EE444 - HW3 OMNeT++

# 2. Data-Link Layer – Ethernet

## 2.1.

### 2.1.a. Create a .ned file for your network. Define a parameter for node count and arrays for EthernetHost and WireJunction. WireJunction will be used for creating a bus. Connect WireJunctions serially using an Eth10M link. Then connect each EthernetHost to a WireJunction.

A screen shot of a computer code

Description automatically generated with low confidence

Figure .ned file for Data-Link Layer - Ethernet

### 2.1.b. Create a .ini file to configure your simulation. Designate destination addresses, request length and response length. Set your send interval as exponential.

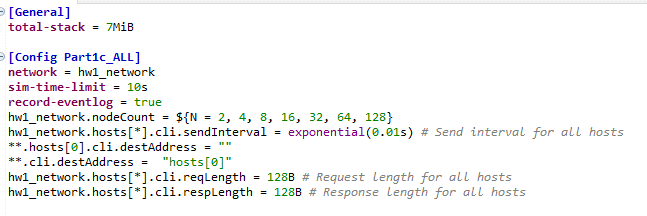


Figure .ini file for Data-Link Layer - Ethernet

## 2.1.c. Simulate your network for different node counts of 2, 4, 8, 16, 32, 64 and 128. Plot the node count vs channel efficiency graph. Channel efficiency is defined as how much payload is sent through the channel per second per physical rate (10Mbps for our case). Comment on your results. Required parameters for your calculations are generated automatically in the Results folder.

I run the simulations for 10 seconds per node count and initialization metrics can be found in Appendix ([Config Part1c\_ALL], network = hw1\_network).

With the low number of nodes, the efficiency is very low as we are not utilizing the line with our sending rate and packets. However, it is seen that after 32 nodes, our efficiency drops. This can be explained by having too much packets in the network and collusion happens, which decreases our network performance.

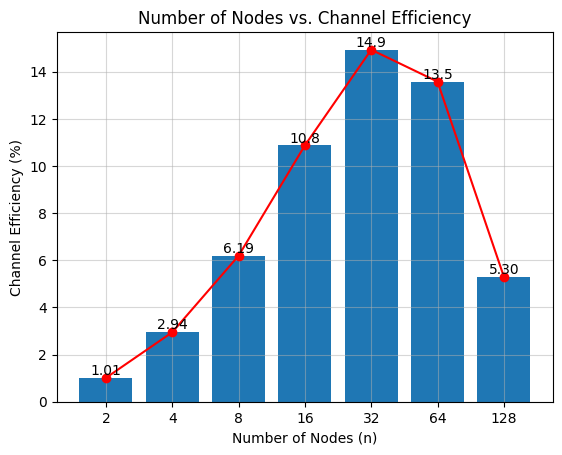


Figure Number of Nodes vs. Channel Efficiency

## 2.1.d. Using 16 nodes, simulate for different frame lengths of 64, 128, 256, 512 and 1024. Plot the channel efficiency vs frame length graph. Comment on your results.

Initialization metrics can be found in Appendix ([Config Part1d\_ALL], network = hw1\_network).

As we increase the frame size, it is observed that our channel efficiency increases with 16 nodes. Since the host already uses the connection between itself and the server, where the packet exists, it can have larger packet size/frame length as the capacity allows.

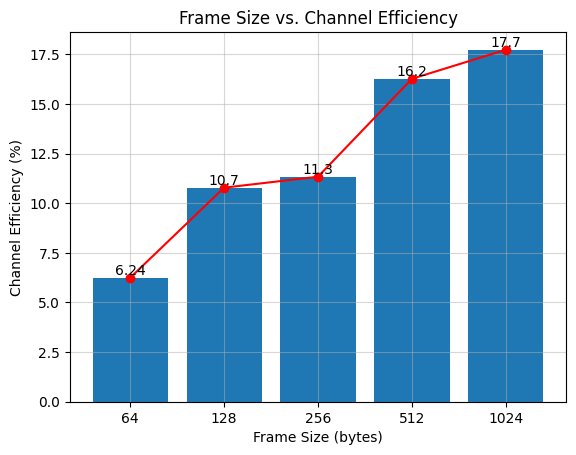


Figure Frame Size vs. Channel Efficiency

## 2.1.e. Using 16 nodes, divide the exponential parameter for your send interval by 2 and simulate. Compare the efficiency with your previous simulation with 32 nodes. Comment on your results.

Initialization metrics can be found in Appendix ([Config Part1e\_ALL], network = hw1\_network).

As we decrease the send interval, we increase packet sending frequency. It is observed that, channel efficiency at 16 nodes with 0.005s interval is almost equal to the channel efficiency at 32 nodes with 0.01s interval. The 16 nodes channel efficiency increases to 15.1% from 10%8, which is very close to 32 nodes 14.9% efficiency. This is due to the approximately same data sent per second with this configuration.

A picture containing text, diagram, line, plot

Description automatically generated

Figure Number of Nodes vs. Channel Efficiency

# 2.2. Add a switch to divide the bus into two equal-size collision domains and repeat the same simulations in Question 1. Comment on the different results.

## 2.2.c.

Initialization metrics can be found in Appendix ([Config Part2c\_ALL], network = hw1\_network2).

It is observed that the results did not change until node 32, as the collusions not create much problem. After node 32, the efficiency increases significantly, since switch effectively decreases the network congestion, and each node is able to utilize the bandwidth better.

A graph with a red line

Description automatically generated with low confidence

Figure Number of Nodes vs. Channel Efficiency

## 2.2.d.

Initialization metrics can be found in Appendix ([Config Part2d\_ALL], network = hw1\_network2).

It is observed that the channel efficiency increases until frames size 512 bytes, but then decreases. For 1024 frame size, it starts decreasing since collisions start to happen with network congestion. From this test, we observe the most efficient frame size is 512 bytes for this network.

A graph with a red line

Description automatically generated with low confidence

Figure Frame Size vs. Channel Efficiency

# 3. Transport Layer – TCP

## 3.1. Create one standard host as a client and another standard host as a server. Create one router with a queue size of 10 packets. Connect the client to the router with a 100kbps line. Connect the server to the router with a 1Mbps line. Use ThruputMeteringChannel for connections with bandwidth measuring mode. For the client, use TcpSessionApp with a large byte count to simulate a continuous stream. Use TcpSinkApp for the server.

A screenshot of a computer program

Description automatically generated with medium confidence

Figure .ned file for Transport Layer - TCP

A screenshot of a computer program

Description automatically generated with medium confidence

Figure .ini file for Transport Layer - TCP

## 3.1.a. What is the mean bandwidth that is used on the connection between the client and the router? Is this the expected result?

Initialization metrics can be found in Appendix ([Config TCP1], network = hw1\_network\_tcp).

From the simulation result, we can see that our mean bandwidth is around 99.5kbps for sending the data, and 8.03kbps for returning data, in total 108 kbps. It is more than our limit, 100kbps, but this is due to the starting time window for reaching the threshold. In general and almost for the whole line, we can say the bandwidth is 100kbps and respects the limit.

A picture containing text, software, computer icon, multimedia software

Description automatically generated

Figure Simulation Results for 3.1.

## 3.1.b. Plot data rate vs time graph. Describe what you observe. Comment on your results.

Initialization metrics can be found in Appendix ([Config TCP], network = hw1\_network\_tcp).

It is seen that the outgoing data rate starts from 0 and increases until the threshould value is reached, then the data rate is configured with the fixed up and down values. After reaching up value, the datarate is cut to down value, and increases linearly with a frequency.

A screen shot of a barcode

Description automatically generated with low confidence

Figure Datarate for 3.1.

## 3.1.c. What is the sender window size? What is the round trip time? Calculate the mean data rate using W and RTT. Do your findings match the measured results? Explain.

☹

## 3.1.d. Plot congestion window vs time graph. Describe what you observe. Comment on your results.

Initialization metrics can be found in Appendix ([Config TCP1], network = hw1\_network\_tcp).

As it can be seen from the graph, congestion windows increases linearly with the time. It can be said that there is no congestion in the system.

A picture containing text, line, plot, parallel

Description automatically generated

Figure Congestion Windows for 3.1.

## 3.2. Copy the same topology from Question 1. Change the bandwidth of the connection between the router and server to 50kbps.

## 3.2.a. What is the mean bandwidth that is used on the connection between client and router? Is this the expected result?

Initialization metrics can be found in Appendix ([Config TCP2], network = hw1\_network\_tcp).

Now we see that our bandwith is decreased to 50 kbps, as we defined for this case. It is showing an expected result.

A picture containing text, software, diagram, plot

Description automatically generated

Figure Simulation Results for 3.2.

## 3.2.b. Plot data rate vs time graph. Describe what you observe. Comment on your results

Initialization metrics can be found in Appendix ([Config TCP2], network = hw1\_network\_tcp).

In this case, we see a similar behavior to the 100 kbps case (slow start, reach threshold and cut). However, this time we see timeout events where the datarate drops to zero, and follows the slow start again.

A picture containing text, line, screenshot, parallel

Description automatically generated

Figure Datarate for 3.2.

## 3.2.c. What is the sender window size? What is the round trip time? Calculate the mean data rate using W and RTT. Do your findings match the measured results? Explain.

☹

## 3.2.d. Plot congestion window vs time graph. Describe what you observe. Comment on your results.

Initialization metrics can be found in Appendix ([Config TCP2], network = hw1\_network\_tcp).

With the congestion window, we see that timeouts disrupt the previous behavior of the TCP network and it is not linear.

A picture containing text, line, plot, number

Description automatically generated

Figure Congestion Windows for 3.2.

## 3.2.e. Compare your results with the previous question. Explain the differences in your findings.

When both cases are compared, it is observed that, our minimum line bandwidth defines our throughput and if the congestion happens, datarate starts from zero again and slow starts happen regularly. For congestion window, if no congestion happens, the graph is linear and if timeouts happen, we see logarithmic(?) increases in congestion windows with cuts and sharp increases.

## 3.3. Create two standard hosts as clients and another standard host as a server. Create one router with a queue size of 100 packets. Connect one client to the router with a 100kbps line. Connect the other client to the router with a 200kbps line. Connect the server to the router with a 100kbps line. Use ThruputMeteringChannel for connections with bandwidth measuring mode. For clients, use TcpSessionApp with a large byte count to simulate a continuous stream. Use TcpSinkApp for the server.

## 3.3.a. What is the mean bandwidth that is used on the connection between client and router? Is this the expected result?

Initialization metrics can be found in Appendix ([Config TCP3], network = hw1\_network\_tcp2).

We observe that both clients have around 50kbps bandwidth, although their bandwidth rates are higher (100 kbps and 200 kbps). The bandwidth is limited by the smallest bandwidth, which is the server-router bandwidth, 100 kbps. The sum of 2 bandwidths from clients and router is equal to that limit, so that is the reason we have less bandwidth than clients’ limits. As the limit is reached, the bandwidth is divided equally to each client.

A screenshot of a computer

Description automatically generated with medium confidence

Figure Simulation Results for 3.3.

## 3.3.b. Plot data rate vs time graph for each client. Describe what you observe. Comment on your results

Initialization metrics can be found in Appendix ([Config TCP3], network = hw1\_network\_tcp2).

We see that initially the client 2 has more bandwidth, but then client1 and client2 shares the same bandwidth although they have different bandwidth settings. This equal division happens since after some time, congestion happens and the network bandwidth is divided to clients equally.

A picture containing text, screenshot, line, plot

Description automatically generated

Figure Datarate for 3.3.

## 3.4. Copy the same topology from Question 3. Change the bandwidth of the connection between the router and server to 240kbps.

## 3.4.a. What is the mean bandwidth that is used on the connection between client and router? Is this the expected result? Explain the changes from the previous question

Initialization metrics can be found in Appendix ([Config TCP4], network = hw1\_network\_tcp2).

It is observed that, now client2 has more bandwidth than client1 since the limiting bandwidth is sufficient enough to divide the bandwidth according to the clients’ bandwidth settings. For client1, we see that the bandwidth reached to the bandwidth setting of the client1. When the clients’ bandwiths are summed, we see that it is around 240 kbps, which is the network limit due to the bandwidth of server-router bandwidth.

A picture containing screenshot, text, diagram, line

Description automatically generated

Figure Simulation Results for 3.4.

## 3.4.b. Plot data rate vs time graph for each client. Describe what you observe. Comment on your results.

Initialization metrics can be found in Appendix ([Config TCP4], network = hw1\_network\_tcp2).

It is observed that Client 1’s bandwidth changes around 140kbps and 100kbps meanwhile Client 2’s bandwidth changes around 180kbps and 100 kbps. There are times that Client 1 passes it’s own limit, and as this limit is passed, it applies a cut to decreases it under 100 kbps again. For client 2, the cutting happens for 240kbps for some time, then it is stabilized at 140 kbps since network bandwidth limit is reached. Client 2 bandwidth then drops to zero. This can be due to the settings that are applied to TCP with OMNeT defaults.

A picture containing text, line, plot, screenshot

Description automatically generated

Figure Datarate for 3.4.

## 3.5. Copy the same topology from Question 4. Now, make the client with the higher bandwidth run two apps instead of one.

## 3.5.a. What is the mean bandwidth that is used on the connection between client and router? Is this the expected result? Explain the changes from the previous question.

Initialization metrics can be found in Appendix ([Config TCP5], network = hw1\_network\_tcp2).

As we have an additional app for client2 now, it is able to get more bandwidth instead of getting the remaining bandwidth. The bandwidth is shared equally between 3 apps, around 80 kbps per app.

A screenshot of a computer

Description automatically generated with medium confidence

Figure Simulation Results for 3.5.

## 3.5.b. Is the obtained result fair? What are the consequences of running multiple apps?

If the observation is considered for app-based share, it is fair since it gives more bandwidth for multiple apps. It slows the overall network, but gives equal priority to each app.

If the observation is considered for client-based share, it is not fair. As the question asks, if a client runs multiple apps, it could use a significant amount of the bandwidth itself, and other clients in the server might not get sufficient bandwidth for their purpose.

## 3.5.c. Plot data rate vs time graph for each client. Describe what you observe. Comment on your results.

Initialization metrics can be found in Appendix ([Config TCP5], network = hw1\_network\_tcp2).

We observe that, due to the app number difference, Client 2 has more bandwidth when it is compared with Client 1.

A picture containing text, screenshot, plot, line

Description automatically generated

Figure Datarate for 3.5.

A screenshot of a social media post

Description automatically generated

Figure My Mental State

# Appendix A. omnetpp.ini file

**[General]**

**total-stack** = 7MiB

**[Config Part1c\_ALL]**

**network** = hw1\_network

**sim-time-limit** = 10s

**record-eventlog** = true

hw1\_network.nodeCount = ${N = 2, 4, 8, 16, 32, 64, 128}

hw1\_network.hosts[\*].cli.sendInterval = exponential(0.01s) *# Send interval for all hosts*

\*\*.hosts[0].cli.destAddress = ""

\*\*.cli.destAddress = "hosts[0]"

hw1\_network.hosts[\*].cli.reqLength = 128B *# Request length for all hosts*

hw1\_network.hosts[\*].cli.respLength = 128B *# Response length for all hosts*

**[Config Part1d\_ALL]**

**network** = hw1\_network

**sim-time-limit** = 10s

**record-eventlog** = true

hw1\_network.nodeCount = 16

hw1\_network.hosts[\*].cli.sendInterval = exponential(0.01s) *# Send interval for all hosts*

\*\*.hosts[0].cli.destAddress = ""

\*\*.cli.destAddress = "hosts[0]"

hw1\_network.hosts[\*].cli.reqLength = ${N = 64B, 128B, 256B, 512B, 1024B} *# Request length for all hosts*

hw1\_network.hosts[\*].cli.respLength = ${N} *# Response length for all hosts*

**[Config Part1e\_ALL]**

**network** = hw1\_network

**sim-time-limit** = 10s

**record-eventlog** = true

hw1\_network.nodeCount = 16

hw1\_network.hosts[\*].cli.sendInterval = exponential(0.005s) *# Send interval for all hosts*

\*\*.hosts[0].cli.destAddress = ""

\*\*.cli.destAddress = "hosts[0]"

hw1\_network.hosts[\*].cli.reqLength = 128B *# Request length for all hosts*

hw1\_network.hosts[\*].cli.respLength = 128B *# Response length for all hosts*

**[Config Part2c\_ALL]**

**network** = hw1\_network2

**sim-time-limit** = 10s

**record-eventlog** = true

hw1\_network2.nodeCount = ${N = 2, 4, 8, 16, 32, 64, 128}

hw1\_network2.hosts[\*].cli.sendInterval = exponential(0.01s) *# Send interval for all hosts*

\*\*.hosts[0].cli.destAddress = ""

\*\*.cli.destAddress = "hosts[0]"

hw1\_network2.hosts[\*].cli.reqLength = 128B *# Request length for all hosts*

hw1\_network2.hosts[\*].cli.respLength = 128B *# Response length for all hosts*

**[Config Part2d\_ALL]**

**network** = hw1\_network2

**sim-time-limit** = 10s

**record-eventlog** = true

hw1\_network2.nodeCount = 16

hw1\_network2.hosts[\*].cli.sendInterval = exponential(0.01s) *# Send interval for all hosts*

\*\*.hosts[0].cli.destAddress = ""

\*\*.cli.destAddress = "hosts[0]"

hw1\_network2.hosts[\*].cli.reqLength = ${N = 64B, 128B, 256B, 512B, 1024B} *# Request length for all hosts*

hw1\_network2.hosts[\*].cli.respLength = ${N} *# Response length for all hosts*

**[Config Part2e\_ALL]**

**network** = hw1\_network2

**sim-time-limit** = 10s

**record-eventlog** = true

hw1\_network2.nodeCount = 16

hw1\_network2.hosts[\*].cli.sendInterval = exponential(0.005s) *# Send interval for all hosts*

\*\*.hosts[0].cli.destAddress = ""

\*\*.cli.destAddress = "hosts[0]"

hw1\_network2.hosts[\*].cli.reqLength = 128B *# Request length for all hosts*

hw1\_network2.hosts[\*].cli.respLength = 128B *# Response length for all hosts*

**[Config TCP1]**

**network** = hw1\_network\_tcp

**sim-time-limit** = 100s

**record-eventlog** = true

hw1\_network\_tcp.server.numPcapRecorders = 1

hw1\_network\_tcp.server.pcapRecorder[0].pcapFile = "results/server.pcap"

hw1\_network\_tcp.client.numPcapRecorders = 1

hw1\_network\_tcp.client.pcapRecorder[0].pcapFile = "results/client1.pcap"

\*\*.crcMode = "computed"

*## tcp apps*

hw1\_network\_tcp.client.numApps = 1

hw1\_network\_tcp.client.app[0].**typename** = "TcpSessionApp"

hw1\_network\_tcp.client.app[0].active = true

hw1\_network\_tcp.client.app[0].localPort = 1000

hw1\_network\_tcp.client.app[0].connectAddress = "server"

hw1\_network\_tcp.client.app[0].connectPort = 1000

hw1\_network\_tcp.client.app[0].tOpen = 0.2s

hw1\_network\_tcp.client.app[0].tSend = 0.4s

hw1\_network\_tcp.client.app[0].sendBytes = 1000000B

hw1\_network\_tcp.client.app[0].tClose = 25s

hw1\_network\_tcp.server.numApps = 1

hw1\_network\_tcp.server.app[0].**typename** = "TcpSinkApp"

hw1\_network\_tcp.server.app[0].localPort = 1000

*# NIC configuration*

hw1\_network\_tcp.router.ppp[\*].queue.**typename** = "DropTailQueue" *# in routers*

hw1\_network\_tcp.router.ppp[\*].queue.packetCapacity = 10 *# in routers*

**[Config TCP2]**

**network** = hw1\_network\_tcp

**sim-time-limit** = 100s

**record-eventlog** = true

hw1\_network\_tcp.server.numPcapRecorders = 1

hw1\_network\_tcp.server.pcapRecorder[0].pcapFile = "results/server.pcap"

hw1\_network\_tcp.client.numPcapRecorders = 1

hw1\_network\_tcp.client.pcapRecorder[0].pcapFile = "results/client1.pcap"

\*\*.crcMode = "computed"

*## tcp apps*

hw1\_network\_tcp.client.numApps = 1

hw1\_network\_tcp.client.app[0].**typename** = "TcpSessionApp"

hw1\_network\_tcp.client.app[0].active = true

hw1\_network\_tcp.client.app[0].localPort = 1000

hw1\_network\_tcp.client.app[0].connectAddress = "server"

hw1\_network\_tcp.client.app[0].connectPort = 1000

hw1\_network\_tcp.client.app[0].tOpen = 0.2s

hw1\_network\_tcp.client.app[0].tSend = 0.4s

hw1\_network\_tcp.client.app[0].sendBytes = 1000000B

hw1\_network\_tcp.client.app[0].tClose = 25s

hw1\_network\_tcp.server.numApps = 1

hw1\_network\_tcp.server.app[0].**typename** = "TcpSinkApp"

hw1\_network\_tcp.server.app[0].localPort = 1000

*# NIC configuration*

hw1\_network\_tcp.router.ppp[\*].queue.**typename** = "DropTailQueue" *# in routers*

hw1\_network\_tcp.router.ppp[\*].queue.packetCapacity = 10 *# in routers*

**[Config TCP3]**

**network** = hw1\_network\_tcp2

**sim-time-limit** = 100s

**record-eventlog** = true

hw1\_network\_tcp2.server.numPcapRecorders = 1

hw1\_network\_tcp2.server.pcapRecorder[0].pcapFile = "results/server.pcap"

hw1\_network\_tcp2.client\*.numPcapRecorders = 1

hw1\_network\_tcp2.client\*.pcapRecorder[0].pcapFile = "results/client1.pcap"

\*\*.crcMode = "computed"

*## tcp apps*

hw1\_network\_tcp2.client\*.numApps = 1

hw1\_network\_tcp2.client\*.app[\*].**typename** = "TcpSessionApp"

hw1\_network\_tcp2.client\*.app[0].active = true

hw1\_network\_tcp2.client\*.app[0].localPort = 1000

hw1\_network\_tcp2.client\*.app[0].connectAddress = "server"

hw1\_network\_tcp2.client\*.app[0].connectPort = 1000

hw1\_network\_tcp2.client\*.app[0].tOpen = 0.2s

hw1\_network\_tcp2.client\*.app[0].tSend = 0.4s

hw1\_network\_tcp2.client\*.app[0].sendBytes = 1000000B

hw1\_network\_tcp2.client\*.app[0].tClose = 25s

hw1\_network\_tcp2.server.numApps = 1

hw1\_network\_tcp2.server.app[0].**typename** = "TcpSinkApp"

hw1\_network\_tcp2.server.app[0].localPort = 1000

*# NIC configuration*

hw1\_network\_tcp2.router.ppp[\*].queue.**typename** = "DropTailQueue" *# in routers*

hw1\_network\_tcp2.router.ppp[\*].queue.packetCapacity = 100 *# in routers*

**[Config TCP4]**

**network** = hw1\_network\_tcp2

**sim-time-limit** = 100s

**record-eventlog** = true

hw1\_network\_tcp2.server.numPcapRecorders = 1

hw1\_network\_tcp2.server.pcapRecorder[0].pcapFile = "results/server.pcap"

hw1\_network\_tcp2.client\*.numPcapRecorders = 1

hw1\_network\_tcp2.client\*.pcapRecorder[0].pcapFile = "results/client1.pcap"

\*\*.crcMode = "computed"

*## tcp apps*

hw1\_network\_tcp2.client\*.numApps = 1

hw1\_network\_tcp2.client\*.app[\*].**typename** = "TcpSessionApp"

hw1\_network\_tcp2.client\*.app[0].active = true

hw1\_network\_tcp2.client\*.app[0].localPort = 1000

hw1\_network\_tcp2.client\*.app[0].connectAddress = "server"

hw1\_network\_tcp2.client\*.app[0].connectPort = 1000

*#hw1\_network\_tcp2.client\*.app[0].tOpen = 0.2s*

*#hw1\_network\_tcp2.client\*.app[0].tSend = 0.4s*

hw1\_network\_tcp2.client\*.app[0].sendBytes = 1000000B

*#hw1\_network\_tcp2.client\*.app[0].tClose = 25s*

hw1\_network\_tcp2.server.numApps = 1

hw1\_network\_tcp2.server.app[0].**typename** = "TcpSinkApp"

hw1\_network\_tcp2.server.app[0].localPort = 1000

*# NIC configuration*

hw1\_network\_tcp2.router.ppp[\*].queue.**typename** = "DropTailQueue" *# in routers*

hw1\_network\_tcp2.router.ppp[\*].queue.packetCapacity = 100 *# in routers*

**[Config TCP5]**

**network** = hw1\_network\_tcp2

**sim-time-limit** = 100s

**record-eventlog** = true

hw1\_network\_tcp2.server.numPcapRecorders = 1

hw1\_network\_tcp2.server.pcapRecorder[0].pcapFile = "results/server.pcap"

hw1\_network\_tcp2.client\*.numPcapRecorders = 1

hw1\_network\_tcp2.client\*.pcapRecorder[0].pcapFile = "results/client1.pcap"

\*\*.crcMode = "computed"

*## tcp apps*

hw1\_network\_tcp2.client1.numApps = 1

hw1\_network\_tcp2.client2.numApps = 2

hw1\_network\_tcp2.client\*.app[\*].**typename** = "TcpSessionApp"

hw1\_network\_tcp2.client\*.app[\*].active = true

hw1\_network\_tcp2.client1.app[\*].localPort = 1000

hw1\_network\_tcp2.client2.app[0].localPort = 1000

hw1\_network\_tcp2.client2.app[1].localPort = 1001

hw1\_network\_tcp2.client\*.app[\*].connectAddress = "server"

hw1\_network\_tcp2.client1.app[0].connectPort = 1000

hw1\_network\_tcp2.client2.app[1].connectPort = 1001

*#hw1\_network\_tcp2.client\*.app[\*].tOpen = 0.2s*

*#hw1\_network\_tcp2.client\*.app[\*].tSend = 0.4s*

hw1\_network\_tcp2.client\*.app[\*].sendBytes = 1000000B

*#hw1\_network\_tcp2.client\*.app[\*].tClose = 25s*

hw1\_network\_tcp2.server.numApps = 2

hw1\_network\_tcp2.server.app[\*].**typename** = "TcpSinkApp"

hw1\_network\_tcp2.server.app[0].localPort = 1000

hw1\_network\_tcp2.server.app[1].localPort = 1001

*# NIC configuration*

hw1\_network\_tcp2.router.ppp[\*].queue.**typename** = "DropTailQueue" *# in routers*

hw1\_network\_tcp2.router.ppp[\*].queue.packetCapacity = 100 *# in routers*

# Appendix B. .ned file

**import** inet.node.ethernet.Eth10M;

**import** inet.node.ethernet.EthernetHost;

**import** inet.node.ethernet.EthernetSwitch;

**import** inet.physicallayer.wired.common.WireJunction;

**import** inet.tests.ethernet.EthernetHost2;

**import** inet.networklayer.configurator.ipv4.Ipv4NetworkConfigurator;

**import** inet.node.inet.StandardHost;

**import** inet.common.misc.ThruputMeteringChannel;

**import** inet.node.inet.Router;

**network** hw1\_network

{

**parameters**:

**int** nodeCount = **default**(16);

**@display**("bgb=6600,500");

**submodules**:

wireJunctions[nodeCount]: WireJunction {

**@display**("p=100,200,r,50");

}

hosts[nodeCount]: EthernetHost {

**@display**("p=100,300,r,50");

}

**connections**:

**for** i=0..nodeCount-2 {

wireJunctions[i].port++ **<-->** Eth10M **<-->** wireJunctions[i+1].port++;

hosts[i].ethg **<-->** Eth10M **<-->** wireJunctions[i].port++;

}

hosts[nodeCount-1].ethg **<-->** Eth10M **<-->** wireJunctions[nodeCount-1].port++;

}

**network** hw1\_network2

{

**parameters**:

**int** nodeCount = **default**(4);

**@display**("bgb=6600,500");

**submodules**:

wireJunctions[nodeCount]: WireJunction {

**@display**("p=100,200,r,50");

}

hosts[nodeCount]: EthernetHost {

**@display**("p=100,300,r,50");

}

ethernetSwitch: EthernetSwitch {

**gates**:

ethg[2];

}

**connections**:

**for** i=0..nodeCount-2, **if** i != **int**(nodeCount/2)-1 {

wireJunctions[i].port++ **<-->** Eth10M **<-->** wireJunctions[i+1].port++;

hosts[i].ethg **<-->** Eth10M **<-->** wireJunctions[i].port++;

}

**for** i=0..nodeCount-2, **if** i == **int**(nodeCount/2)-1 {

wireJunctions[i].port++ **<-->** Eth10M **<-->** ethernetSwitch.ethg[0];

wireJunctions[i+1].port++ **<-->** Eth10M **<-->** ethernetSwitch.ethg[1];

hosts[i].ethg **<-->** Eth10M **<-->** wireJunctions[i].port++;

}

hosts[nodeCount-1].ethg **<-->** Eth10M **<-->** wireJunctions[nodeCount-1].port++;

}

**network** hw1\_network\_tcp

{

**types**:

**channel** tc1 **extends** ThruputMeteringChannel

{

thruputDisplayFormat = "B";

datarate = 0.1Mbps;

}

**channel** tc2 **extends** ThruputMeteringChannel

{

thruputDisplayFormat = "B";

*//datarate = 1Mbps; // TCP - Q1*

datarate = 0.05Mbps; *// TCP - Q2*

}

**submodules**:

configurator: Ipv4NetworkConfigurator {

**@display**("p=30,30");

}

client: StandardHost {

**@display**("p=400,90");

}

router: Router {

**@display**("p=300,90");

}

server: StandardHost {

**@display**("p=200,90");

}

**connections**:

client.pppg++ **<-->** tc1 **<-->** router.pppg++;

server.pppg++ **<-->** tc2 **<-->** router.pppg++;

}

**network** hw1\_network\_tcp2

{

**types**:

**channel** tc1 **extends** ThruputMeteringChannel

{

thruputDisplayFormat = "B";

datarate = 0.1Mbps;

}

**channel** tc2 **extends** ThruputMeteringChannel

{

thruputDisplayFormat = "B";

*//datarate = 0.1Mbps; // TCP - Q1*

datarate = 0.240Mbps; *// TCP - Q2*

}

**channel** tc3 **extends** ThruputMeteringChannel

{

thruputDisplayFormat = "B";

datarate = 0.2Mbps;

}

**submodules**:

configurator: Ipv4NetworkConfigurator {

**@display**("p=30,30");

}

client1: StandardHost {

**@display**("p=400,60");

}

client2: StandardHost {

**@display**("p=400,120");

}

router: Router {

**@display**("p=300,90");

}

server: StandardHost {

**@display**("p=200,90");

}

**connections**:

client1.pppg++ **<-->** tc1 **<-->** router.pppg++;

client2.pppg++ **<-->** tc3 **<-->** router.pppg++;

server.pppg++ **<-->** tc2 **<-->** router.pppg++;

}

# Appendix C. Python\_Plot.ipynb file

import matplotlib.pyplot as plt

# received bits by hosts

ETH10M = 10000000

node\_2 = 126720\*8

node\_4 = 368512\*8

node\_8 = 774144\*8

node\_16 = 1359232\*8

node\_32 = 1866496\*8

node\_64 = 1697280\*8

node\_128 = 663680\*8

labels = ['# of Nodes', 'Received Bits', 'Channel Efficiency']

nodes = ['2', '4', '8', '16', '32', '64', '128']

received\_bits = [node\_2, node\_4, node\_8, node\_16, node\_32, node\_64, node\_128]

channel\_efficiency = []

for i in range(len(received\_bits)):

    channel\_efficiency.append(received\_bits[i]/10/ETH10M\*100) # divide by 10 to get per seconds, divide by ETH10M to get efficiency, multiply by 100 to get percentage

print(channel\_efficiency)

#plot it as a bar graph

plt.bar(nodes, channel\_efficiency)

plt.xlabel('Number of Nodes (n)')

plt.ylabel('Channel Efficiency (%)')

plt.title('Number of Nodes vs. Channel Efficiency')

plt.grid(True, alpha=0.5)

# show values on top of bars

for i in range(len(channel\_efficiency)):

    plt.text(i, channel\_efficiency[i], str(channel\_efficiency[i])[:4], ha='center', va='bottom')

# add a line graph to bar graph

plt.plot(nodes, channel\_efficiency, color='red', marker='o')

plt.savefig('part1c.png')

plt.show()

import matplotlib.pyplot as plt

# received bits by hosts

ETH10M = 10000000

frame\_64 = 780480\*8

frame\_128 = 1348096\*8

frame\_256 = 1417216\*8

frame\_512 = 2033920\*8

frame\_1024 = 2216960\*8

labels = ['Frame Size', 'Received Bits', 'Channel Efficiency']

frames = ['64', '128', '256', '512', '1024']

received\_bits = [frame\_64, frame\_128, frame\_256, frame\_512, frame\_1024]

channel\_efficiency = []

for i in range(len(received\_bits)):

    channel\_efficiency.append(received\_bits[i]/10/ETH10M\*100) # divide by 10 to get per seconds, divide by ETH10M to get efficiency, multiply by 100 to get percentage

print(channel\_efficiency)

#plot it as a bar graph

plt.bar(frames, channel\_efficiency)

plt.xlabel('Frame Size (bytes)')

plt.ylabel('Channel Efficiency (%)')

plt.title('Frame Size vs. Channel Efficiency')

plt.grid(True, alpha=0.5)

# show values on top of bars

for i in range(len(channel\_efficiency)):

    plt.text(i, channel\_efficiency[i], str(channel\_efficiency[i])[:4], ha='center', va='bottom')

# add a line graph to bar graph

plt.plot(frames, channel\_efficiency, color='red', marker='o')

plt.savefig('part1d.png')

plt.show()

# received bits by hosts

ETH10M = 10000000

node\_2 = 126720\*8

node\_4 = 368512\*8

node\_8 = 774144\*8

node\_16 = 1359232\*8

node\_32 = 1866496\*8

node\_64 = 1697280\*8

node\_128 = 663680\*8

node\_16\_int\_new = 1898240\*8

labels = ['# of Nodes', 'Received Bits', 'Channel Efficiency']

nodes = ['2', '4', '8', '16', '32', '64', '128']

received\_bits = [node\_2, node\_4, node\_8, node\_16, node\_32, node\_64, node\_128]

channel\_efficiency = []

for i in range(len(received\_bits)):

    channel\_efficiency.append(received\_bits[i]/10/ETH10M\*100) # divide by 10 to get per seconds, divide by ETH10M to get efficiency, multiply by 100 to get percentage

new\_eff = node\_16\_int\_new/10/ETH10M\*100

new\_effs = [0, 0, 0, new\_eff, 0, 0, 0]

print(channel\_efficiency)

#plot it as a bar graph

plt.bar(nodes, new\_effs, color='green')

plt.bar(nodes, channel\_efficiency)

plt.xlabel('Number of Nodes (n)')

plt.ylabel('Channel Efficiency (%)')

plt.title('Number of Nodes vs. Channel Efficiency')

plt.grid(True, alpha=0.5)

# show values on top of bars

for i in range(len(channel\_efficiency)):

    plt.text(i, channel\_efficiency[i], str(channel\_efficiency[i])[:4], ha='center', va='bottom')

    if i == 3:

        plt.text(i, new\_effs[i], str(new\_effs[i])[:4], ha='center', va='bottom')

# add a line graph to bar graph

plt.plot(nodes, channel\_efficiency, color='red', marker='o')

plt.savefig('part1e.png')

plt.show()

import matplotlib.pyplot as plt

# received bits by hosts

ETH10M = 10000000

node\_2 = 126720\*8

node\_4 = 374016\*8

node\_8 = 809856\*8

node\_16 = 1488256\*8

node\_32 = 2278784\*8

node\_64 = 2598656\*8

node\_128 = 1771392\*8

labels = ['# of Nodes', 'Received Bits', 'Channel Efficiency']

nodes = ['2', '4', '8', '16', '32', '64', '128']

received\_bits = [node\_2, node\_4, node\_8, node\_16, node\_32, node\_64, node\_128]

channel\_efficiency = []

for i in range(len(received\_bits)):

    channel\_efficiency.append(received\_bits[i]/10/ETH10M\*100) # divide by 10 to get per seconds, divide by ETH10M to get efficiency, multiply by 100 to get percentage

print(channel\_efficiency)

#plot it as a bar graph

plt.bar(nodes, channel\_efficiency)

plt.xlabel('Number of Nodes (n)')

plt.ylabel('Channel Efficiency (%)')

plt.title('Number of Nodes vs. Channel Efficiency')

plt.grid(True, alpha=0.5)

# show values on top of bars

for i in range(len(channel\_efficiency)):

    plt.text(i, channel\_efficiency[i], str(channel\_efficiency[i])[:4], ha='center', va='bottom')

# add a line graph to bar graph

plt.plot(nodes, channel\_efficiency, color='red', marker='o')

plt.savefig('part2c.png')

plt.show()

import matplotlib.pyplot as plt

# received bits by hosts

ETH10M = 10000000

frame\_64 = 827456\*8

frame\_128 = 1484288\*8

frame\_256 = 2406144\*8

frame\_512 = 3180032\*8

frame\_1024 = 2880512\*8

labels = ['Frame Size', 'Received Bits', 'Channel Efficiency']

frames = ['64', '128', '256', '512', '1024']

received\_bits = [frame\_64, frame\_128, frame\_256, frame\_512, frame\_1024]

channel\_efficiency = []

for i in range(len(received\_bits)):

    channel\_efficiency.append(received\_bits[i]/10/ETH10M\*100) # divide by 10 to get per seconds, divide by ETH10M to get efficiency, multiply by 100 to get percentage

print(channel\_efficiency)

#plot it as a bar graph

plt.bar(frames, channel\_efficiency)

plt.xlabel('Frame Size (bytes)')

plt.ylabel('Channel Efficiency (%)')

plt.title('Number of Nodes vs. Channel Efficiency')

plt.grid(True, alpha=0.5)

# show values on top of bars

for i in range(len(channel\_efficiency)):

    plt.text(i, channel\_efficiency[i], str(channel\_efficiency[i])[:4], ha='center', va='bottom')

# add a line graph to bar graph

plt.plot(frames, channel\_efficiency, color='red', marker='o')

plt.savefig('part2d.png')

plt.show()

# received bits by hosts

ETH10M = 10000000

node\_2 = 126720\*8

node\_4 = 368512\*8

node\_8 = 774144\*8

node\_16 = 1359232\*8

node\_32 = 1866496\*8

node\_64 = 1697280\*8

node\_128 = 663680\*8

node\_16\_int\_new = 2304896\*8

labels = ['# of Nodes', 'Received Bits', 'Channel Efficiency']

nodes = ['2', '4', '8', '16', '32', '64', '128']

received\_bits = [node\_2, node\_4, node\_8, node\_16, node\_32, node\_64, node\_128]

channel\_efficiency = []

for i in range(len(received\_bits)):

    channel\_efficiency.append(received\_bits[i]/10/ETH10M\*100) # divide by 10 to get per seconds, divide by ETH10M to get efficiency, multiply by 100 to get percentage

new\_eff = node\_16\_int\_new/10/ETH10M\*100

new\_effs = [0, 0, 0, new\_eff, 0, 0, 0]

print(channel\_efficiency)

#plot it as a bar graph

plt.bar(nodes, new\_effs, color='green')

plt.bar(nodes, channel\_efficiency)

plt.xlabel('Number of Nodes (n)')

plt.ylabel('Channel Efficiency (%)')

plt.title('Number of Nodes vs. Channel Efficiency')

plt.grid(True, alpha=0.5)

# show values on top of bars

for i in range(len(channel\_efficiency)):

    plt.text(i, channel\_efficiency[i], str(channel\_efficiency[i])[:4], ha='center', va='bottom')

    if i == 3:

        plt.text(i, new\_effs[i], str(new\_effs[i])[:4], ha='center', va='bottom')

# add a line graph to bar graph

plt.plot(nodes, channel\_efficiency, color='red', marker='o')

plt.savefig('part2e.png')

plt.show()