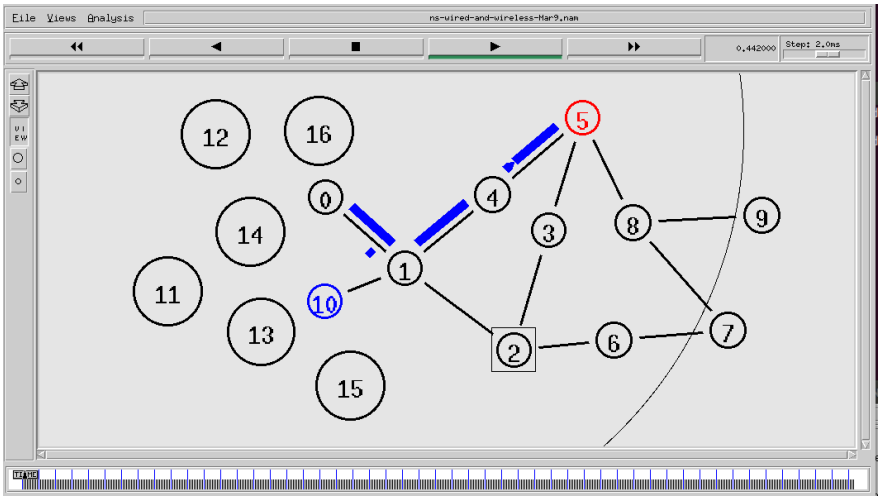


Figure 2 A snapshot of the simulation topology in NAM for Wired-Wireless Topology

Simulation Parameters

Network Elements	No. of Network Elements
Topology	800 x 800
Traffic Source	5
Gateway (Base station)	1
Simulation time	100s
Network Nodes	10 wired, 6 wireless
Packet size	1000B
Routing Protocols	DSDV and AODV
Source Type	FTP (TCP)
No of source	7
interval	0.05s

Table 2 Simulation Parameters

Routing Protocol

Wired Topology: we use dynamic routing protocol "Distance Vector Routing" that is implemented based on Distributed Bellman-Ford routing that sends and receive route update every interval time and compute the topology based in the messages exchanged. Also, the agent sends updates whenever changes occur in the topology like link failure. There are various routing possibilities, and we selected the routing protocol, which chooses the shortest route in terms of link cost and bandwidth.

Wireless and ad-hoc networks: In ad hoc networks, routing protocol classified into three categories: Proactive, Reactive, Hybrid. We run two main simulations using Destination Sequenced Distance Vector (DSDV) and Ad-Hoc on-Demand Distance Vector (AODV). Both protocols are distance vector type routing.

In DSDV, each node has a routing table and periodically broadcast routing updates with all available destinations. DSDV is suitable for small ad hoc networks, but it requires a regular update of its routing tables, which uses a small amount of bandwidth and reduces network traffic. The routing information is always available regardless of the node needs information or not.

In AODV, nodes are not required to maintain routes to destinations that are not actively used. It requires a low delay for connection setup but requires massive control overhead, which leads to bandwidth consumption. It uses sequence numbers to check the age of routes. It uses REQUEST to build a route and REPLY as a query.

Wireless Node Interface

It is essential to define wireless node characteristics before creating them. We select the hierarchical type for addressing structure in our simulation, DSDV, and AODV for the ad hoc routing protocol for wireless node. We turn to trace on at the agent and router level.

Addressing Type: hierarchical
Ad hoc Routing: AODV or DSDV
Link-Layer Type: Logical Link (LL) type
Mac Type: Mac/802_11
Queue Type: Drop Tail
Channel Type: Wireless
Antenna model: OmniAntenna

Queuing Management

Queues can hold or drop the packet, and using different types of queues will have a different decision in choosing which packets should be serviced or dropped. We consider the network topology with three bottlenecks, and we use the default value Drop tail (FIFO) for the queue management. When there are dropped packets, TCP reduces the window size for the session considerably. All sessions that experienced packet loss from tail drop will decrease their window size at about the same time resulting in a queue reduction. However, the queue builds up again very quickly, and the process repeats itself because there is a lot of data to be sent.

Queue Monitoring

We use monitor-queue to collect information on queue length, losses, and arrivals and departures. The output file has many columns, including time, input and output nodes, queue size in packets, number of packets have arrived, number of packets have departed, number of packets dropped, number of bytes have arrived and departed. To monitor between node 5 and node 4:

```
set qfile [$ns_ monitor-queue $W(4) $W(5)  [open queue4-5.tr w] 0.05]
[$ns_ link $W(4) $W(5)] queue-sample-timeout;
```

TCP and Window Flow Control

In our simulation, we use the TCP agent that adapts the transmission rate of packets based on the assigned bandwidth. With TCP, we guaranteed a reliable connection by retransmitting lost packets and avoid congestions. This means the source is forced to wait and stop transmission if acknowledge does not reach. With a sequence number, TCP able to retransmit the dropped packet. The performance of TCP can be affected by the number and location of wireless nodes. Delay and drop rates of multiple wireless nodes add up, making loss recovery and congestion control more difficult.

TCP agent in ns2 does not generate application data. Therefore, we connect TCP agent with FTP application. TCP agent has many variables, including windows size. We set the size of the window to 50 for all TCP agents, packet size to 1000, and overhead equal to zero. For each TCP agent as a source, we configure TCP sink to return ACK to a TCP source for each received packet. We kept the default ACK size 40.

There are seven source nodes, and each one of them generates TCP connections. We experiment with 5 Mbps (short propagation delays) with completely symmetric links. Links between the edge nodes and the corresponding source nodes have delays of 0.5 sec. Here are the flows:

Flow	Type	Source	Destination
Flow 1	wired - wired	W(0)	W(5) server
Flow 2	wireless - wired	Node 11	W(5) server
Flow 3	wireless - wireless	Node 13	Node 16
Flow 4	wired - wired	W(5) server	W(0)
Flow 5	wireless - wired	Node 14	W(5) server
Flow 6	wired - wireless	W(5) server	Node 11
Flow 7	wireless - wireless	Node 15	Node 11

Table 3 TCP Source Flows

Link Failure

To simulate and model noisy links and link disconnection between two nodes for a certain time, we use rtmodel up and down for two links between nodes 1 and 4 and between node two and node 3. We set the time in a way to make both links down to force all traffic to redirect to the link between node 8 and node 5 (server). In this way, we model bottleneck scenarios with bursty traffic.

```
$ns rtmodel-at 1.0 down $W(5) $W(4)
$ns rtmodel-at 10.5 up W(5) $W(4
```

Random Variables and Seeds

In our scripts, we use random variables that use the seed for beginning the FTP connections. We give the seed value of 0 to generate a new random variable each time we run the simulation. If we want to create the same values of random variables, we should use seed not equal to zero.

Simlution Results

We have two output files generated from running the simulation .tr and .nam. The trace file .tr contains two types of trace formats, wired and wireless traces because of heterogeneous topology.

```
+ 1.048801 5 4 ack 40 ------- 1 0.5.0.0 0.0.0.0 545 1411
- 1.048801 5 4 ack 40 ------- 1 0.5.0.0 0.0.0.0 545 1411
r 1.048829 1 0 ack 40 ------- 1 0.5.0.0 0.0.0.0 544 1409
+ 1.048829 0 1 tcp 1040 ------- 1 0.0.0.0 0.5.0.0 576 1412
r 1.048957901 _13_ AGT  --- 1286 ack 60 [13a 3 6 800] ------- [4194310:0 4194307:0 32 4194307] [44 0] 1 0
s 1.048957901 _13_ AGT  --- 1413 tcp 1040 [0 0 0 0] ------- [4194307:0 4194310:0 32 0] [65 0] 0 0
r 1.048957901 _13_ RTR  --- 1413 tcp 1040 [0 0 0 0] ------- [4194307:0 4194310:0 32 0] [65 0] 0 0
f 1.048957901 _13_ RTR  --- 1413 tcp 1060 [0 0 0 0] ------- [4194307:0 4194310:0 32 4194310] [65 0] 0 0
r 1.049177 0 1 tcp 1040 ------- 1 0.0.0.0 0.5.0.0 553 1362
+ 1.049177 1 4 tcp 1040 ------- 1 0.0.0.0 0.5.0.0 553 1362
r 1.049365 5 4 ack 40 ------- 1 0.5.0.0 0.0.0.0 545 1411
+ 1.049365 4 1 ack 40 ------- 1 0.5.0.0 0.0.0.0 545 1411
- 1.049365 4 1 ack 40 ------- 1 0.5.0.0 0.0.0.0 545 1411
- 1.049433 1 4 tcp 1040 ------- 1 0.0.0.0 0.5.0.0 548 1352
```

The format in the red box is for the wireless nodes, and the blue box is for the wired topology. In both formats, the first field is the event type, and the second is time.

Wireless Trace format:

<event> <time> <destination node id> <packet layer AGT/RTR/IFG> --- <sequence no> <packet type tcp,cbr,ack> <size> [exp time MAC sender ID Mac Receiver ID MAC type]>----<[source IP Destination IP TTL]><[tcp seq no tcp ck no]>

Event: r: receive, s: send, f:forward, D: drop.

Wired Trace format:

<event> <time> <from node > <To node><packet type tcp,cbr,ack><packet size >------ <Src addr><Dst addr><seq num><pkt id>

Event: (+): enqueue, (-): dequeue, r: receive, d: drop.

Performance Parameters

We perform statistical analysis on the trace file (.tr) and using awk script. We divided the study for the wired network and wireless sensors network. The metrics considered in this analysis include:

- **Number of packets sent**
- **Number of packets received**
- **Packet Delivery Ratio**: the ratio of a total number of received packet at a destination to the total number of data packets sent from the source.
- **Packet Dropping Ratio:** the ratio of total dropped packet to the total number of sent packets from the source. **Throughput:** average number of bits transmitted successfully from source node to destination per unit time. It improved by increasing node density.
- **Normalized routing overhead:** the ratio of number of routing packets to the total received packets.
- **Control overhead:** the number of routing packets
- **Delay:** time taken by packet to reach to the destination from the source.

Evaluating protocols over wireless links, we focus on throughput in assessing performance. High throughput means efficient use of radio spectrum and lower interference. To make these calculations possible, we had to prepare (.AWK) file, including massive code, because each parameter needs a separate approach in AWK coding to calculate these values.

Wired Analysis in Wired Topology using DSDV

```
No_of_Packets_Sent:                499859.00
No_of_Packets_Recieved:            497317.00
No of Dropped Packet:                2523.00
Packet_Delivery_Ratio:                99.49 %
Packet_Dropping_Ratio:                 0.50 %
Throughput:                           40.37 [Mbps]
Delay:                                 0.02 Seconds
```

Wireless Analysis in Ad hoc Topology using DSDV

```
No_of_Packets_Sent:             8250
No_of_Packets_Received:         8104
No_of_Packets_Dropped:          476
Packet_Delivery_Ratio:          98.23 %
Packet_Dropping_Ratio:          5.77 %
Throughput:                     686.76 Kbps
Normalized_Routing_Overhead:    0.89 %
Control_Overhead:               72
Delay:                          0.64 Seconds
```

Wired-cum-Wireless Analysis using DSDV

Wired Analysis

```
No_of_Packets_Sent:                354935.00
No_of_Packets_Recieved:            354298.00
No of Dropped Packet:                 578.00
Packet_Delivery_Ratio:                 99.82 %
Packet_Dropping_Ratio:                  0.16 %
Throughput:                            28.77 [Mbps]
Delay:                                  0.03 Seconds
```

- In a wired network, more data is circulating, as there are many data sources transmitting data through a wired network is less vulnerable to external circumstances causing fewer packets to drop and the vast majority of packets to be received.
- Data rates in the wired network are almost(~29 Mbps)

Wireless Analysis

We can create a new file that consists of only lines for wireless traces from the main trace file. Divide the trace file to Wireless traces:

grep " --- " ns-wired-and-wireless.tr > wireless_traces.tr

```
No_of_Packets_Sent:             7602
No_of_Packets_Received:         6287
No_of_Packets_Dropped:          44
Packet_Delivery_Ratio:          82.70 %
Packet_Dropping_Ratio:          0.58 %
Throughput:                     533.10 Kbps
Normalized_Routing_Overhead:    1.10 %
Control_Overhead:               69
Delay:                          5.74 Seconds
```

The observations that can be induced from the above parameters for a wireless sensor network:

- A small amount of data being transmitted over the wireless network, and this is something related to the nature of sensor networks that transmits a small amount of data over the air interface. Due to the small amount of data being sent over the wireless network, the loss of packets will have an impact on packet delivery/packet dropping ratios.
- Due to the fact discussed above, throughput will show low values (low circulated traffic). The overhead will be more since wireless routing will be more vulnerable to noise or environmental circumstances that require additional bits for the correct routing and better protection.

Wired-cum-Wireless Analysis using AODV

Wired Analysis

```
No_of_Packets_Sent:                355311.00
No_of_Packets_Recieved:            354703.00
No_of_Dropped_Packet:                 536.00
Packet_Delivery_Ratio:                 99.83 %
Packet_Dropping_Ratio:                  0.15 %
Throughput:                            28.79 [Mbps]
Delay:                                  0.03 Seconds
```

Wireless Analysis

```
No_of_Packets_Sent:             7856
No_of_Packets_Received:         6197
No_of_Packets_Dropped:          223
Packet_Delivery_Ratio:          78.88 %
Packet_Dropping_Ratio:          3.60 %
Throughput:                     478.32 Kbps
Normalized_Routing_Overhead:    1.89 %
Control_Overhead:               78
Delay:                          8.03 Seconds
```

The observations found in DSDV protocol in the previous section apply to the metrics of AODV-based wireless network in terms of the amount of traffic and throughput in both networks (wired and wireless)

Comparing the results of DSDV-based wireless network performance with AODV-based wireless network performance, we can see that DSDV protocol is performing better than AODV protocol in our proposed topology. For this reason, we will continue with the DSDV protocol on the wireless network for the rest of the analysis.

TCP Congestions Window Information

Using the function ***WinVsTime*** in the main TCL file to obtain windows information from all TCP sources and save them in ***win*** file. The output file formatted to contain Time column and TCP window flow columns for each source.

PlotTCPWin Function

```
proc plotTCPWin {tcpSource file k} {
global ns_ NumbSrc
set time 0.03
set now [$ns_ now]
set cwnd [$tcpSource set cwnd_]
if {$k == 1} {
  puts -nonewline $file "$now \t $cwnd \t"
 } else {
  if {$k < $NumbSrc } {
  puts -nonewline $file "$cwnd \t" } }
if { $k == $NumbSrc } {
  puts -nonewline $file "$cwnd \n" }
$ns_ at [expr $now+$time] "plotTCPWin  $tcpSource $file $k" }
```

We use the following commands in Gnuplot to plot the Windows trend for each TCP flow:

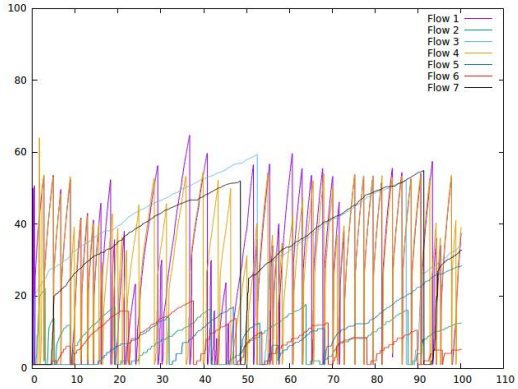


Figure 3 Windows Size of All TCP connections DSDV

TCP Windows size for wired nodes

```
plot "win" using 1:2 t "Flow 1" w lines linetype 1, "win" using 1:5 t "Flow 4" w lines  linetype 4
```

At the beginning of the simulation, we notice a transient behavior in TCP as it will be in the slow-start phase. Around time 5 sec onwards, TCP starts a kind of steady-state cyclic performance, it enters congestion avoidance status, and window size increases until congestion takes place.

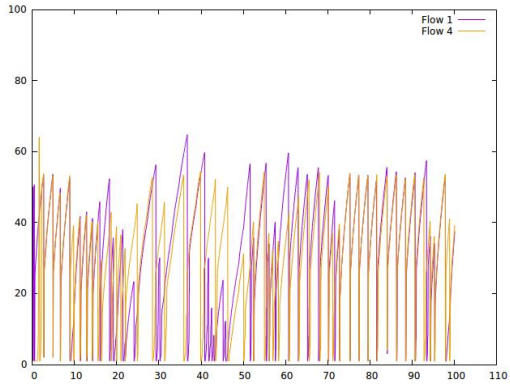


Figure 4 TCP Window size for Wired nodes with DSDV

TCP Windows size for wireless nodes

```
plot "win" using 1:4 t "Flow 3" w lines linetype 3, "win" using 1:9 t "Flow 8" w lines  linetype 8
```

In wireless nodes, there is a difference in TCP window size. More packets can be sent in the TCP window before congestion takes place, and the time each TCP window takes is much more compared to the time taken by wired transmissions. This means that it takes longer time to reach the congestion status in TCP transmission over mobile wireless sensor network (MWSN)

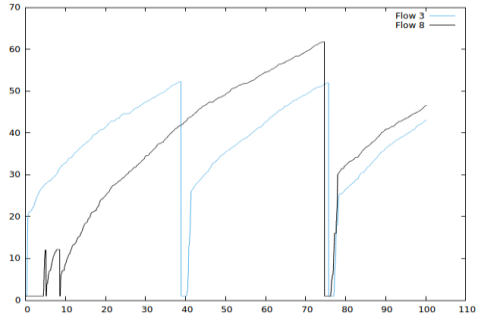


Figure 5 TCP Window Size in Wireless nodes with DSDV

Queue Length

As we mentioned, we monitored three queues using the following code in tcl:

```
set qfile1 [$ns_ monitor-queue $W(8) $W(5)  [open queue8-5.tr w] 0.05]
[$ns_ link $W(8) $W(5)] queue-sample-timeout;
set qfile [$ns_ monitor-queue $W(4) $W(5)  [open queue4-5.tr w] 0.05]
[$ns_ link $W(4) $W(5)] queue-sample-timeout;
set qfile [$ns_ monitor-queue $W(3) $W(5)  [open queue3-5.tr w] 0.05]
[$ns_ link $W(3) $W(5)] queue-sample-timeout;
```

To plot queues size in packet using gnuplot:

```
set yrange [0.0:100.0]
plot "queue4-5.tr" using 1:5 t " Queue between Node 4 and Server " w lines linetype 2, "queue3-5.tr" using 1:5 t " Queue between Node 3 and Server" w lines linetype 7, "queue8-5.tr" using 1:5 t "Queue between Node 8 and Server" w lines linetype 4
```

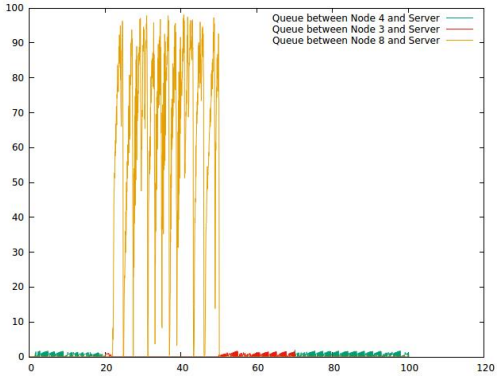


Figure 6 Queues Length

Figure.6 shows the utilization of the bottlenecks found in this network. As we can see, queue utilization between times (0 sec >> 22 sec) and (50 sec >> 100 sec) shows under-utilized performance. all bottlenecks can easily cater to the traffic moving through the bottlenecks. This scenario changed during the time (22 sec >> 50 sec) when link failures are observed in the network. During this period, all traffics (from data source node 0, Data source Node 9, and wireless network) pass through the bottleneck between Node 8 and Node 9. This scenario shows that the bottleneck suffers extremely high utilization that reaches 100%. In the following sections, we will study the impact of high utilization of the link on the network performance and the degradation that this scenario may cause.

Dropped packets at the Bottleneck

Plot "queue8-5.tr" using 1:8 t "Queue between Node 8 and Server" w lines linetype 4

Below, Figure.7 demonstrates the accumulation of packet drops simulated over the bottleneck between Node 8 and the server (Node 5). During the time when we have link failures, this bottleneck must accommodate all the traffics coming from different sources in the network. Because the bandwidth is limited, utilization of the bandwidth reaches 100%, as we can see in Figure.6, causing link congestion and over-utilization and packets dropping.

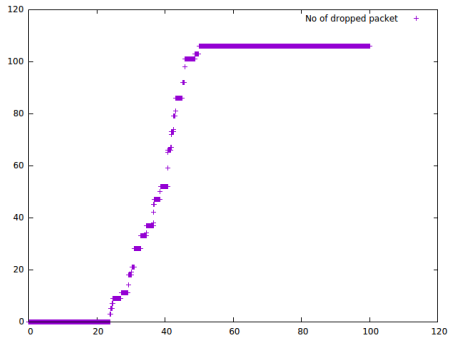


Figure 7 Dropped packets in Bottleneck (between node 8 and the server)

Throughput

Measuring the average throughput of TCP is an important performance measure. We are using a drop-tail queue, and we calculate the connection throughput using Perl script on the trace file .tr generated by the main TCL script. By calling the throughput.pl PERLfile, we obtain an output file that has the average TCP packet was received at the destination node (node 5- server) every period. Then, we display the output file using Gnuplot.

Server Throughput

To calculate the throughput of a TCP connection received by destination node “Server” every 0.5 seconds.

Perl throughput.pl wired-and-wireless.tr 5 0.5 > server_out

The output file “server_out” has two columns: Time and Throughput in (byte per second). To Plot the throughput of the server:

plot "server_out" using 1:2 title "Server Throughput" with lines linetype 3

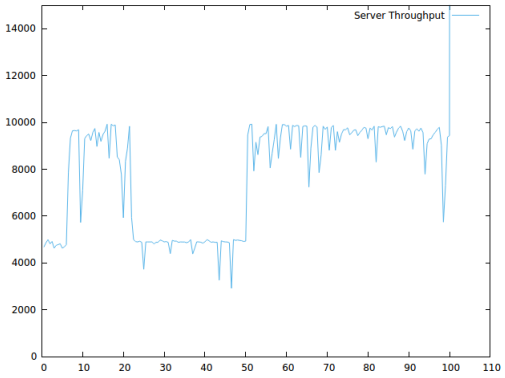


Figure 8 Throughput of TCP Connection of Node 5 (server)- DSDV

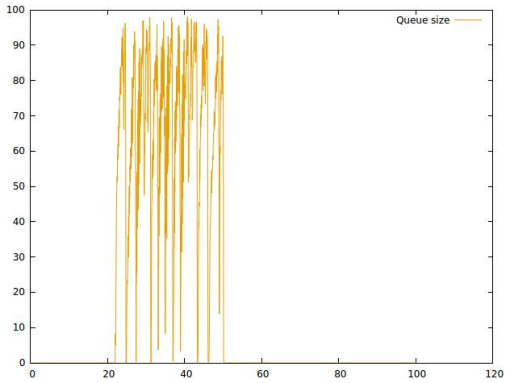


Figure 9 Queue size at the server -DSDV

At time 20 to 50 there is a decrease in the throughput, whereas the throughput was around 9000 and becomes low close to 5000 bytes between time 20 to 50. The reason is that at time 20 sec, and due to link failures simulated in the network, all traffics generated in the network has to go through a bottleneck (between Node 8 and the server Node 5). As we already saw in Figure.9, link utilization reaches 90% during this time, causing throughput degradation observed in Figure.8. In other words, an action is needed to either fix the links that fail regularly or expand the connection between node 8 and the server to avoid such degradation.

Base Station Throughput

To Calculate the throughput of a TCP connection received by destination node "Base station (10)" over 0.5 second.

```
perl throughput-wireless.pl wireless_traces.tr _10_ 0.5 >base_throughput
plot "base_throughput" using 1:2 title "Basestation Throughput " with lines linetype 3
```

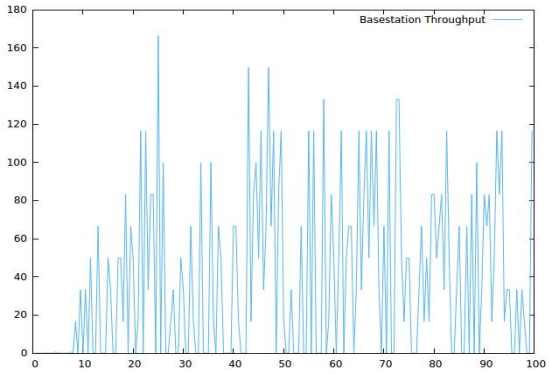


Figure 10 Base Station Throughput DSDV

The base station (BS) or Gateway is the access point that receives data transmissions from wireless nodes to send them to the wired network. We calculated the throughput at the base station, and the average throughput of data at the BS is around 120 kbps.

Automation

The main objective of using automation is to reduce the effort of manual execution. With performance analysis, we require changes in many files, and while changes this file, we may produce errors. Therefore, it reduces human errors and speed up the process and reduce human intervention. Also, it can test long simulations and can be used repeatedly by little or no modifications. To measures the performance accuracy, we run the simulations multiple times and get an average of all of them. Therefore, we run the simulation ten times. The parameters used in automation are No of packets sent and received, packet delivery ratio, dropping ratio and throughput, etc.

Automation Process

We generate an automation file using shell scripts; we are going to create different positions and mobility files for the wireless nodes (with zero mobility and various locations). Then, we will pass those files to the main TCL file. The .tcl file will produce trace and nam files. We finally, pass the trace output files to the analysis awk file to calculate the performance parameters and generate graphs.

Note: We use a random variable with seed=0 for start time for TCP sources.

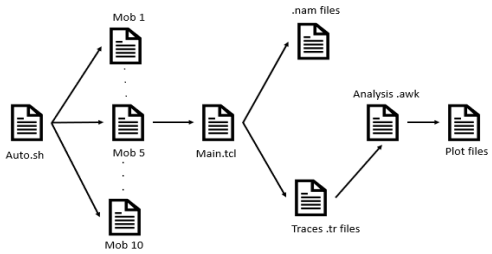


Figure 11 Automation Process

Automation Result

The result of the ten simulations using DSDV saved in Excel file, as seen below.

	A	B	C	D	E	F	G	H	I	J
1	Simulation_Number	No_of_Packets_Sent	No_of_Packets_Received	Packet_Delivery_Ratio	Packet_Dropping_Ratio	Throughput	Normalized_Routing_Overhead	Control_Overhead	Delay	Jitter
2	1	7533	5917	88.55	12.45	515.4	1.18	70	3.29	0.02
3	2	7701	5754	84.72	11.02	494.72	1.27	73	3.21	0.02
4	3	7363	6100	92.85	10.73	525.7	1.03	63	5.41	0.02
5	4	6983	6706	96.03	7.51	573.17	1.04	70	6.79	0.01
6	5	7165	6514	90.91	9.12	555.53	1	65	7.18	0.02
7	6	7615	6040	89.32	10.93	518.62	1.42	86	3.9	0.02
8	7	7551	5774	96.47	10.61	498.85	1.63	94	3.56	0.02
9	8	7755	5844	95.36	9.64	496.89	1.08	63	3.97	0.02
10	9	7528	6330	94.09	9.91	537.43	1.11	70	6.45	0.02
11	10	7500	6158	92.11	7.89	533.76	1.15	71	4.98	0.02
12	Average	7469.4	6113.7	92.041	9.95	525.007	1.191	72.5	4.874	0.019

Figure 12 Automation Result of 10 simulation

Automation Graphs for Wireless Network using DSDV

We use the Gnuplot script to plot the average performance parameters: No packet Sent and Received, Delay (second), Throughput (Kbps), Drop Ratio, Packet Delivery Ratio.

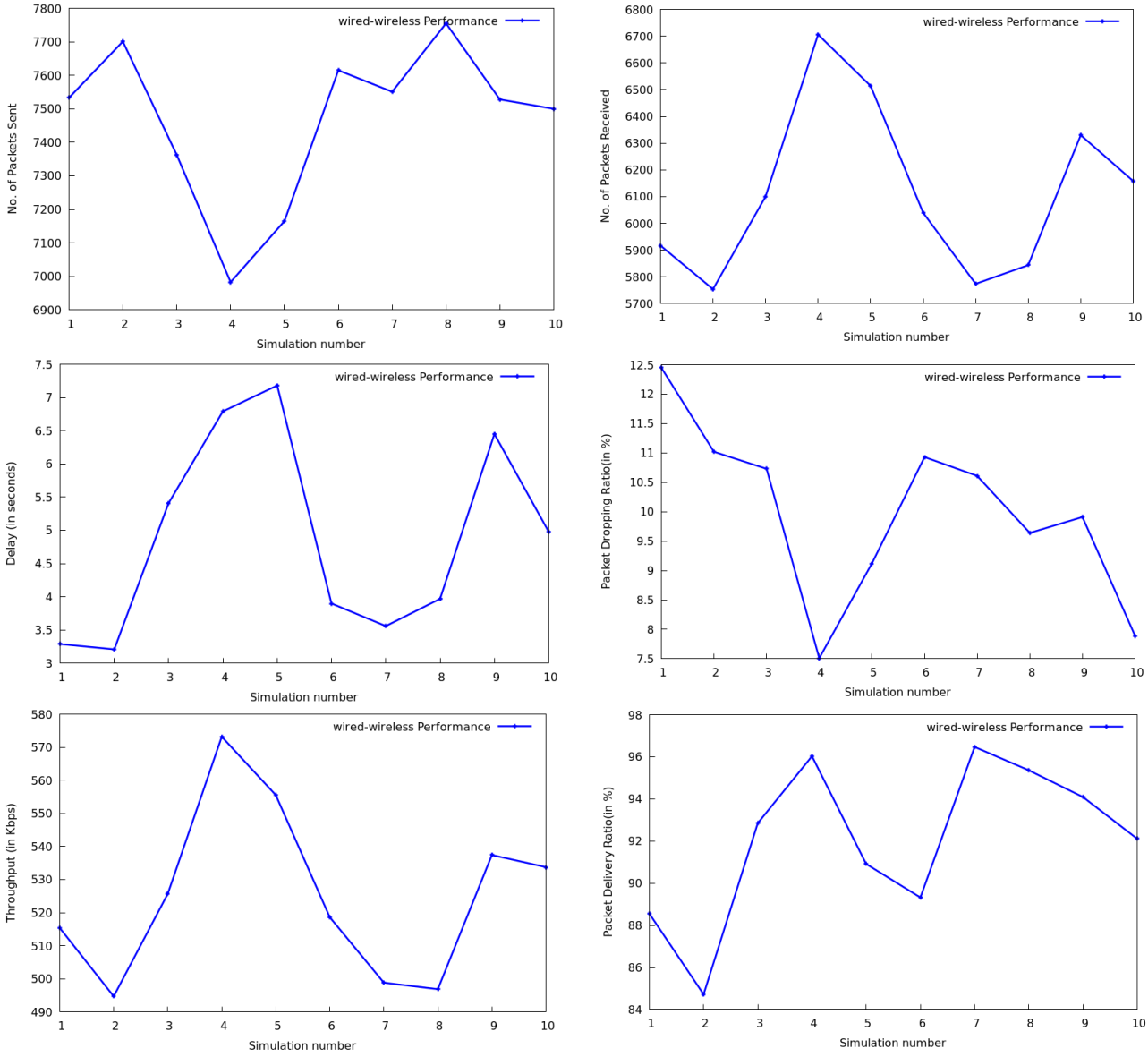


Figure 14 Automation Average Result of 10 Simulations

High-Level Observations:

- We found that the DSDV-based wireless sensor network performs better than AODV wireless protocol in wireless-cum-wired topology.
- It is essential to simulate possible link failures to monitor the performance of the network, including link over-utilization and congestion.
- It is crucial to prepare a comprehensive plan to simulate network performance in different scenarios. In this project, we studied different protocols, topologies, network scenarios, and we also simulated the impact of possible problems that may arise in real networks.

Conclusion and Future Work

In wired-cum-wireless network topology that we propose in this project, DSDV-based wireless network has shown better performance than AODV protocol in terms of throughput, packet drop ratio, packet delivery success rate, and delay. Based on the results we have found, we proceeded with DSDV protocol in a wireless network to study the proposed topology and analyze the results.

Moreover, the type of Queue has a noticeable effect on overall performance. For queue with Drop-Tail queue management, the queue size affects the performance of the TCP protocol. In conclusion, a large line causes high

queuing delay and weak loss recovery due to long retransmission timeouts. While small queue size cause in more packet drops. In general, Drop-Tail queue management can result in bursts of packet drops.
In future work, we are likely including further investigation on the benefits of using the RED buffer management scheme instead of a drop tail. As we noticed, the drop tail has a bias against bursty traffic. With many connections, congestion may occur and lead to undesirable results in the throughputs and packet loss. Therefore, we are going to consider this in our future work and use RED that detects congestion that lasts long and controls average queue size and delay. We are also going to study queuing in a wireless network. An additional topic for further work is that the effect of more routing protocol on the wired-cum-wireless network. We also plan to explore more simulation scenarios for AES and evaluate the performance over the ad-hoc network.

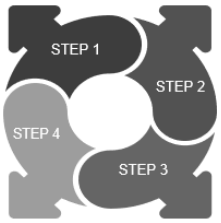
References

1. Ahlund, Christer, and Arkady Zaslavsky. "Integration of ad hoc network and IP network capabilities for mobile hosts." *10th International Conference on Telecommunications, 2003. ICT 2003.*. Vol. 1. IEEE, 2003.
2. Castellanos-Hernández, Wilder Eduardo, Juan Carlos Guerri Cebollada, and Mónica Edith Chacón Osorio. "A hybrid gateway discovery algorithm for supporting QoS communications in heterogeneous networks." Revista Facultad de Ingeniería Universidad de Antioquia 78 (2016): 80-88.
3. Chan, Hinghung Anthony. "System and method for distributed and integrated mobility support for mobile networks and mobile hosts." U.S. Patent No. 10,231,164. 12 Mar. 2019.
4. Deep Medhi.A Perspective on Network Restoration. accessed:https://pdfs.semanticscholar.org/488c/f9ccf30b88bdaa13d048120f5fe9d4024338.pdf
5. Gurtov, Andrei, and Sally Floyd. "Modeling wireless links for transport protocols." *ACM SIGCOMM Computer Communication Review* 34.2 (2004): 85-96.
6. Hemalatha, P. "Traffic Control in 5G Heterogenous Network." *2019 International Conference on Vision Towards Emerging Trends in Communication and Networking (ViTECoN)*. IEEE, 2019.
7. Issariyakul, Teerawat, and Ekram Hossain. "Introduction to network simulator 2 (NS2)." Introduction to network simulator NS2. Springer, Boston, MA, 2009. 1-18.
8. Rajankumar, Patel, Patel Nimisha, and Pariza Kamboj. "A comparative study and simulation of AODV MANET routing protocol in NS2 & NS3." 2014 International Conference on Computing for Sustainable Global Development (INDIACom). IEEE, 2014.

Project Milestones and Sechdule

Definition Phase Week 4,5 (Jan 20 - Feb 3)

- Define the problem
- Literature review for related work
- Simulation tool – installation and environment setup
- Determine network attributes:
  - Channel or link capacity
  - Bandwidth
  - Routing Protocol
  - Traffic Engineering



Modeling phase Week 6 - 7 (Feb 3 – Feb 24)

- Identification of the topology and network components.
- Build network simulation scenarios.
- Create network simulation scripts and codes.
- Identify performance parameters:
  - Packet Delivery and dropping Ratio,
  - Throughput
  - Time to restoration
  - Delay, loss, Routing Overhead, etc..
- Run simulation.
- Data collection (Simulation result and traces)

Result & Performance Analysis Phase Week 8 - 10 (Feb 24 - March 16)

- Create analysis script .awk, Perl
- Automation & Plotting

Finalize Report Week 11- 12 (Mar 16- Mar 23)

- Performance Analysis
- Conclusion
- Project documentation

Appendix

The complete project codes are available in the Github.

Main.tcl

```
global opt
set opt(chan)       Channel/WirelessChannel
set opt(prop)       Propagation/TwoRayGround
set opt(netif)      Phy/WirelessPhy
set opt(mac)        Mac/802_11
set opt(ifq)        Queue/DropTail/PriQueue
set opt(ll)         LL
set opt(ant)        Antenna/OmniAntenna
set opt(x)            800
set opt(y)            800
set opt(ifqlen)         50
set opt(tr)         wired-and-wireless.tr
set opt(namtr)      wired-and-wireless.nam
set opt(nn)           6
set opt(adhocRouting)   DSDV
set opt(cp)           ""
set opt(sc)           "scen-3-test"
set opt(stop)         100
set num_wired_nodes      10
set num_bs_nodes         1
# ####################
# Set number of sources
set NumbSrc 8
set ns_   [new Simulator]
#$ns_ use-newtrace
$ns_ rtproto Session
# set up for hierarchical routing
   $ns_ node-config -addressType hierarchical
  AddrParams set domain_num_ 2
  #lappend cluster_num 2 1 1
  lappend cluster_num 10 1
 AddrParams set cluster_num_ $cluster_num
  #lappend eilastlevel 1 1 4 1
  lappend eilastlevel 1 1 1 1 1 1 1 1 1 7
  AddrParams set nodes_num_ $eilastlevel
  set tracefd  [open $opt(tr) w]
  $ns_ trace-all $tracefd
  set namtracefd [open $opt(namtr) w]
  $ns_ namtrace-all $namtracefd
#set tf [open out.tr w]
set windowVsTime [open win w]
set param [open parameters w]
set trc  [open tracetcp.tr w]
set drop [open drop.tr w]
#set droptrc  [open droptrace.tr w]
set topo   [new Topography]
  $topo load_flatgrid $opt(x) $opt(y)
  # god needs to know the number of all wireless interfaces
  create-god [expr $opt(nn) + $num_bs_nodes]
  #create wired nodes
  set temp {0.0.0 0.1.0 0.2.0 0.3.0 0.4.0 0.5.0 0.6.0 0.7.0 0.8.0 0.9.0}
  for {set i 0} {$i < $num_wired_nodes} {incr i} {
      set W($i) [$ns_ node [lindex $temp $i]]  }
$ns_ at 0.0 "$W(5) color red"
$W(5) color "red"
$ns_ node-config -adhocRouting $opt(adhocRouting) \
                 -llType $opt(ll) \
                 -macType $opt(mac) \
                 -ifqType $opt(ifq) \
                 -ifqLen $opt(ifqlen) \
                 -antType $opt(ant) \
                 -propInstance [new $opt(prop)] \
                 -phyType $opt(netif) \
                 -channel [new $opt(chan)] \
                 -topoInstance $topo \
                 -wiredRouting ON \
                 -agentTrace ON \
```

```
            -routerTrace ON \
            -macTrace OFF
  set temp {1.0.0 1.0.1 1.0.2 1.0.3 1.0.4 1.0.5 1.0.6}
  set BS(0) [$ns_ node [lindex $temp 0]]
  $BS(0) random-motion 0
  $BS(0) set X_ 1.0
  $BS(0) set Y_ 2.0
  $BS(0) set Z_ 0.0
  $ns_ at 0.0 "$BS(0) color blue"
  $BS(0) color "blue"
   #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 [$BS(0) node-addr]]
  }
#ns_ at 0.0 "$node_(0) color blue"
#node_(0) color "blue"
#ns_ at 0.0 "$node_(1) color cyan"
#node_(1) color "cyan"
#ns_ at 1.0 "$node_(1) setdest 25.0 20.0 15.0"
#ns_ at 1.5 "$node_(0) setdest 20.0 18.0 1.0"
#ns_ at 0.5 "$node_(2) setdest 25.0 17.0 10.0"
  #create links between wired and BS nodes
#$ns_ duplex-link $W(1) $BS(1) 5Mb 2ms DropTail
$ns_ duplex-link $W(0) $W(1) 5Mb 0.5ms DropTail
#$ns_ queue-limit $W(0) $W(1) 10
#Set Queue Size of link (base-n1) to 150
$ns_ duplex-link $BS(0) $W(1) 5Mb 0.5ms DropTail
#$ns_ queue-limit $BS(0) $W(1) 10
$ns_ duplex-link $W(2) $W(1) 5Mb 0.5ms DropTail
#$ns_ queue-limit $W(2) $W(1) 10
$ns_ duplex-link $W(2) $W(3) 5Mb 0.5ms DropTail
#$ns_ queue-limit $W(2) $W(3) 10
$ns_ duplex-link $W(1) $W(4) 5Mb 0.5ms DropTail
#$ns_ queue-limit $W(1) $W(4) 10
$ns_ duplex-link $W(3) $W(5) 5Mb 0.5ms DropTail
$ns_ queue-limit $W(3) $W(5) 100
$ns_ duplex-link $W(4) $W(5) 5Mb 0.5ms DropTail
$ns_ queue-limit $W(4) $W(5) 100
$ns_ duplex-link $W(2) $W(6) 5Mb 0.5ms DropTail
#$ns_ queue-limit $W(2) $W(6) 100
$ns_ duplex-link $W(6) $W(7) 5Mb 0.5ms DropTail
#$ns_ queue-limit $W(6) $W(7) 100
$ns_ duplex-link $W(7) $W(8) 5Mb 0.5ms DropTail
#$ns_ queue-limit $W(7) $W(8) 100
$ns_ duplex-link $W(8) $W(5) 5Mb 0.01ms DropTail
#Set Queue Size of link (n8-n5) to 100
$ns_ queue-limit $W(8) $W(5) 100
$ns_ duplex-link $W(9) $W(8) 5Mb 0.5ms DropTail
#$ns_ queue-limit $W(9) $W(8) 100

$ns_ duplex-link-op $W(0) $W(1) orient right
$ns_ duplex-link-op $W(1) $BS(0) orient down-right
#$ns_ duplex-link-op $W(1) $BS(1) orient down-left
$ns_ duplex-link-op $W(1) $W(2) orient down
$ns_ duplex-link-op $W(1) $W(4) orient up-left
$ns_ duplex-link-op $W(2) $W(3) orient up-right
$ns_ duplex-link-op $W(2) $W(6) orient right
$ns_ duplex-link-op $W(3) $W(5) orient left-up
$ns_ duplex-link-op $W(4) $W(5) orient right-up
$ns_ duplex-link-op $W(6) $W(7) orient right-up
$ns_ duplex-link-op $W(7) $W(8) orient left-up
$ns_ duplex-link-op $W(8) $W(5) orient left
$ns_ duplex-link-op $W(8) $W(9) orient right
#Monitor the queue for link (n8-n5). (for NAM)
$ns_ duplex-link-op $W(8) $W(5) queuePos 0.5

#Monitor the queue for link (n4-n5). (for NAM)
$ns_ duplex-link-op $W(4) $W(5) queuePos 0.5

#Monitor the queue for link (n3-n5). (for NAM)
$ns_ duplex-link-op $W(3) $W(5) queuePos 0.5
```

```
#Monitor the queue for link (base-n1). (for NAM)
#$ns_ duplex-link-op $BS(0) $W(1) queuePos 0.5

$ns_ cost $W(1) $W(4) 1
$ns_ cost $W(2) $W(3) 1
$ns_ cost $W(3) $W(2) 1
$ns_ cost $W(5) $W(3) 1
$ns_ cost $W(3) $W(5) 1
$ns_ cost $W(4) $W(1) 1
$ns_ cost $W(2) $W(6) 1
$ns_ cost $W(6) $W(2) 1
$ns_ cost $W(7) $W(6) 1
$ns_ cost $W(6) $W(7) 1
$ns_ cost $W(5) $W(8) 1
$ns_ cost $W(8) $W(5) 1
$ns_ cost $W(9) $W(8) 1
$ns_ cost $W(8) $W(9) 1
$ns_ rtmodel-at 20 down $W(1) $W(4)
################# Houssam ############################
$ns_ rtmodel-at 22.0 down $W(2) $W(3)
################# Houssam ############################
$ns_ rtmodel-at 50.0 up $W(2) $W(3)
$ns_ rtmodel-at 70.0 up $W(1) $W(4)
# ######################################################################
# Create TCP Sources
for {set j 1} {$j<=$NumbSrc} { incr j } {
set tcp($j) [new Agent/TCP]
$tcp($j) set window_ 100   ;#maxVound on Window Size
$tcp($j) set packetSize_ 1000    ;#packet size used by sender
$tcp($j) set overhead_ 0   ;# !=0 adds random time between sends}
# ######################################################################
# Create TCP Destinations
for {set j 1} {$j<=$NumbSrc} { incr j } {
set sink($j) [new Agent/TCPSink]}
# ######################################################################
# Create a random generator for starting the ftp and for bottleneck link delays
set rng [new RNG]
$rng seed 2
# parameters for random variables for begenning of ftp connections
set RVstart [new RandomVariable/Uniform]
$RVstart set min_ 0
$RVstart set max_ 7
$RVstart use-rng $rng
#We define random starting times for each connection
for {set i 1} {$i<=$NumbSrc} { incr i } {
set startT($i)  [expr [$RVstart value]]
set dly($i) 1
puts $param "startT($i)  $startT($i) sec"}
# Color the packets
$tcp(1) set fid_ 1
$ns_ color 1 blue
$tcp(2) set fid_ 2
$ns_ color 2 yellow
$tcp(3) set fid_ 3
$ns_ color 3 cyan
$tcp(4) set fid_ 4
$ns_ color 4 green
$tcp(5) set fid_ 5
$ns_ color 5 orange
# setup TCP connections
##wired to wired
$ns_ attach-agent $W(0) $tcp(1)
$ns_ attach-agent $W(5) $sink(1)
$ns_ connect $tcp(1) $sink(1)
set ftp(1) [new Application/FTP]
$ftp(1) attach-agent $tcp(1)
#$ns_ at 1 "$ftp(1) start"
#wireless to wired
#from node 12 to server 5
$ns_ attach-agent $node_(0) $tcp(2)
$ns_ attach-agent $W(5) $sink(2)
$ns_ connect $tcp(2) $sink(2)
set ftp(2) [new Application/FTP]
$ftp(2) attach-agent $tcp(2)
#$ns_ at 1.5 "$ftp(2) start"
```

```
#wireless to wireless
#from node 13 to node 16
$ns_ attach-agent $node_(2) $tcp(3)
$ns_ attach-agent $node_(5) $sink(3)
$ns_ connect $tcp(3) $sink(3)
set ftp(3) [new Application/FTP]
$ftp(3) attach-agent $tcp(3)
#$ns_ at 3.0 "$ftp(3) start"
$ns_ attach-agent $W(5) $tcp(4)
$ns_ attach-agent $W(0) $sink(4)
$ns_ connect $tcp(4) $sink(4)
set ftp(4) [new Application/FTP]
$ftp(4) attach-agent $tcp(4)
#$ns_ at 1.2 "$ftp(4) start"
##wired to wired
$ns_ attach-agent $W(9) $tcp(5)
$ns_ attach-agent $W(5) $sink(5)
$ns_ connect $tcp(5) $sink(5)
set ftp(5) [new Application/FTP]
$ftp(5) attach-agent $tcp(5)
#$ns_ at 2.3 "$ftp(5) start"
#from node 14 to server 5
$ns_ attach-agent $node_(3) $tcp(6)
$ns_ attach-agent $W(5) $sink(6)
$ns_ connect $tcp(6) $sink(6)
set ftp(6) [new Application/FTP]
$ftp(6) attach-agent $tcp(6)
#$ns_ at 1.5 "$ftp(2) start"
#wireless to wired to wireless
#t0 node 11 from server 5
$ns_ attach-agent $W(5) $tcp(7)
$ns_ attach-agent $node_(0) $sink(7)
$ns_ connect $tcp(7) $sink(7)
set ftp(7) [new Application/FTP]
$ftp(7) attach-agent $tcp(7)
$ns_ attach-agent $node_(4) $tcp(8)
$ns_ attach-agent $node_(1) $sink(8)
$ns_ connect $tcp(8) $sink(8)
set ftp(8) [new Application/FTP]
$ftp(8) attach-agent $tcp(8)
#Schedule events for the FTP agents:
for {set i 1} {$i<=$NumbSrc} { incr i } {
$ns_ at $startT($i) "$ftp($i) start"
puts $startT($i)
$ns_ at $opt(stop) "$ftp($i) stop"
$ns_ at $opt(stop) "finish opt(tr)"}
#$ns_ trace-queue $W(1) $W(0) $trc
#$ns_ trace-queue $W(4) $W(5) $trc
#$ns_ trace-queue $W(8) $W(5) $trc
# Monitor avg queue length of link
set qfile [$ns_ monitor-queue $W(8) $W(5)  [open queue8-5.tr w] 0.05]
[$ns_ link $W(8) $W(5)] queue-sample-timeout;
set qfile [$ns_ monitor-queue $W(4) $W(5)  [open queue4-5.tr w] 0.05]
[$ns_ link $W(4) $W(5)] queue-sample-timeout;
set qfile [$ns_ monitor-queue $W(3) $W(5)  [open queue3-5.tr w] 0.05]
[$ns_ link $W(3) $W(5)] queue-sample-timeout;
#set qfile [$ns_ monitor-queue $BS(0) $W(1)  [open queue-base-5.tr w] 0.05]
#[$ns_ link $BS(0) $W(1)] queue-sample-timeout;
##################PLOTTING###################
proc plotWindow {tcpSource file k} {
global ns_ NumbSrc
set time 0.03
set now [$ns_ now]
set cwnd [$tcpSource set cwnd_]
if {$k == 1} {
   puts -nonewline $file "$now \t $cwnd \t"
  } else {
   if {$k < $NumbSrc } {
   puts -nonewline $file "$cwnd \t" } }
if { $k == $NumbSrc } {
   puts -nonewline $file "$cwnd \n" }
$ns_ at [expr $now+$time] "plotWindow $tcpSource $file $k" }
# The procedure will now be called for all tcp sources
for {set j 1} {$j<=$NumbSrc} { incr j } {
$ns_ at 0.1 "plotWindow $tcp($j) $windowVsTime $j" }
```

```
proc finish { file } {
        set f [open temp.rands w]
        puts $f "TitleText: Simulation"
        puts $f "Device: Postscript"
        exec rm -f temp.p temp.d
        exec touch temp.d temp.p
        exec awk {
                {
                        if (($1 == "+" || $1 == "-" ) && \
                           ($5 == "tcp" || $5 == "ack"))\
                                                print $2, $8 + ($11 % 90) * 0.01  }
        } wired-and-wireless.tr > temp.p
        exec awk {
                {
                        if ($1 == "d")
                                print $2, $8 + ($11 % 90) * 0.01  }
        } wired-and-wireless.tr > temp.d
        puts $f \"packets
        flush $f
        exec cat temp.p >@ $f
        flush $f
        # insert dummy data sets so we get X's for marks in data-set 4
        puts $f [format "\n\"skip-1\n0 1\n\n\"skip-2\n0 1\n\n"]
        puts $f \"drops
        flush $f
        exec head -n 1 temp.d >@ $f
        exec cat temp.d >@ $f
        close $f
        exec xgraph -bb -tk -nl -m -x time -y packet temp.rands }
 for {set i 0} {$i < $opt(nn)} {incr i} {
    $ns_ initial_node_pos $node_($i) 20  }
 for {set i } {$i < $opt(nn) } {incr i} {
    $ns_ at $opt(stop).0000010 "$node_($i) reset"; }
 $ns_ at $opt(stop).0000010 "$BS(0) reset";
 $ns_ at $opt(stop).1 "puts \"NS EXITING...\" ; $ns_ halt"
 puts "Starting Simulation..."
 $ns_ run
```

Wireless_Analysis.awk

```
BEGIN {
        send = 0;
        recv = 0;
        bytes = 0;
        drop=0;
        st = 0;
        ft = 0;
        rtr = 0;
        delay = 0;
        jitter=0;
        jitter_count=0;
        last_pkt_recv_time=0;
}
{
        if (( $1 == "s" || $1 == "f" )  && $4 == "RTR" && $7 == "message" )
        {
                rtr++;
        }
        if ( $1 == "s" && $4 == "AGT" && $7 == "tcp" )
        {
                if( send == 0 )
                {
                        st = $2;     }
                ft = $2;
                st_time[$6] = $2;
                send++;
        }
        if ( $1 == "r" && $4 == "AGT" && $7 == "tcp" )
        {
                if( recv == 0 )
                {
```

```
                        last_pkt_recv_time = $2;
                }
                else
                {
                        jitter += $2 - last_pkt_recv_time;
                        jitter_count++;
                        last_pkt_recv_time = $2
                }
                recv++;
                bytes += $8;
                ft_time[$6] = $2;
                delay += ft_time[$6] - st_time[$6] }
        if ($1 == "D" || $1 =="d" )
                drop++;
}
END {
        if(recv == 0) recv=1;
        #Printing results
        printf("No_of_Packets_Sent: \t\t%.f\n",send);
        printf("No_of_Packets_Received: \t%.f\n",recv);
        printf("No_of_Packets_Dropped: \t\t%.f\n",drop);
        printf("Packet_Delivery_Ratio: \t\t%.2f %%\n",recv/send*100);
        printf("Packet_Dropping_Ratio: \t\t%.2f %%\n",drop/send*100);
        printf("Throughput: \t\t\t%.2f Kbps\n",bytes*8/(ft-st)/1000);
        printf("Normalized_Routing_Overhead: \t%.2f %%\n",rtr/recv*100);
        printf("Control_Overhead: \t\t%d\n",rtr);
        printf("Delay: \t\t\t\t%.2f Seconds\n",delay/recv);
        printf("Jitter: \t\t\t%.2f\n",jitter/jitter_count);
}
```

Wired_analysis.awk

```
BEGIN {
        max_node = 2000;
        nSentPackets = 0.0 ;
        nReceivedPackets = 0.0 ;
        rTotalDelay = 0.0 ;
        max_pckt = 10000;
        idHighestPacket = 0;
        idLowestPacket = 100000;
        rStartTime = 10000.0;
        rEndTime = 0.0;
        nReceivedBytes = 0;
        rtr = 0;
        nDropPackets = 0.0;
        jitter=0;
        jitter_count=0;
        last_pkt_recv_time=0;
        temp = 0;
    frd_rtr=0;
        drop_rtr=0;
        rcv_rtr=0;
        snd_rtr=0;
        frd_agent=0;
        drop_agent=0;
        rcv_agent=0;
        snd_agent=0;
        rTime=0;
        jitter=0;
        jitter_count=0;
        for (i=0; i<max_node; i++) {
                node_thr[i] = 0;
        }
        total_retransmit = 0;
        for (i=0; i<max_pckt; i++) {
                retransmit[i] = 0;
        }
}
{
```

```
if ( $5 != "---" ) {
                        strEvent = $1;
                        rTime = $2;
                        from_node = $3;
                        to_node = $4;
                        pkt_type = $5;
                        pkt_size = $6;
                        flgStr = $7;  #---
                        flow_id = $8;
                        src_addr = $9; #0.0.0.0
                        dest_addr = $10;# 2.0.0.0
                        #seq_no = $11;
                        pkt_id = $11;
                        if(pkt_type == "tcp"){
                                if (pkt_id > idHighestPacket)
                                        idHighestPacket = pkt_id;
                                if (pkt_id < idLowestPacket)
                                        idLowestPacket = pkt_id;
                                if(rTime<rStartTime)
                                        rStartTime=rTime;
                                if(rTime>rEndTime)
                                        rEndTime=rTime;
                                if ( strEvent == "+") {

                                                nSentPackets += 1 ;
                                                rSentTime[ pkt_id ] = rTime ;
                                                send_flag[pkt_id] = 1;
                                        }
                                potential_dest = int(to_node)
                                dest = int(dest_addr)

                                if ( strEvent == "r" && pkt_size >= 512) {

                                        if( nReceivedPackets == 0 )
                                        {
                                                rTime = $2;}
                                        else
                                        {
                                                jitter += $2 - rTime;
                                                jitter_count++;
                                                rTime = $2}
                                        nReceivedPackets += 1 ;
                                        nReceivedBytes += pkt_size;
                                        potential_source = int(src_addr)
                                        rReceivedTime[ pkt_id ] = rTime ;
                                        rDelay[pkt_id] = rReceivedTime[ pkt_id] - rSentTime[ pkt_id ];
                                        rTotalDelay += rDelay[pkt_id];
                                        node_thr[potential_source] += pkt_size;
                                }
                                if(strEvent == "d" && pkt_size >= 512){
                                        #printf("Packet Dropped\n");

                                        nDropPackets += 1;
                                }
                        }
                        else
                        {
                                if(strEvent == "d")
                                #               nDropPackets += 1;
                        }}}

END {
        rTime = rEndTime - rStartTime ;
        rThroughput = (nReceivedBytes/(rTime))*(8/1000);
        rPacketDeliveryRatio = nReceivedPackets / nSentPackets * 100 ;
        rPacketDropRatio = nDropPackets / nSentPackets * 100;
        if ( nReceivedPackets != 0 ) {
                rAverageDelay = rTotalDelay / nReceivedPackets ;
        }
```

```
		for (i=0; i<max_pckt; i++) {
			total_retransmit += retransmit[i] ;
		}
printf("Simulation Time:	\t\t%15.2f\n",rTime)
printf("No_of_Packets_Sent: \t\t%15.2f\n",nSentPackets);
printf("No_of_Packets_Recieved:\t\t%15.2f\n",nReceivedPackets);
printf("No of Dropped Packet:\t\t%15.2f\n",nDropPackets);
printf("Packet_Delivery_Ratio: \t\t%15.2f %%\n",rPacketDeliveryRatio);
printf("Packet_Dropping_Ratio: \t\t%15.2f %%\n",rPacketDropRatio);
printf("Throughput:   \t\t\t%.2f [Mbps]\n",rThroughput/1024);
printf("Delay: \t\t\t%15.2f Seconds\n",rAverageDelay);
printf("Jitter: \t\t\t%.2f\n",jitter/jitter_count);}
```

Throughput_wired.pl

```
$infile=$ARGV[0];
$tonode=$ARGV[1];
$granularity=$ARGV[2];
$sum=0;
$clock=0;
     open (DATA,"<$infile")
       || die "Can't open $infile $!";
   while (<DATA>) {
          @x = split(' ');
#column 1 is time
if ($x[1]-$clock <= $granularity)
{
#checking if the event corresponds to a reception
if ($x[0] eq 'r')
{
#checking if the destination corresponds to arg 1
if ($x[3] eq $tonode)
{
#checking if the packet type is TCP
if ($x[4] eq 'tcp')
{
   $sum=$sum+$x[5];}}}}
else
{   $throughput=$sum/$granularity*(8/1000);
   print STDOUT "$x[1] $throughput\n";
   $clock=$clock+$granularity;
   $sum=0;}
}
  $throughput=$sum/$granularity;
   print STDOUT "$x[1] $throughput\n";
   $clock=$clock+$granularity;
   $sum=0;
   close DATA;
exit(0);
```

Throughput_wireless.pl

```
$infile=$ARGV[0];
$tonode=$ARGV[1];
$granularity=$ARGV[2];
$sum=0;
$clock=0;
     open (DATA,"<$infile")
       || die "Can't open $infile $!";

   while (<DATA>) {
          @x = split(' ');
#column 1 is time
if ($x[1]-$clock <= $granularity)
{
#checking if the event corresponds to a reception
if ($x[0] eq 'r')
{
```

```
#checking if the destination corresponds to arg 1
if ($x[2] eq $tonode)
{
#checking if the packet type is TCP
if ($x[6] eq 'tcp')
{
   $sum=$sum+$x[7];
}}}}
else
{   $throughput=$sum/$granularity*(8/1000);
   print STDOUT "$x[1] $throughput\n";
   $clock=$clock+$granularity;
   $sum=0;
}   }
  $throughput=$sum/$granularity;
   print STDOUT "$x[1] $throughput\n";
   $clock=$clock+$granularity;
   $sum=0;
   close DATA;
exit(0);
```

Automation.sh

```
#!/bin/bash
#Create required directories and set permissions
mkdir mobility_files
mkdir -p Output/nam
mkdir -p Output/trace
mkdir -p Output/Results
chmod 777 -R mobility_files
#Create required variables to store the parameter values.
send=0
recv=0
pdr=0
pdrop=0
tp=0
nro=0
co=0
dl=0
jitter=0
#Create required variables to store the average of parameter values.
avg_send=0
avg_recv=0
avg_pdr=0
avg_pdrop=0
avg_tp=0
avg_nro=0
avg_co=0
avg_dl=0
avg_jitter=0
count=10
i=0
#Create header of CSV file.
echo ""| awk 'BEGIN{printf "Simulation_Number,No_of_Packets_Sent,No_of_Packets_Received,Packet_Delivery_Ratio,Packet_Dropping_Ratio,Throughput,Normalized_Routing_Overhead,Control_Overhead,Delay,Jitter"}'>Output/Results/Final_Result.csv

#Main loop which runs 10 times.
while : ; do
        ((i++))
                #Creates 10 mobility files. Each of which is named as mob1, mob2, ... , mob10.
                ./setdest -v 2 -n 6 -m 1 -M 10 -t 100 -p 5 -x 800 -y 800 > mobility_files/mob$i
                #Call TCL script with mobility file as a parameter.
                ns main.tcl mobility_files/mob$i
                #Call analysis file and redirect the parameter values to file Result.txt
                printf "########## Simulation number $i ##########\n\n" >> Result.txt
                awk -f analysis-wireless.awk out_$i.tr >> Result.txt
                printf "\n\n" >> Result.txt
                #Call analysis file and redirect the parameter values to file tmp_result.
```

```
                awk -f analysis-wireless.awk out_$i.tr > tmp_result
                #Extract the parameter value from the file tmp_result and redirect it to the temporary file 1.txt.
                grep "No_of_Packets_Sent:" tmp_result |  awk 'BEGIN{}{print $2;}' > 1.txt
                send=$(<1.txt)
                grep "No_of_Packets_Received:" tmp_result |  awk 'BEGIN{}{print $2;}' > 1.txt
                recv=$(<1.txt)
                grep "Packet_Delivery_Ratio:" tmp_result |  awk 'BEGIN{}{print $2;}' > 1.txt
                pdr=$(<1.txt)
                grep "Packet_Dropping_Ratio:" tmp_result |  awk 'BEGIN{}{print $2;}' > 1.txt
                pdrop=$(<1.txt)
                grep "Throughput:" tmp_result |  awk 'BEGIN{}{print $2;}' > 1.txt
                tp=$(<1.txt)
                grep "Normalized_Routing_Overhead:" tmp_result |  awk 'BEGIN{}{print $2;}' > 1.txt
                nro=$(<1.txt)
                grep "Control_Overhead:" tmp_result |  awk 'BEGIN{}{print $2;}' > 1.txt
                co=$(<1.txt)
                grep "Delay:" tmp_result |  awk 'BEGIN{}{print $2;}' > 1.txt
                dl=$(<1.txt)
                grep "Jitter:" tmp_result |  awk 'BEGIN{}{print $2;}' > 1.txt
                jitter=$(<1.txt)
                #This condition handles if Packet Delivery Ratio is below 1. Sometimes if nodes are apart from each other, PDR is below 1.
                if [ 1 -eq "$(echo "${pdr} < 10" | bc)" ]
                then
                        ((i--))
                else
                        #Add all the values to average variable.
                        avg_send=$( bc <<< "$send + $avg_send ")
                        avg_recv=$( bc <<< "$recv + $avg_recv ")
                        avg_pdr=$( bc <<< "$pdr + $avg_pdr ")
                        avg_pdrop=$( bc <<< "$pdrop + $avg_pdrop ")
                        avg_tp=$( bc <<< "$tp + $avg_tp ")
                        avg_nro=$( bc <<< "$nro + $avg_nro ")
                        avg_co=$( bc <<< "$co + $avg_co ")
                        avg_dl=$( bc <<< "$dl + $avg_dl ")
                        avg_jitter=$( bc <<< "$jitter + $avg_jitter ")
                        #Print all the values to CSV file.
                        echo  "$i" | awk '{printf "\n%.5f", $1}' >> Output/Results/Final_Result.csv
                        echo  "$send" | awk '{printf ", %.5f", $1}' >> Output/Results/Final_Result.csv
                        echo  "$recv" | awk '{printf ", %.5f", $1}' >> Output/Results/Final_Result.csv
                        echo  "$pdr" | awk '{printf ", %.5f", $1}' >> Output/Results/Final_Result.csv
                        echo  "$pdrop" | awk '{printf ", %.5f", $1}' >> Output/Results/Final_Result.csv
                        echo  "$tp" | awk '{printf ", %.5f", $1}' >> Output/Results/Final_Result.csv
                        echo  "$nro" | awk '{printf ", %.5f", $1}' >> Output/Results/Final_Result.csv
                        echo  "$co" | awk '{printf ", %.5f", $1}' >> Output/Results/Final_Result.csv
                        echo  "$dl" | awk '{printf ", %.5f", $1}' >> Output/Results/Final_Result.csv
                        echo  "$jitter" | awk '{printf ", %.5f", $1}' >> Output/Results/Final_Result.csv
                fi

                if [ $i -eq $count ]
                then
                        break
                fi
        done
#Print all the average values to CSV file.
echo  "" | awk '{printf "\nAverage", $1}' >> Output/Results/Final_Result.csv
echo  "$avg_send $count" | awk '{printf ", %.5f", $1/$2}' >> Output/Results/Final_Result.csv
echo  "$avg_recv $count" | awk '{printf ", %.5f", $1/$2}' >> Output/Results/Final_Result.csv
echo  "$avg_pdr $count" | awk '{printf ", %.5f", $1/$2}' >> Output/Results/Final_Result.csv
echo  "$avg_pdrop $count" | awk '{printf ", %.5f", $1/$2}' >> Output/Results/Final_Result.csv
echo  "$avg_tp $count" | awk '{printf ", %.5f", $1/$2}' >> Output/Results/Final_Result.csv
echo  "$avg_nro $count" | awk '{printf ", %.5f", $1/$2}' >> Output/Results/Final_Result.csv
echo  "$avg_co $count" | awk '{printf ", %.5f", $1/$2}' >> Output/Results/Final_Result.csv
echo  "$avg_dl $count" | awk '{printf ", %.5f", $1/$2}' >> Output/Results/Final_Result.csv
echo  "$avg_jitter $count" | awk '{printf ", %.5f", $1/$2}' >> Output/Results/Final_Result.csv
#Remove all temporary files.
rm 1.txt
rm tmp_result
#Move all output files to their locations.
mv *.nam Output/nam
```

```
mv *.tr Output/trace
mv *.txt Output/Results
#Generate and print graphs to their own .png file.
#gnuplot gnuplot_script
echo "Completed successfully!"
```

Automation_plot

```
set style data linespoints
set datafile separator ","
set border 15
set xtics nomirror
set ytics nomirror
set key
set term pngcairo enhanced size 800,500
set style line 1 lt 2 lc rgb "blueviolet" lw 3
set linetype 1 lc rgb "blueviolet" lw 2.5  pt 1
set xlabel "Simulation number"

set ylabel "No. of Packets Sent"
set output "Output/Results/No_of_Packets_Sent.png"
plot 'Output/Results/Final_Result.csv' using 1:2  title "wired-wireless Performance" with linespoints;
set ylabel "No. of Packets Received"
set output "Output/Results/No_of_Packets_Received.png"
plot 'Output/Results/Final_Result.csv' using 1:3  title "wired-wireless Performance" with linespoints;
set ylabel "Packet Delivery Ratio(in %)"
set output "Output/Results/Packet_Delivery_Ratio.png"
plot 'Output/Results/Final_Result.csv' using 1:4  title "wired-wireless Performance" with linespoints;
set ylabel "Packet Dropping Ratio(in %)"
set output "Output/Results/Packet_Dropping_Ratio.png"
plot 'Output/Results/Final_Result.csv' using 1:5  title "wired-wireless Performance" with linespoints;
set ylabel "Throughput (in Kbps)"
set output "Output/Results/Throughput.png"
plot 'Output/Results/Final_Result.csv' using 1:6  title "wired-wireless Performance" with linespoints;
set ylabel "Normalized Routing Overhead(in %)"
set output "Output/Results/Normalized_Routing_Overhead.png"
plot 'Output/Results/Final_Result.csv' using 1:7  title "wired-wireless Performance" with linespoints;
set ylabel "Control Overhead"
set output "Output/Results/Control_Overhead.png"
plot 'Output/Results/Final_Result.csv' using 1:8  title "wired-wireless Performance" with linespoints;
set ylabel "Delay (in seconds)"
set output "Output/Results/Delay.png"
plot 'Output/Results/Final_Result.csv' using 1:9  title "wired-wireless Performance" with linespoints;
set ylabel "Jitter (in seconds)"
set output "Output/Results/Jitter.png"
plot 'Output/Results/Final_Result.csv' using 1:10  title "wired-wireless Performance" with linespoints;
```