Wireless Simulation

CA2

Mohammad Araghi 810198436

Mohammad Saadati 810198410

Introduction

In this computer assignment, we simulate the wireless connection of 9 nodes, including two sender nodes (A and D) and two receiver nodes (L and H).

We use 802.11 protocol which will decide some attributes of our WLAN implementation and ways for MAC (Media Access Control) with some protocols for physical layers.

RTS / CTS

RTS / CTS Flow Control is a flow control mechanism that is part of the RS232 standard. It makes use of two further pins on the RS232 connector, RTS (Request to Send) and CTS (Clear to Send). These two lines allow the receiver and the transmitter to alert each other to their state.

Using RTS / CTS will increase reliability of our packet transferring and stops collision (when two nodes want to send packets to a same node in same time).

Acknowledgment will be sent by transferrin ACKs.

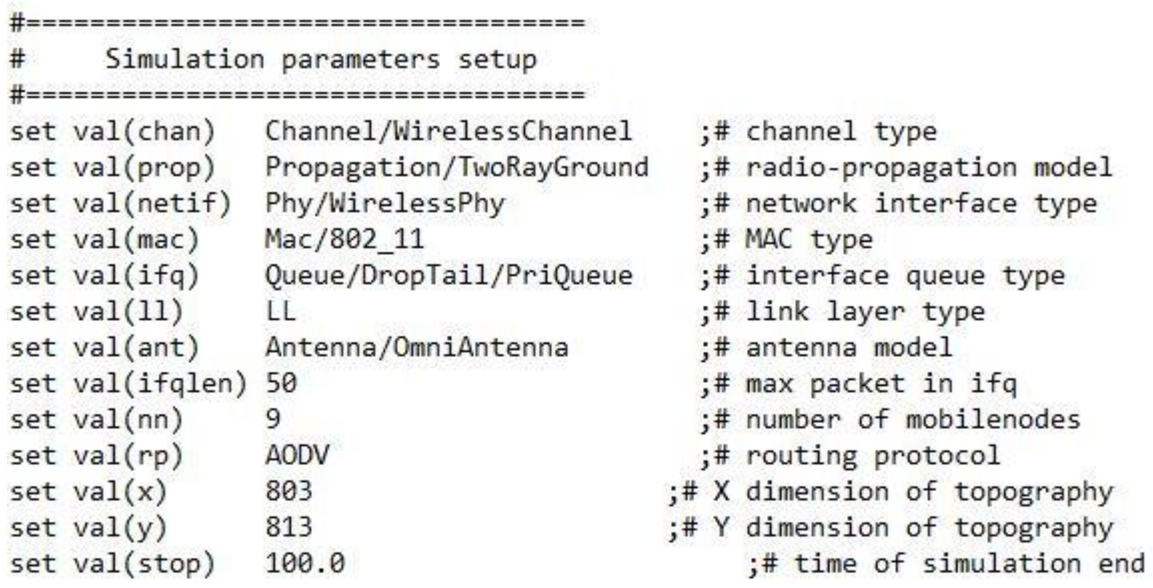
NS2

NS2 is a powerful tool for simulating networks with the ability to support various network protocols. This software provides the possibility to simulate TCP/UDP layers in wired and wireless mods like LAN, WAN, Ad Hoc, Satellite, etc.

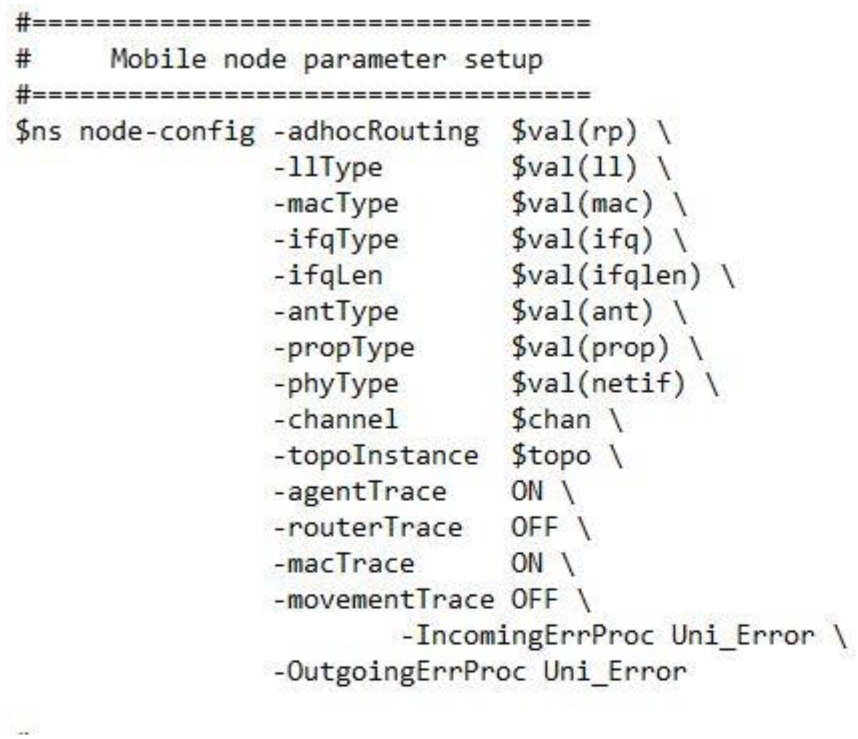
Also various protocols in different layers like TCP, UDP, FTP, DSR, AODV, CBR and … are implemented in NS2.

Topology

For our topology, we used configurations as below:



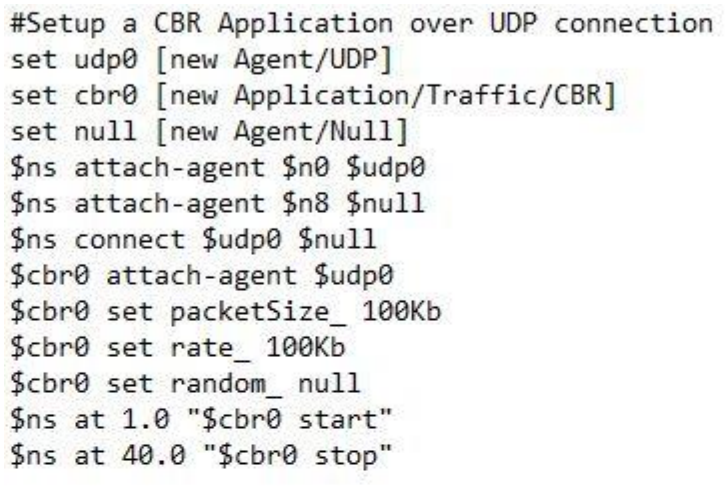
```
#===================================
#     Simulation parameters setup
#===================================
set val(chan)   Channel/WirelessChannel    ;# channel type
set val(prop)   Propagation/TwoRayGround   ;# radio-propagation model
set val(netif)  Phy/WirelessPhy            ;# network interface type
set val(mac)    Mac/802_11                 ;# MAC type
set val(ifq)    Queue/DropTail/PriQueue    ;# interface queue type
set val(ll)     LL                         ;# link layer type
set val(ant)    Antenna/OmniAntenna        ;# antenna model
set val(ifqlen) 50                         ;# max packet in ifq
set val(nn)     9                          ;# number of mobilenodes
set val(rp)     AODV                       ;# routing protocol
set val(x)      803                      ;# X dimension of topography
set val(y)      813                      ;# Y dimension of topography
set val(stop)   100.0                         ;# time of simulation end
```



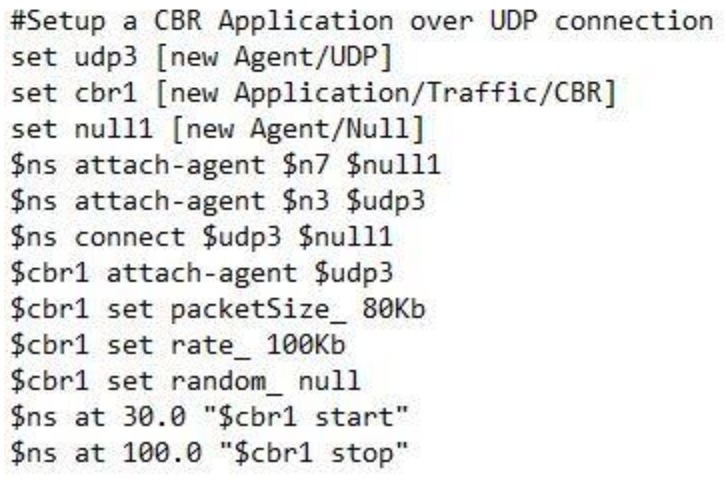
```
#===================================
#     Mobile node parameter setup
#===================================
$ns node-config -adhocRouting  $val(rp) \
                -llType        $val(ll) \
                -macType       $val(mac) \
                -ifqType       $val(ifq) \
                -ifqLen        $val(ifqlen) \
                -antType       $val(ant) \
                -propType      $val(prop) \
                -phyType       $val(netif) \
                -channel       $chan \
                -topoInstance  $topo \
                -agentTrace    ON \
                -routerTrace   OFF \
                -macTrace      ON \
                -movementTrace OFF \
                        -IncomingErrProc Uni_Error \
                -OutgoingErrProc Uni_Error
```

Also an error rate and simulation bandwidth, which are given by our python script as the values of assignment.

For our simulation, we used UDP and CBR (constant bit rate) which will send packets by a constant rate with parameters as below:



```
#Setup a CBR Application over UDP connection
set udp0 [new Agent/UDP]
set cbr0 [new Application/Traffic/CBR]
set null [new Agent/Null]
$ns attach-agent $n0 $udp0
$ns attach-agent $n8 $null
$ns connect $udp0 $null
$cbr0 attach-agent $udp0
$cbr0 set packetSize_ 100Kb
$cbr0 set rate_ 100Kb
$cbr0 set random_ null
$ns at 1.0 "$cbr0 start"
$ns at 40.0 "$cbr0 stop"
```



```
#Setup a CBR Application over UDP connection
set udp3 [new Agent/UDP]
set cbr1 [new Application/Traffic/CBR]
set null1 [new Agent/Null]
$ns attach-agent $n7 $null1
$ns attach-agent $n3 $udp3
$ns connect $udp3 $null1
$cbr1 attach-agent $udp3
$cbr1 set packetSize_ 80Kb
$cbr1 set rate_ 100Kb
$cbr1 set random_ null
$ns at 30.0 "$cbr1 start"
$ns at 100.0 "$cbr1 stop"
```

Neighborhoods are shown by nodes' distances.

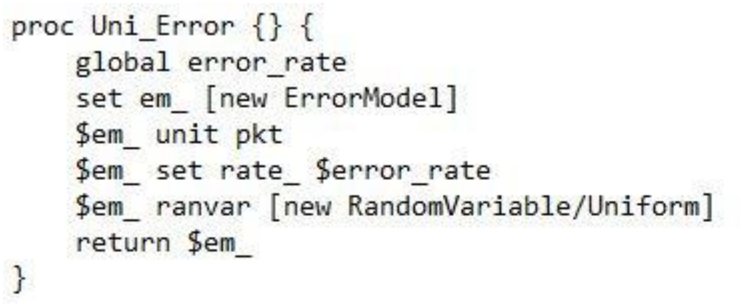
- A node starts sending data at t = 1s and stops at t = 40s.
- D node starts sending data at t = 30s and stops at t = 100s.

By simulating this TCL file, we will get a trace file which we will analysis soon.

Error rate

For generating error rate with different values between 0.000001 and 0.00001,

We defined Uni_Error proc as below:

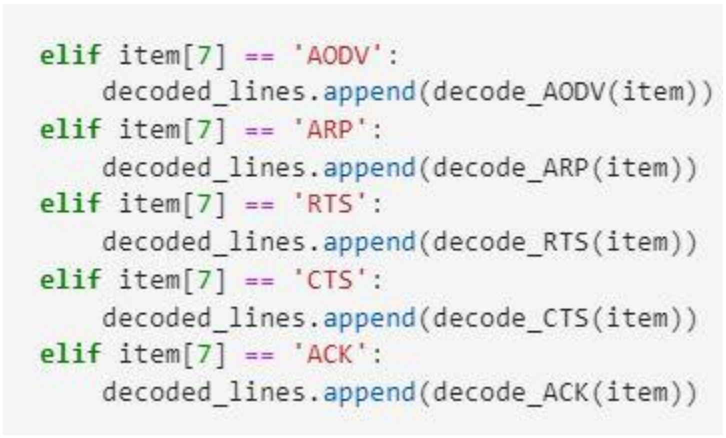


```
proc Uni_Error {} {
    global error_rate
    set em_ [new ErrorModel]
    $em_ unit pkt
    $em_ set rate_ $error_rate
    $em_ ranvar [new RandomVariable/Uniform]
    return $em_
}
```

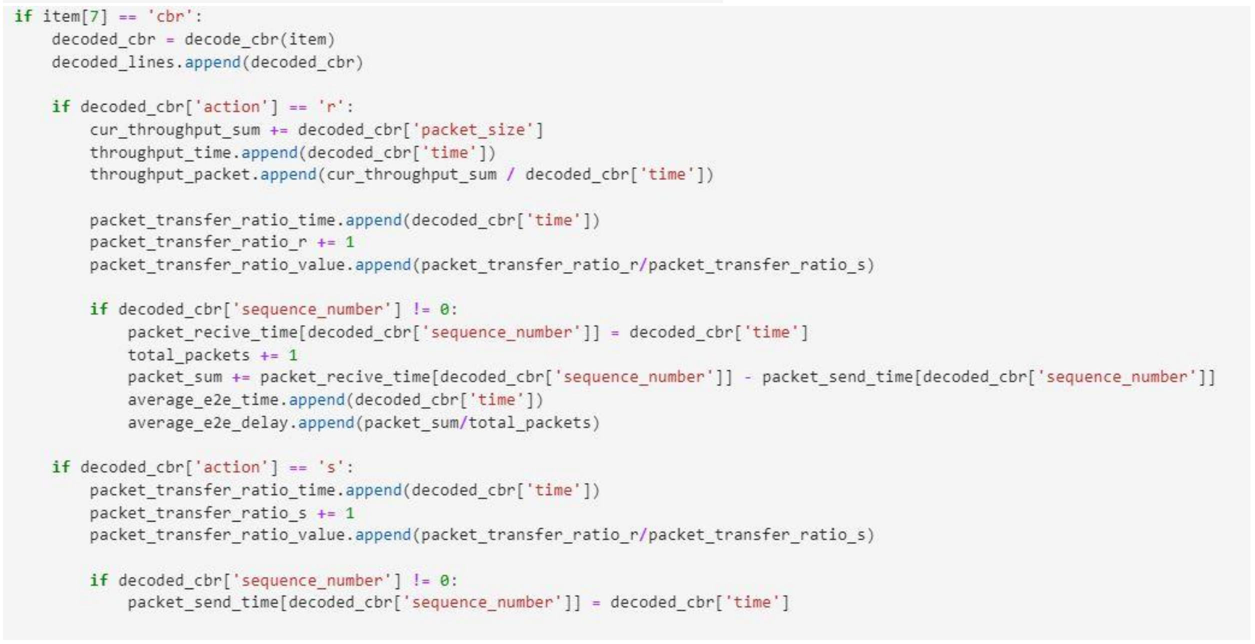
And changed the values by our python script.

Decoding

For decoding and trace file and finding useful information, we separated our data based on 8th argument in line:

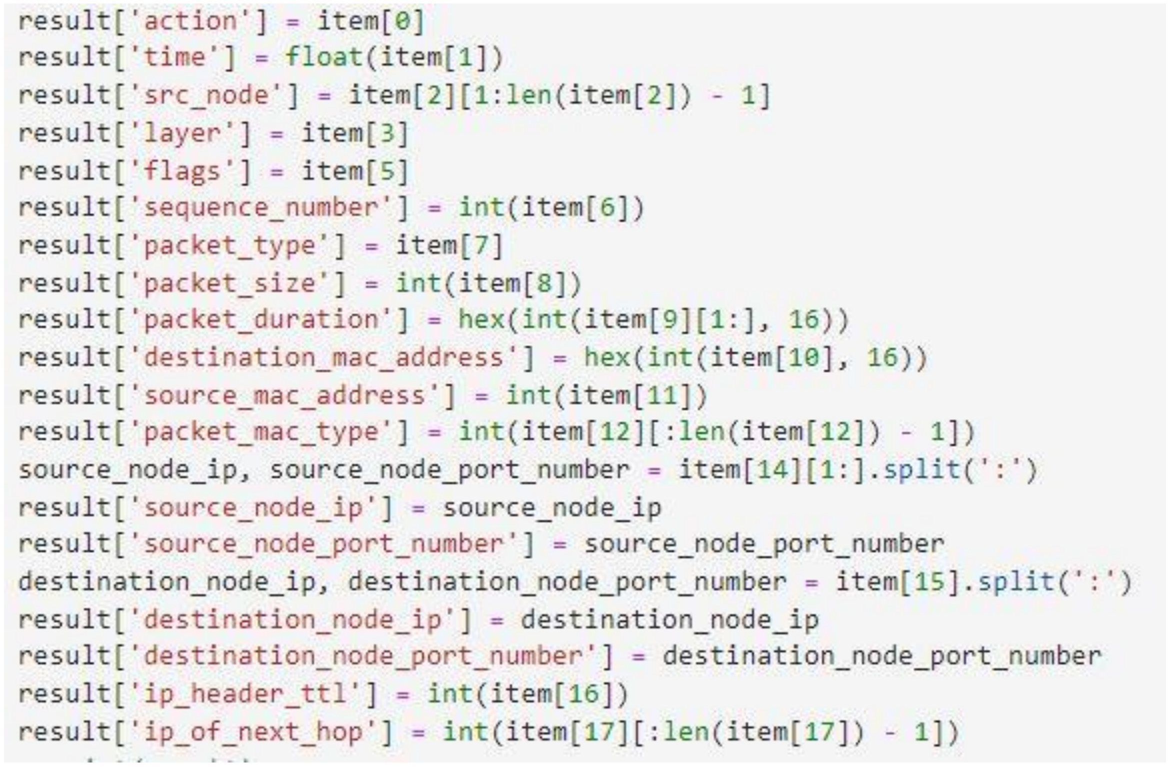


```
elif item[7] == 'AODV':
    decoded_lines.append(decode_AODV(item))
elif item[7] == 'ARP':
    decoded_lines.append(decode_ARP(item))
elif item[7] == 'RTS':
    decoded_lines.append(decode_RTS(item))
elif item[7] == 'CTS':
    decoded_lines.append(decode_CTS(item))
elif item[7] == 'ACK':
    decoded_lines.append(decode_ACK(item))
```



```
if item[7] == 'cbr':
    decoded_cbr = decode_cbr(item)
    decoded_lines.append(decoded_cbr)

    if decoded_cbr['action'] == 'r':
        cur_throughput_sum += decoded_cbr['packet_size']
        throughput_time.append(decoded_cbr['time'])
        throughput_packet.append(cur_throughput_sum / decoded_cbr['time'])

        packet_transfer_ratio_time.append(decoded_cbr['time'])
        packet_transfer_ratio_r += 1
        packet_transfer_ratio_value.append(packet_transfer_ratio_r/packet_transfer_ratio_s)

        if decoded_cbr['sequence_number'] != 0:
            packet_recive_time[decoded_cbr['sequence_number']] = decoded_cbr['time']
            total_packets += 1
            packet_sum += packet_recive_time[decoded_cbr['sequence_number']] - packet_send_time[decoded_cbr['sequence_number']]
            average_e2e_time.append(decoded_cbr['time'])
            average_e2e_delay.append(packet_sum/total_packets)

    if decoded_cbr['action'] == 's':
        packet_transfer_ratio_time.append(decoded_cbr['time'])
        packet_transfer_ratio_s += 1
        packet_transfer_ratio_value.append(packet_transfer_ratio_r/packet_transfer_ratio_s)

        if decoded_cbr['sequence_number'] != 0:
            packet_send_time[decoded_cbr['sequence_number']] = decoded_cbr['time']
```

And for our arguments respectively in CBR (which is used the most for our analyzing:

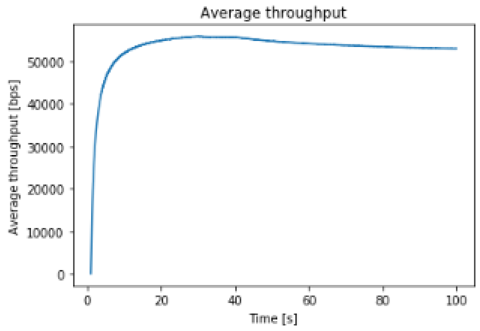


```
result['action'] = item[0]
result['time'] = float(item[1])
result['src_node'] = item[2][1:len(item[2]) - 1]
result['layer'] = item[3]
result['flags'] = item[5]
result['sequence_number'] = int(item[6])
result['packet_type'] = item[7]
result['packet_size'] = int(item[8])
result['packet_duration'] = hex(int(item[9][1:], 16))
result['destination_mac_address'] = hex(int(item[10], 16))
result['source_mac_address'] = int(item[11])
result['packet_mac_type'] = int(item[12][:len(item[12]) - 1])
source_node_ip, source_node_port_number = item[14][1:].split(':')
result['source_node_ip'] = source_node_ip
result['source_node_port_number'] = source_node_port_number
destination_node_ip, destination_node_port_number = item[15].split(':')
result['destination_node_ip'] = destination_node_ip
result['destination_node_port_number'] = destination_node_port_number
result['ip_header_ttl'] = int(item[16])
result['ip_of_next_hop'] = int(item[17][:len(item[17]) - 1])
```

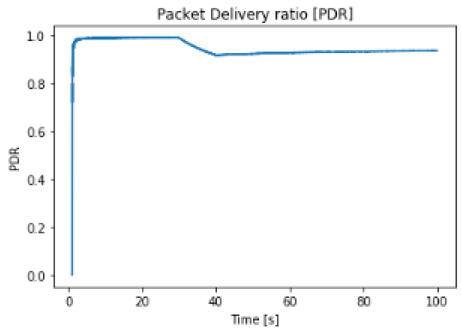
Results

For this 6 different example values of bandwidth and error rate (30 original values):

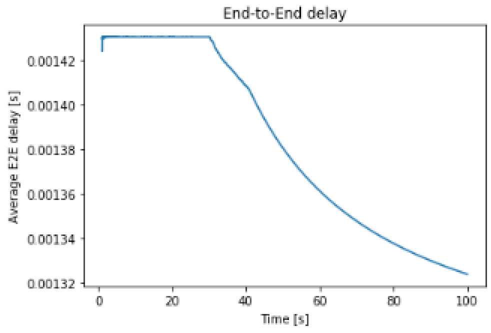
```
Bandwidth = 1.5
Error rate = 0.000001


Throughput = 52906.92614014437
```



```
Packet Transfer ratio = 0.9356244223550937
```

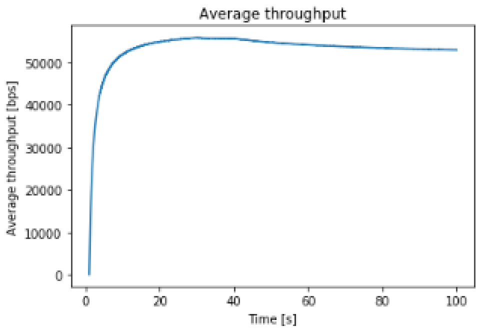


```
Average End-to-End delay = 0.0013240680695161067
```

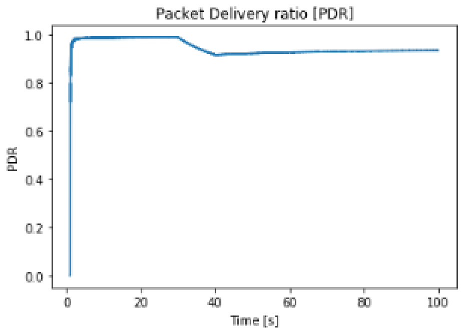


```
------------------------------------------------
```

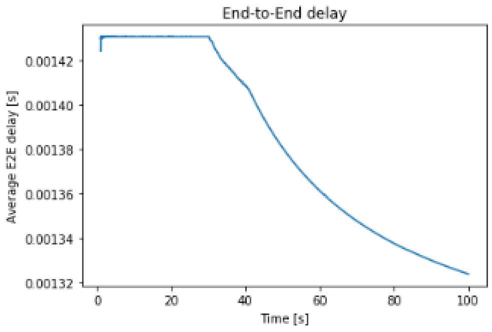
```
------------------------------------------------
Bandwidth = 1.5
Error rate = 0.000005


Throughput = 52900.22867319458
```



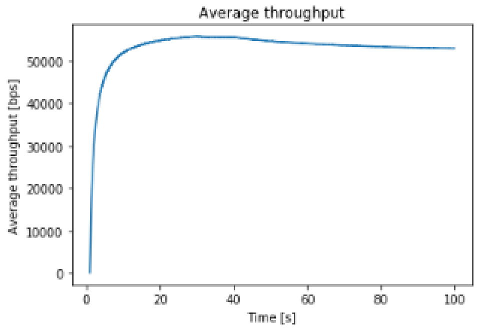
```
Packet Transfer ratio = 0.9355641905911646
```



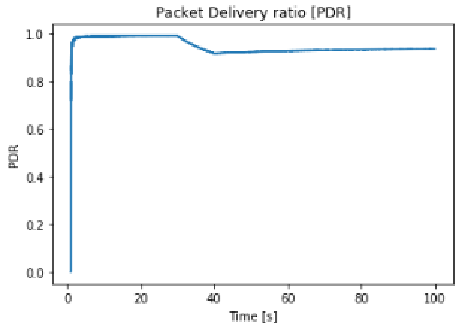
```
Average End-to-End delay = 0.0013240742620478176
```



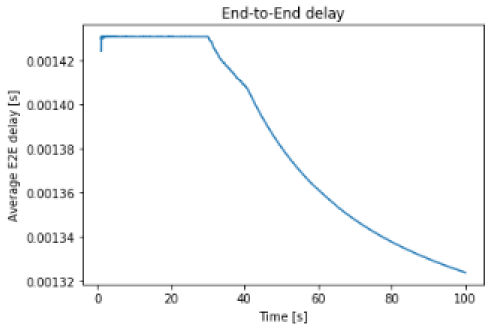
```
Bandwidth = 55.0
Error rate = 0.000001


Throughput = 52906.92614014437
```



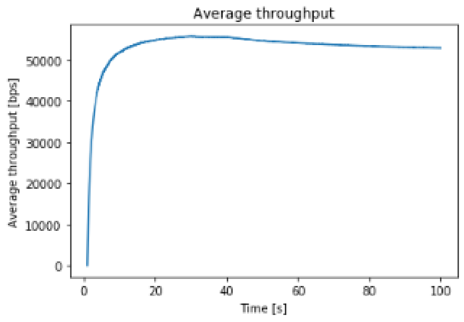
```
Packet Transfer ratio = 0.9356244223550937
```



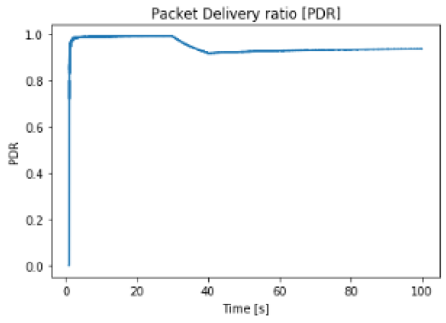
```
Average End-to-End delay = 0.0013240680695161067
```



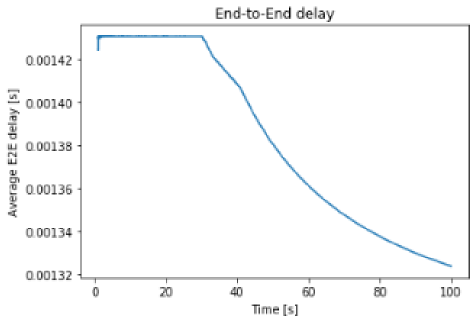
```
----------------------------------------------
Bandwidth = 55.0
Error rate = 0.000005


Throughput = 52900.22867319458
```



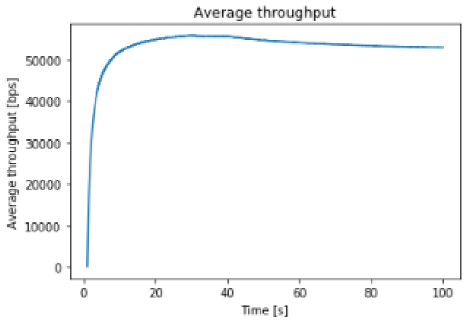
```
Packet Transfer ratio = 0.9355641905911646
```



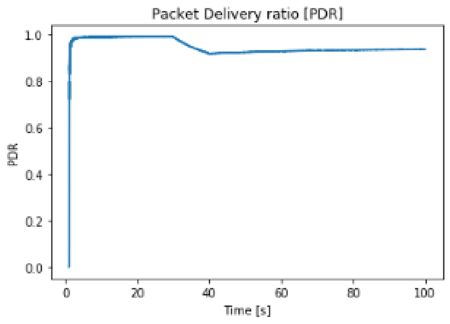
```
Average End-to-End delay = 0.0013240742620478176
```



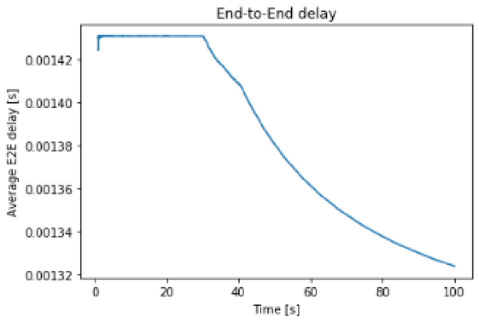
```
------------------------------------------------
Bandwidth = 155.0
Error rate = 0.000001


Throughput = 52906.92614014437
```



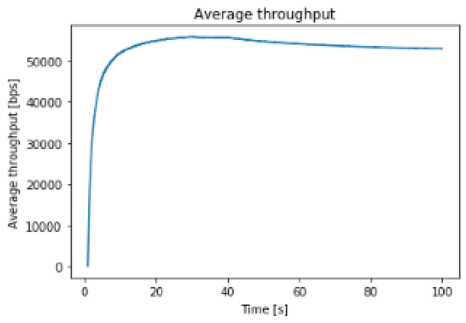
```
Packet Transfer ratio = 0.9356244223550937
```



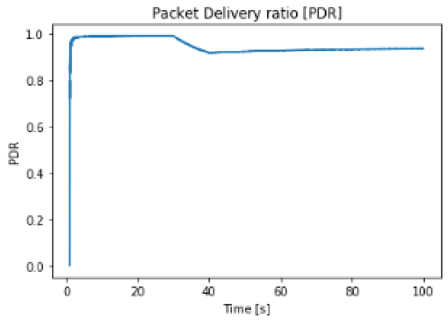
```
Average End-to-End delay = 0.0013240680695161067
```



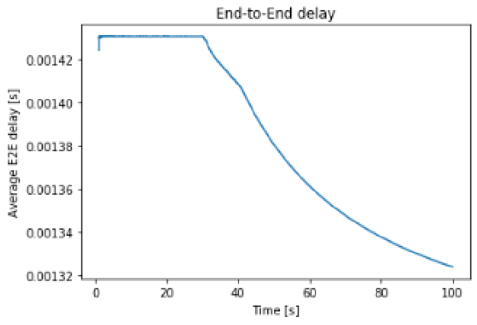
```
------------------------------------------------
Bandwidth = 155.0
Error rate = 0.000005


Throughput = 52900.22867319458
```



```
Packet Transfer ratio = 0.9355641905911646
```



```
Average End-to-End delay = 0.0013240742620478176
```



We will see that:

1. Since we are not using our full bandwidth limit, there is no change in throughput, PDR and E2E delay amount based on the bandwidth change, But if we did, increasing bandwidth would have increased average throughput, and might have decreased E2E delay.
2. By increasing our error value, average throughput will decrease, PDR will decrease since many sent packets will be dropped and will not be delivered, and E2E will increase. But since our error value is really small in this project, not much change can be seen.