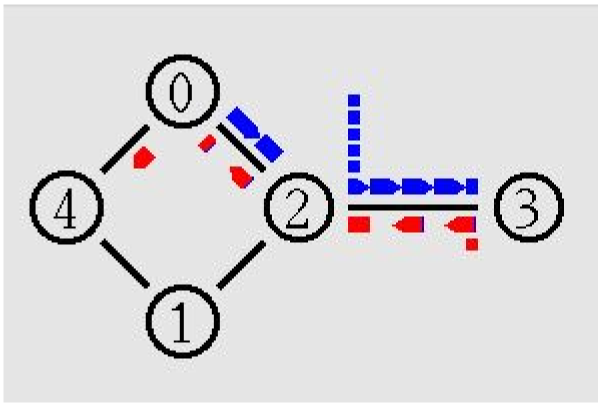
Data Communications & Networking (F29DC) Report

Drew Tomkins (H00294338)

Question 1: Lab Reports

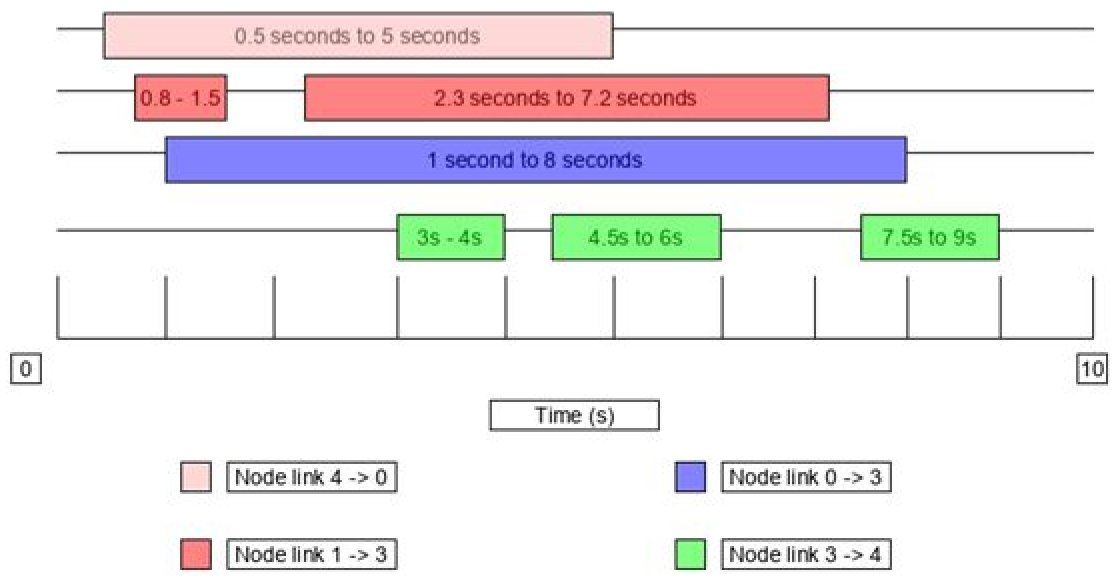
Lab 1: Link load and bandwidth

When node 1 transmits the packets from the cbr generator to node 2, it is transmitting it in fixed and consistent intervals of 8 nanoseconds with 1000 bytes in each packet whilst when node 0 is transmitting data packets from the ftp generator to node 2, it has more irregular patterns as it is sending packets according to the ACK packets it receives from the sink agent in node 3. When both nodes transmit packets at the same time, some of the packets start dropping once they reach node 2.



After updating the topology, I noticed that node 1 was being ignored by the simulator. I presume this is because there is no need to send the packets through two nodes at once as one is enough to send the data, so the simulator has determined node 0 is the optimal node to use in its simulation. I also changed the queue size for node 2 to 1 and 5 – only cbr packets were being sent from node 3 and on a queue size of 5 there were a lot more ftp data packets being sent within a very short amount of time of each other.

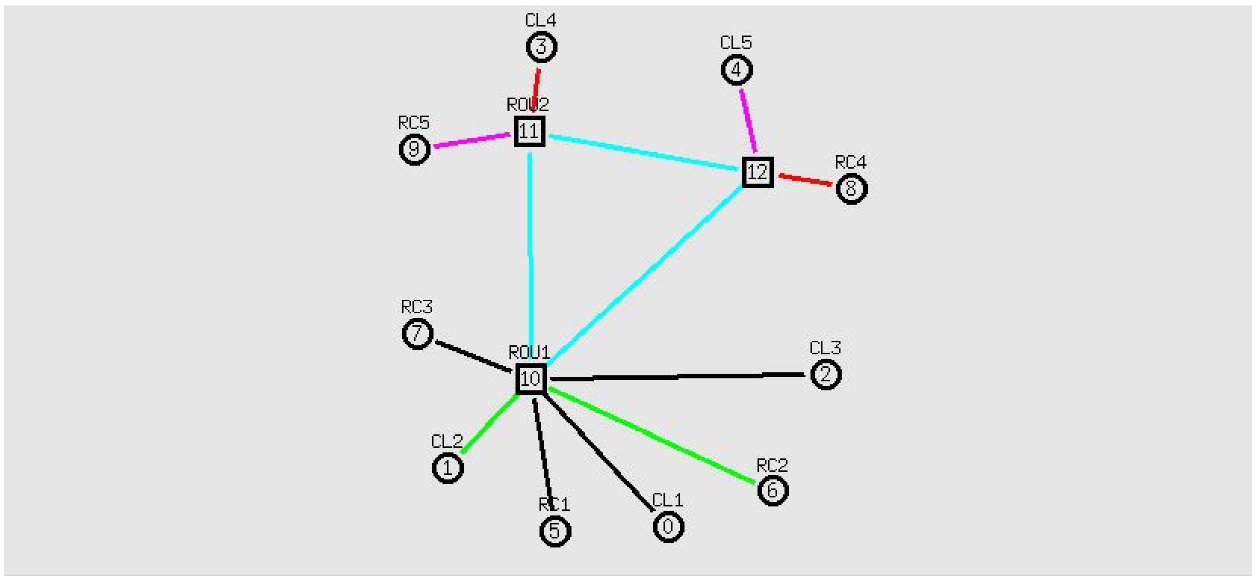
Lab 2: Congestion



After modifying the topology, I created a timeline representing when each node was communicating in the network. I used this to aid my understanding of the communications taking place - The 4 -> 0 and 3 ->4 connections had no issues, but I found that a decent amount of the data packets from nodes 1 and 0 going to 3 were dropped after reaching it. This is presumably due to the queue size for these links being 10, causing connection issues due to packages not being delivered.

Lab 3: Creating large, wired topologies

Firstly, I tried example 4 in the simulator, then I altered the colours of some of the links in the file. I also decided to add some nodes to the network to see if I could try and make the topology larger without messing anything up, giving them magenta coloured links to their respective routers in the process so I could tell them apart from pre-existing nodes.



After trying example 4, I ran scripts 7, 9 and 19. After trying out these scripts in the simulator, I experimented with various modifications to the scripts such as adding more nodes, and it helped me get a better idea in regard to making larger topologies.

Lab 4: Stop-N-Wait, Sliding-Window and Trace File Analysis

After looking at both features, I noticed that the Sliding-Window seemed to be the superior choice as it appeared to drop less data packets compared to the Stop-N-Wait feature. A notable thing about this is that the Sliding-Window continued to perform well even when network traffic was congested, so network bandwidth was put to good use.

As for trace files, nothing seemed too out of the ordinary when taking the context of the .tcl files into account with each trace being noted down, however I did notice that the example4.tr file was empty – presumably due to no actual communications happening, however I feel as if I understand trace files well now as a result of looking over them.

Lab 5: Dynamic networks and creating output files for Xgraph

To create the first dynamic network, I used the example provided as a base to work off of given the already present node orientation and dynamic routing. After modifying it to include the broken link, I observed it with both DV and LS protocols and noticed that the DV version sent the routing table to linked nodes whilst the LS version sent information to all the networks routers regarding connected links. I also noticed that the LS protocol seemed slightly faster, however it could just be me but I would say it is a superior protocol to the DV protocol.

In regards to the Xgraph output, I was unable to get any Xgraph window to appear despite me running the example provided with no changes. Traffic sources were recorded though and show up as files. Instead, I used jTrcezer to visualize and analyse my data, which proved to be effective in helping me understand what happened in the network simulations.

Lab 6: Ping

I tested numerous commands in Linux such as using the ping command on a website and using the ping echo command. Then I observed the ping.tcl file in the simulator and noticed the ping requests being sent to and back from nodes. This is presumably so that the network can verify that the host can receive data by pinging it to see if it responds.

Then I researched RTT and found out that it is how long a ping request can take to go from one node/host to another and back. RTT can be used to measure throughput by checking to see how long it takes for the ping request to finish, with longer ping times indicating a worser output, for example.

Lab 7: RED (Random Early Detection) Queue Management

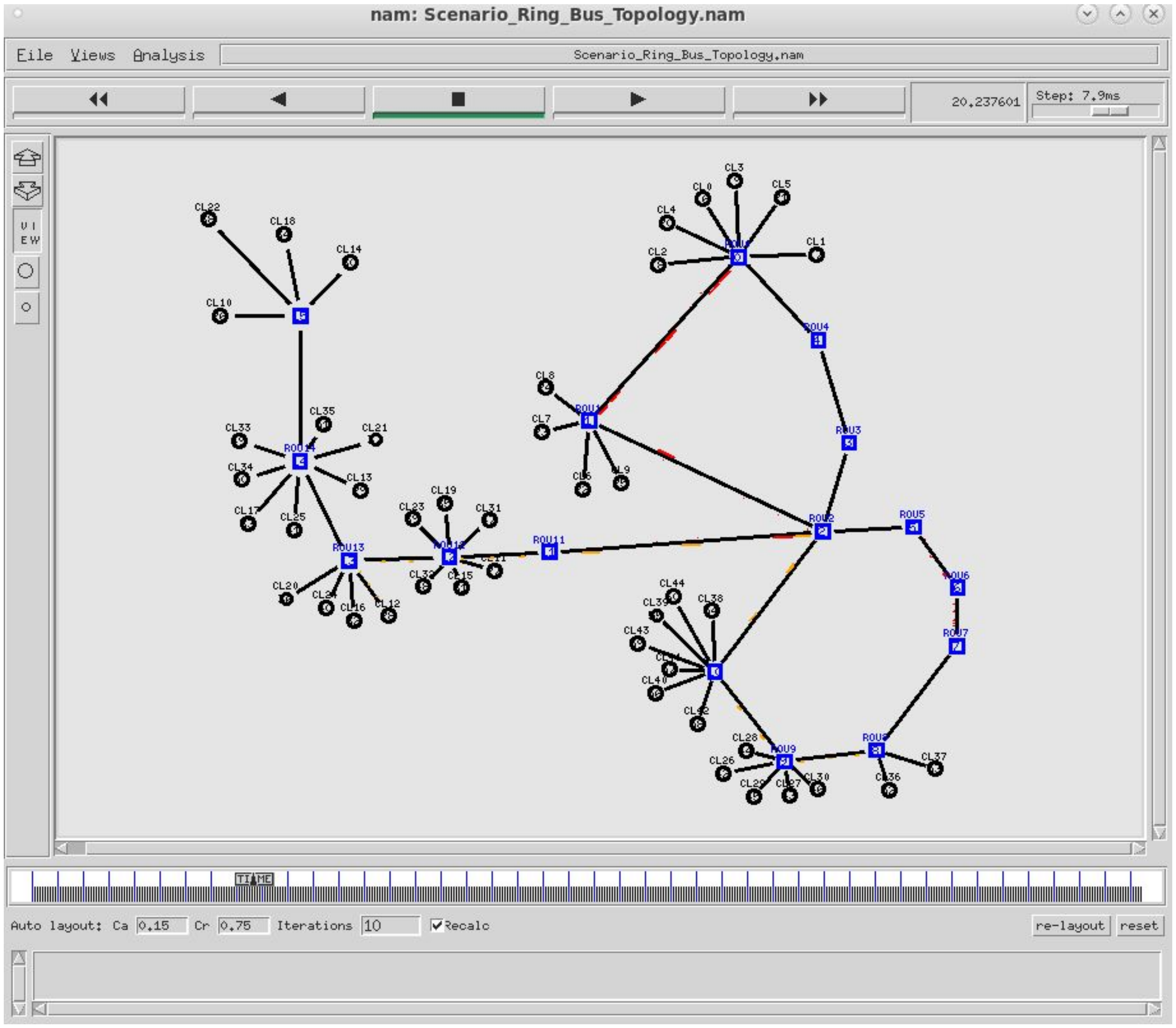
(note: I couldn’t get XGraph to work like mentioned in a previous lab so I couldn’t do exercise 2)

After looking at the code, the first thing that stood out aside from code that was already in other .tcl files I've been working with was the 'create-connection TCP/Reno' line, presumably this is for establishing TCP connections. There was also the section for tracing the queue for data packets so that the queue size can be tracked so I can see it in the XGraph plot. Then the code would let the simulator check the queue size and print results to the trace file before said queue size is used to determine whether packets go through or are discarded.

Some differences that I could find between the RED mechanism and the DropTail queue include the DropTail queue dropping packets if the amount of packets received by the queue is more than the current buffer size whilst the RED mechanism checks the queue length and if the queue length is too large then it drops packets randomly due to congestion, so at a glance the DropTail queue is more passive with dealing with congestion whilst the RED mechanism actively attempts to prevent congestion.

Question 2: Topologies Report

Ring Topology


nam: Scenario_Ring_Bus_Topology.nam
File Views Analysis
Scenario_Ring_Bus_Topology.nam
20.237601
Step: 7.9ms
Auto layout: Ca 0.15 Cr 0.75 Iterations 10 Recalc
re-layout reset

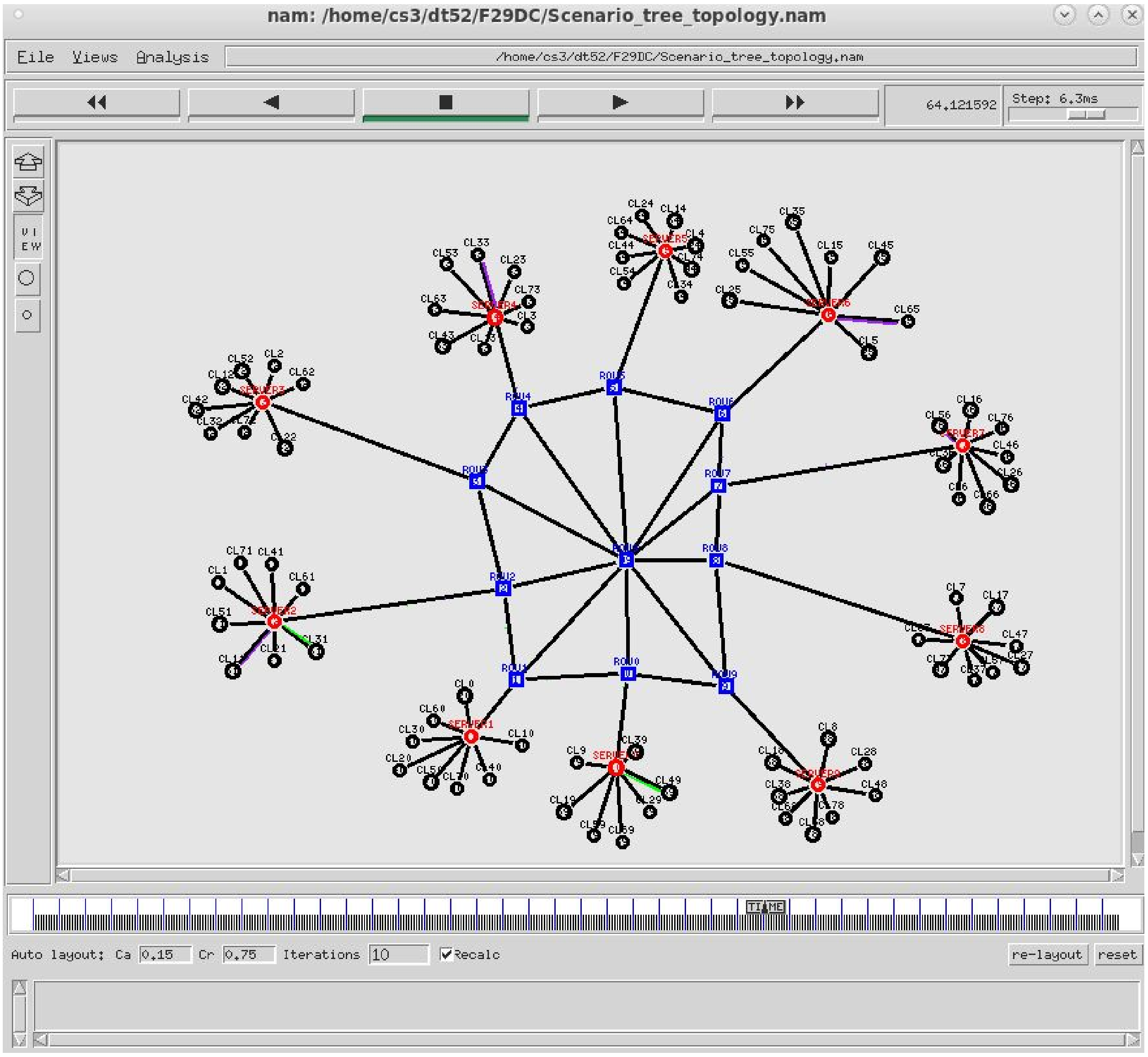
Advantages

Ring topologies such as what is featured in my simulation are able to easily avoid packet collisions due to having their packets flow in one direction, meaning that less if any data is dropped which ensures that communications between the nodes are smooth - I didn't notice any packet loss in my simulation. Additionally, I could go and add more nodes to one of the ring topologies and it wouldn't affect the performance as a result of the one direction flow.

Disadvantages

However, if a computer went offline, that would bring down or at least greatly affect my topology, and there's also an issue of cost as to allow the ring topology to function properly, a large amount of money needs to be spent on features and hardware to let it function, making it prohibitively expensive to use.

Star/Tree Topology


nam: /home/cs3/dt52/F29DC/Scenario_tree_topology.nam
File
Views
Analysis
/home/cs3/dt52/F29DC/Scenario_tree_topology.nam
64.121592
Step: 6.3ms
TIME
re-layout
reset

Auto layout: Ca 0.15 Cr 0.75 Iterations 10 ✓ Recalc

<u>Advantages</u>

A big advantage to using a star topology is that if one of the outer computers fails, then the rest of the network can continue to operate, meaning that overall a lot less disruption is caused. It's also easy to manage my network from the main computer and there's also the fact that computers can be added without disrupting the network much.

<u>Disadvantages</u>

If the central computer in my topology fails, then the whole network is brought down due to relying on it, causing mass disruption due to this. In addition to this, network performance relies on the performance of the central computer itself so if it's performing badly then the network will function poorly. Like ring topologies, they are also expensive to set up and maintain.

<u>Subnetting</u>

Advantages
In my topologies, I have subnet the topology in the first example into a bus topology and ring topology and then I subnet the star topology into multiple tree topologies connected to the star topology, by doing this I'm able to decrease possible congestion in the network because the load on the network is lower.

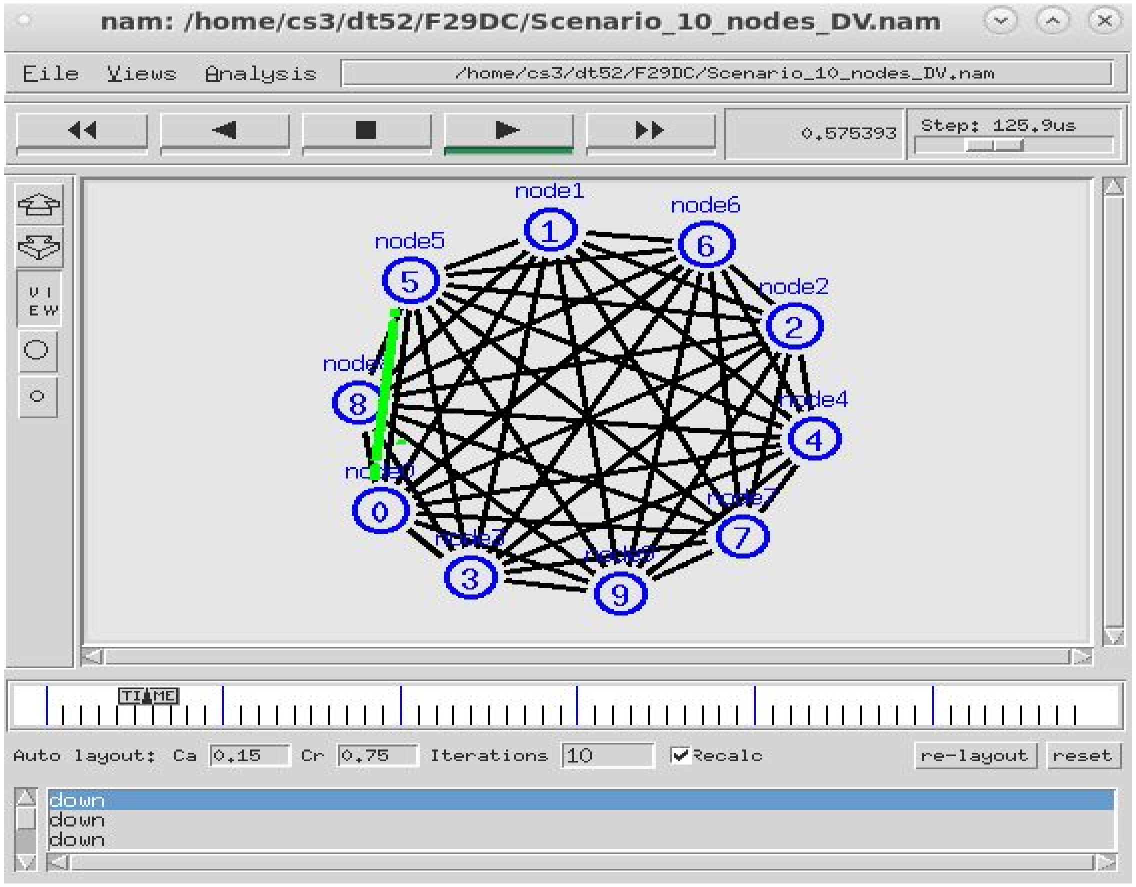
It also allows me to split off areas of the topology for various reasons such as security, meaning that if I have a tree topology that's important for the network, it is less likely to be affected by any potential network issues.

Disadvantages
The subnetting in my topologies would be expensive due to needing more equipment like routers to enable the subnetting, so it could potentially be too costly to maintain. There's also a potential for sluggish communication if the subnetted networks have a lot of hosts, meaning my topologies would be less efficient.

Questions 3 & 4: Routing Protocols Report

Distance Vector Protocol

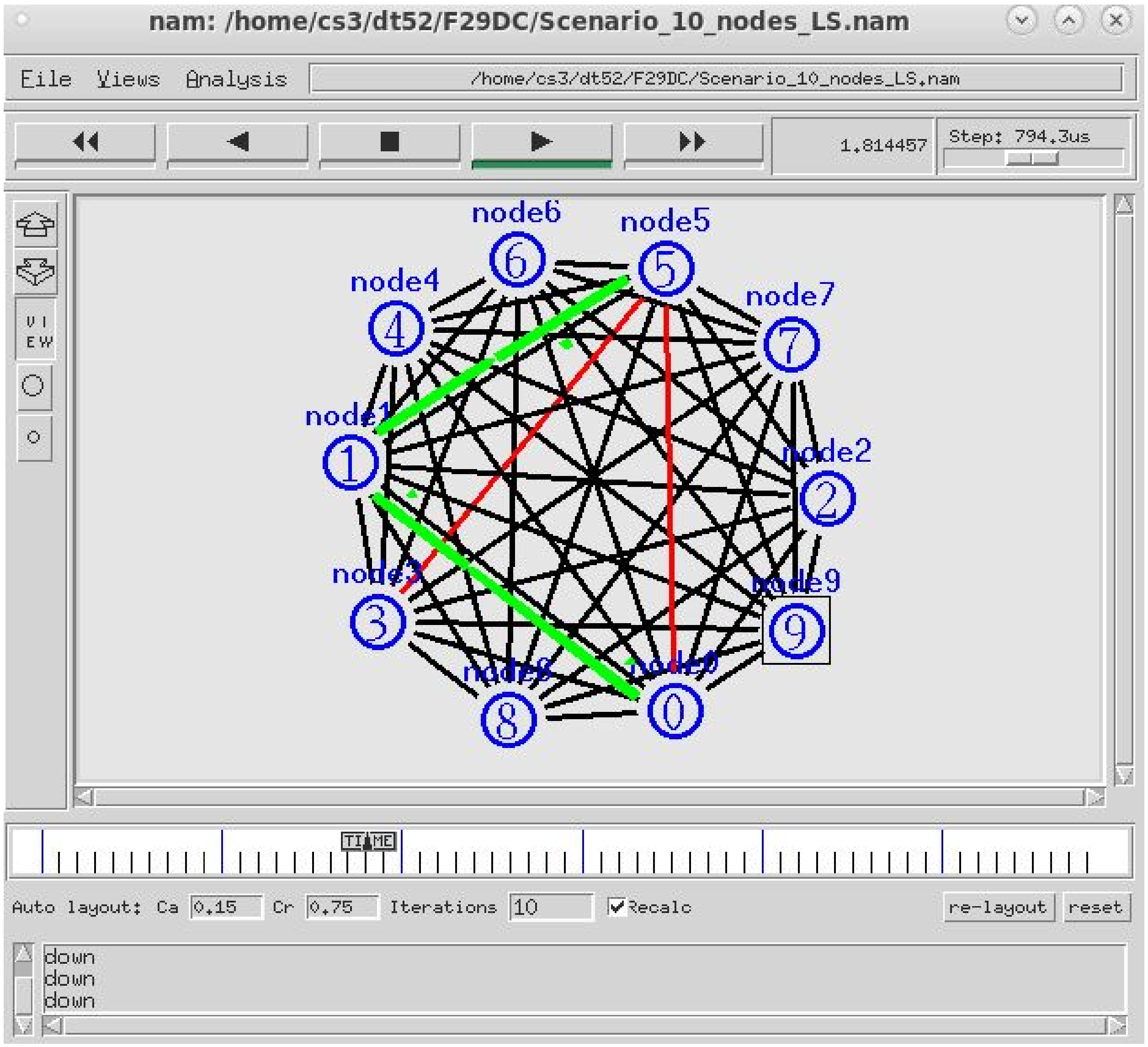
For Question 3, I decided to use a Distance Vector protocol for the initial topology (seen below) so that I would easily be able to compare it with a similar topology using the Link State protocol. I also made my topologies for this question into mesh network topologies so that I could more effectively test aspects such as what would happen if a link went down due to all the nodes being connected, letting the protocols pick another appropriate route to send data packets.


nam: /home/cs3/dt52/F29DC/Scenario_10_nodes_DV.nam
File
Views
Analysis
/home/cs3/dt52/F29DC/Scenario_10_nodes_DV.nam
0.575393
Step: 125.9us
node1
node6
node5
node2
node8
node4
node0
node7
node3
node9
TIME
Auto layout: Ca 0.15 Cr 0.75 Iterations 10 Recalc
re-layout
reset

```
down
down
down
```

<u>Link State Protocol</u>

My topologies and code for the LS protocol were virtually identical to the DV protocol topology, with the main difference being that the LS protocol was being used instead of the DV protocol.


nam: /home/cs3/dt52/F29DC/Scenario_10_nodes_LS.nam
File
Views
Analysis
/home/cs3/dt52/F29DC/Scenario_10_nodes_LS.nam
1.814457
Step: 794.3us
node6
node5
node4
node7
node1
node2
node3
node9
node8
node0
TIME
Auto layout: Ca 0.15 Cr 0.75 Iterations 10 Recalc
re-layout
reset

```
down
down
down
```

I created multiple versions of the topologies for the 2 protocols, with them having 3 each corresponding to the amount of nodes being used. Each protocol had a version with 10, 20 and 28 nodes. I intended to go higher than 28 nodes for my largest topology but with the method I was using to establish what links went down for my 20 node topologies to create better situations to analyse as well as saving time on coding, any higher than 28 nodes for the LS protocol topologies would give me errors which I couldn't debug due to lack of information. For the 10 node topologies, I input when the links went down manually for simplicity's sake.

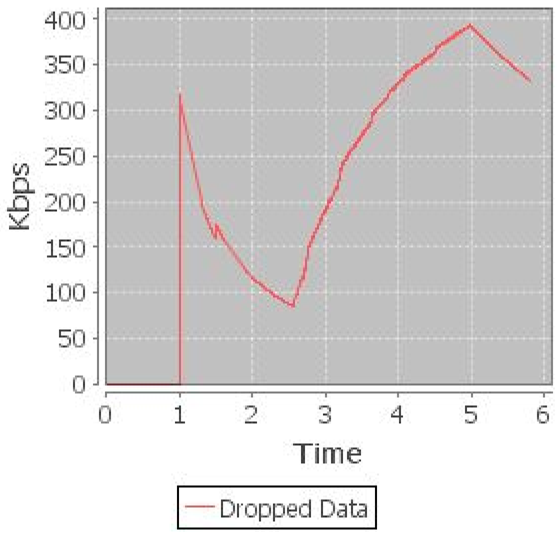
Comparison

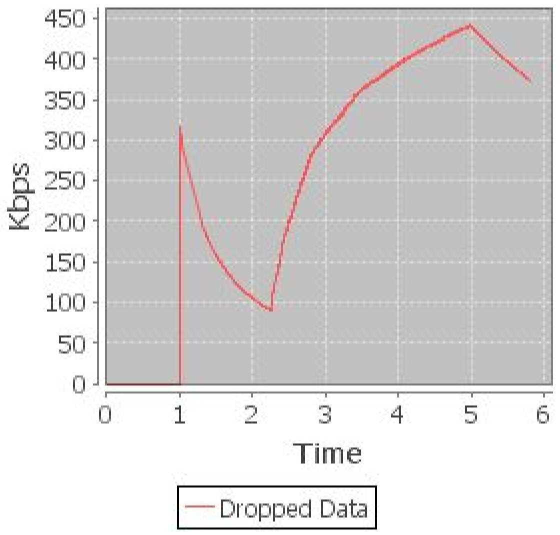
To determine which protocol is more efficient, I ran similar simulations on the aforementioned topologies for each protocol, measuring these factors to compare them: throughput, dropped packets, sent packets, forwarded packets and end-to-end delay. I used jTrcezer's graphs to visualize the data.

10 Node Comparisons (small topologies)

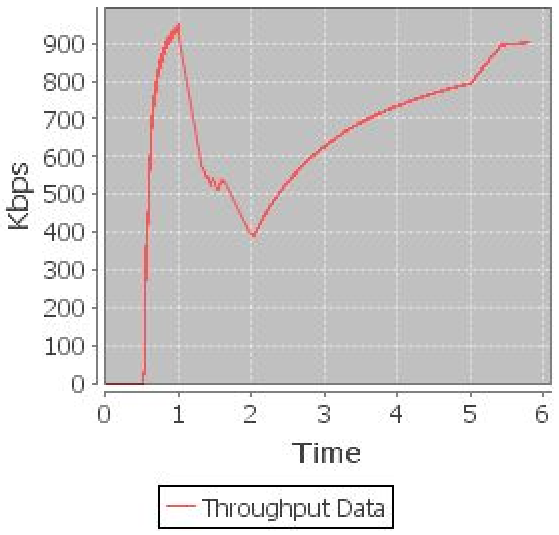
The 10 node topologies can be seen in the 2 most recent pictures. Results from running the simulations are as follows (DV protocol graphs are on the left hand side and LS protocol graphs are on the right hand side):

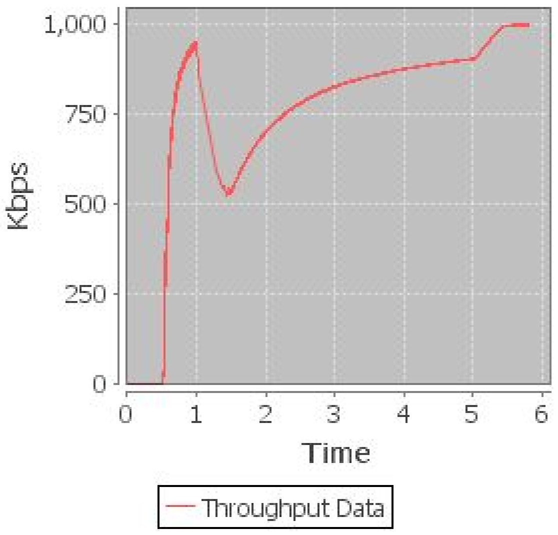
Dropped Data



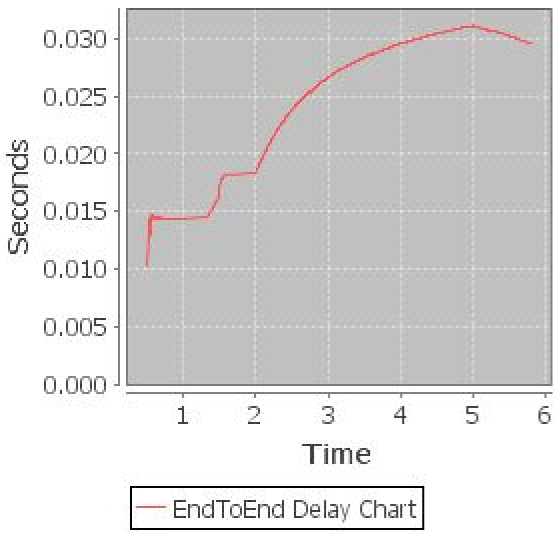


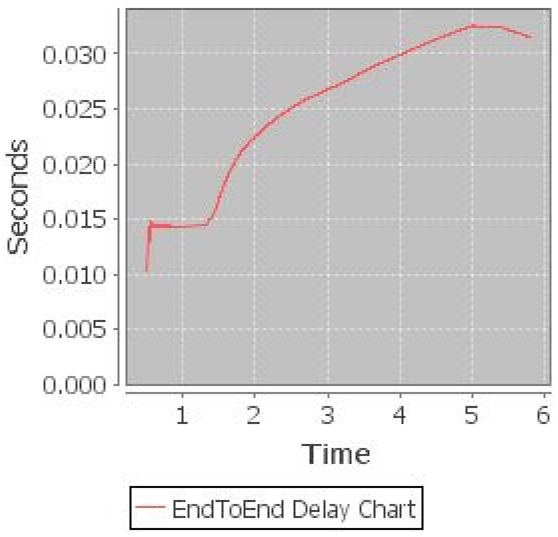
Throughput



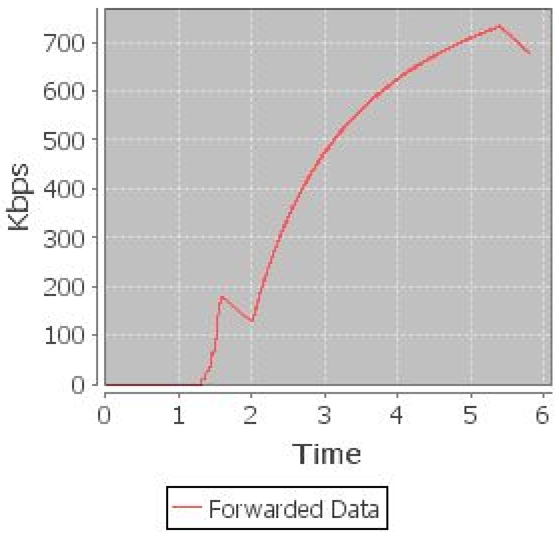


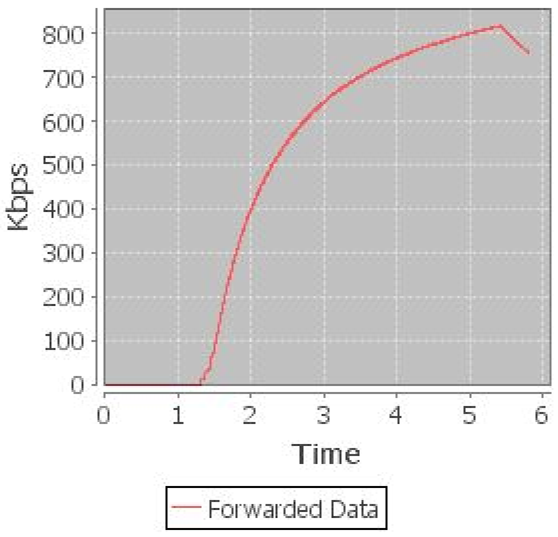
End-To-End Delay



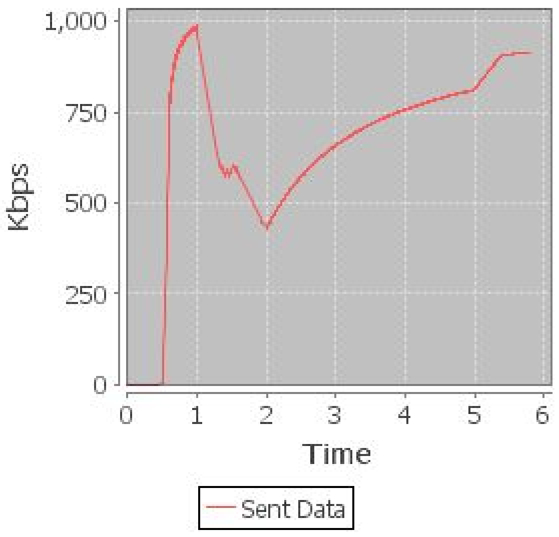


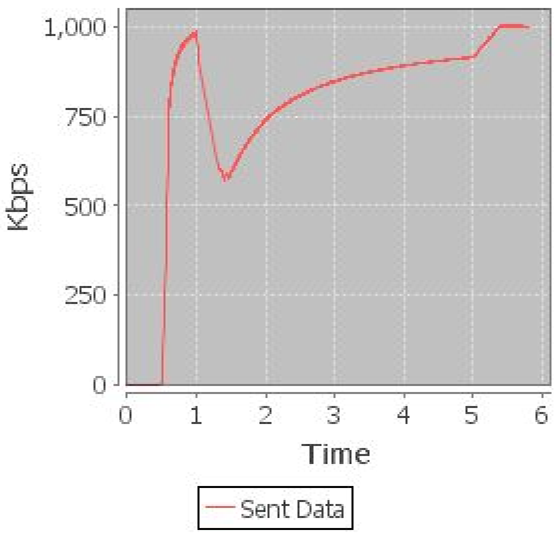
Forwarded Packets





Sent Packets



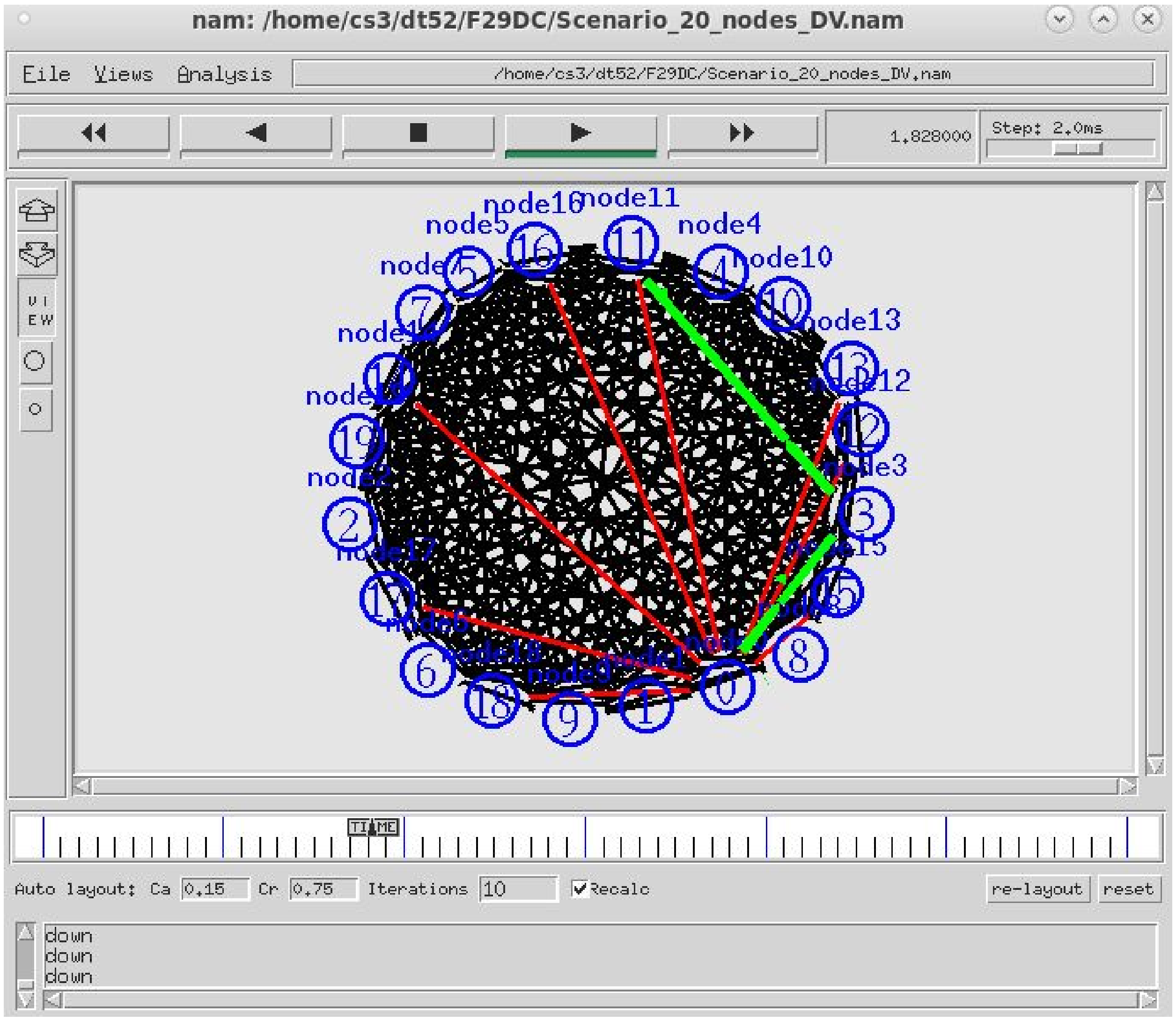


Upon looking at these graphs, the 2 protocols seem fairly similar when used on a 10 node network. The LS protocol fared slightly better due to dropping less packets as well as having a superior throughput whilst the DV protocol tended to have less data sent and forwarded as a result. The end-to-end delay may seem to favour the LS protocol however both protocols have similar delays, presumably due to the packets not having much to get in their way aside from dropped links.

Another notable feature is that in the DV graphs, there are more obvious spikes which make it easy to tell when a link went down in the network whilst the LS graphs there isn't anything as obvious, this is most likely due to the constant pings sent out to each node ensuring that data can immediately continue transferring to the target node.

After making these observations, it would seem that the LS protocol performs slightly better on a 10-node topology compared to the DV protocol.

20 node Comparisons (medium topologies)

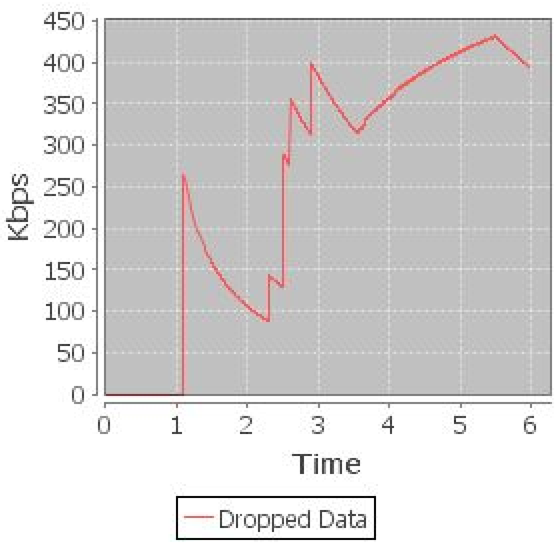

nam: /home/cs3/dt52/F29DC/Scenario_20_nodes_DV.nam
File Views Analysis
1.828000
Step: 2.0ms
TIME
node0 node1 node2 node3 node4 node5 node6 node7 node8 node9 node10 node11 node12 node13 node14 node15 node16 node17 node18 node19
re-layout reset

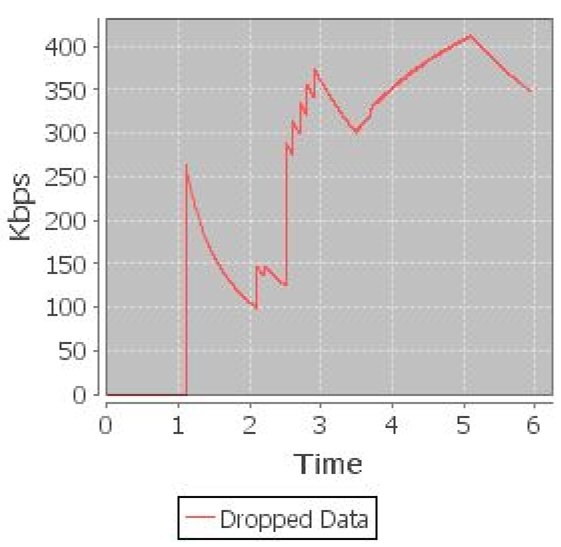
Auto layout: Ca 0.15 Cr 0.75 Iterations 10 ✓Recalc

```
down
down
down
```

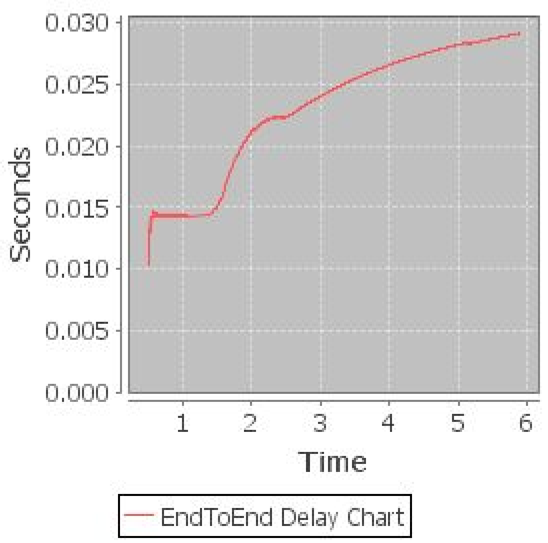
The LS topology is identical to the DV one seen in the above picture in appearance. Results from running the simulations are as follows (DV protocol graphs are on the left hand side and LS protocol graphs are on the right hand side):

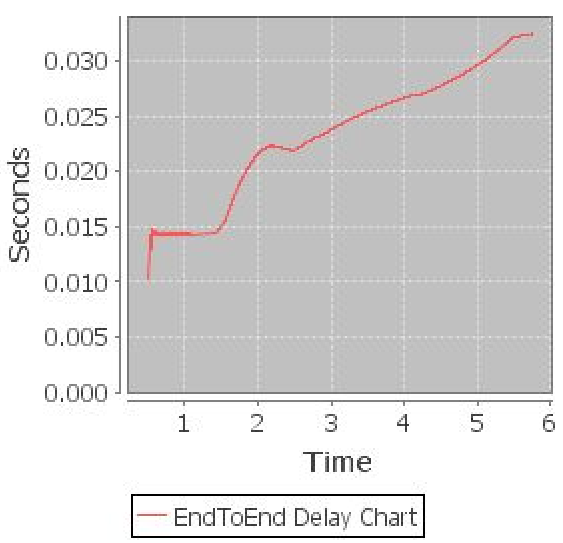
<u>Dropped Data</u>



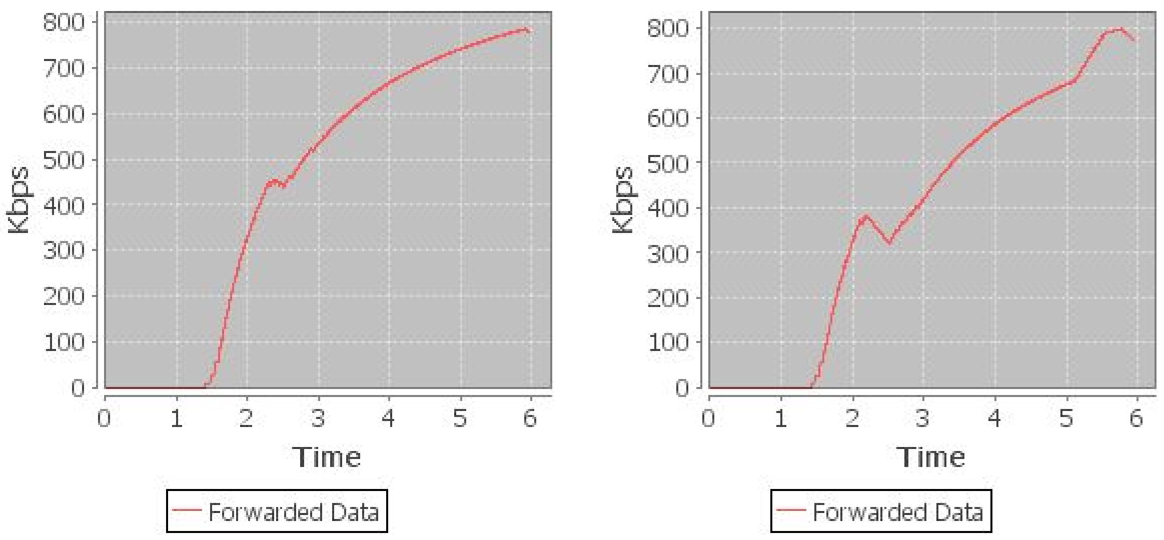


<u>End-to-End Delay</u>

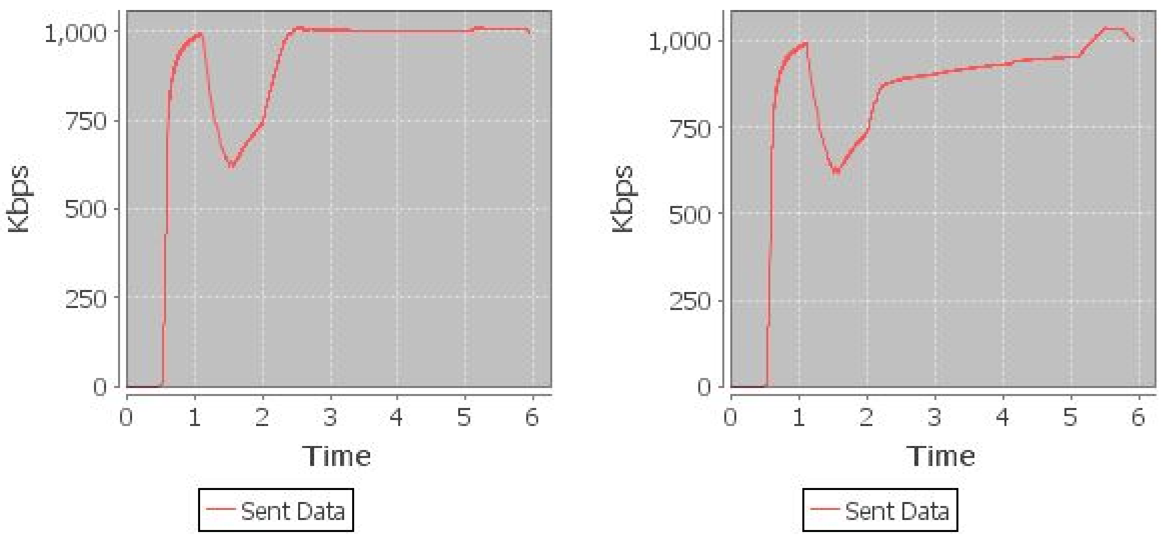




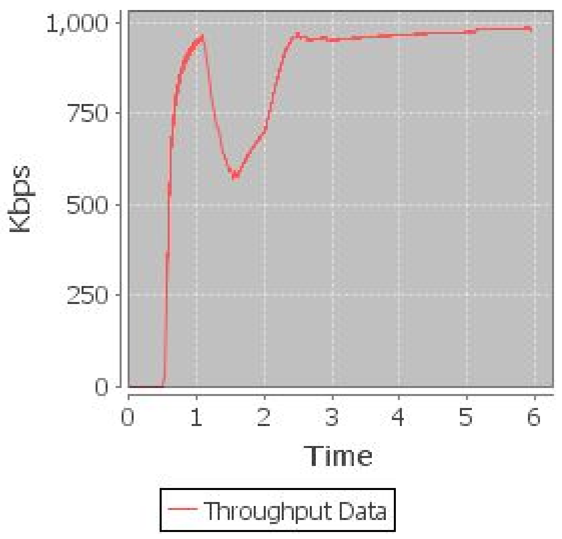
<u>Forwarded Data</u>

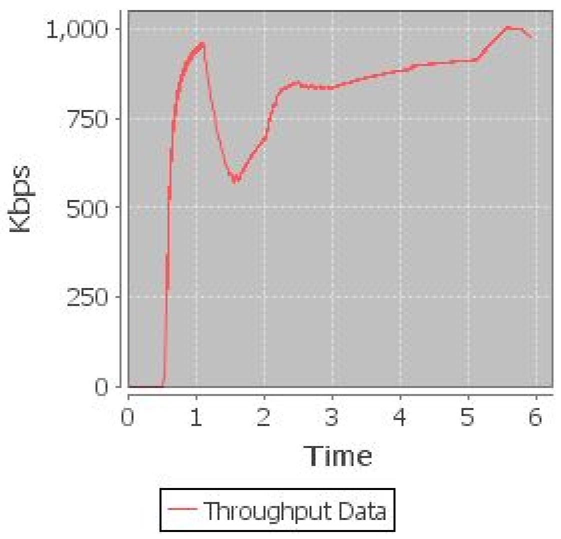


Sent Data



Throughput

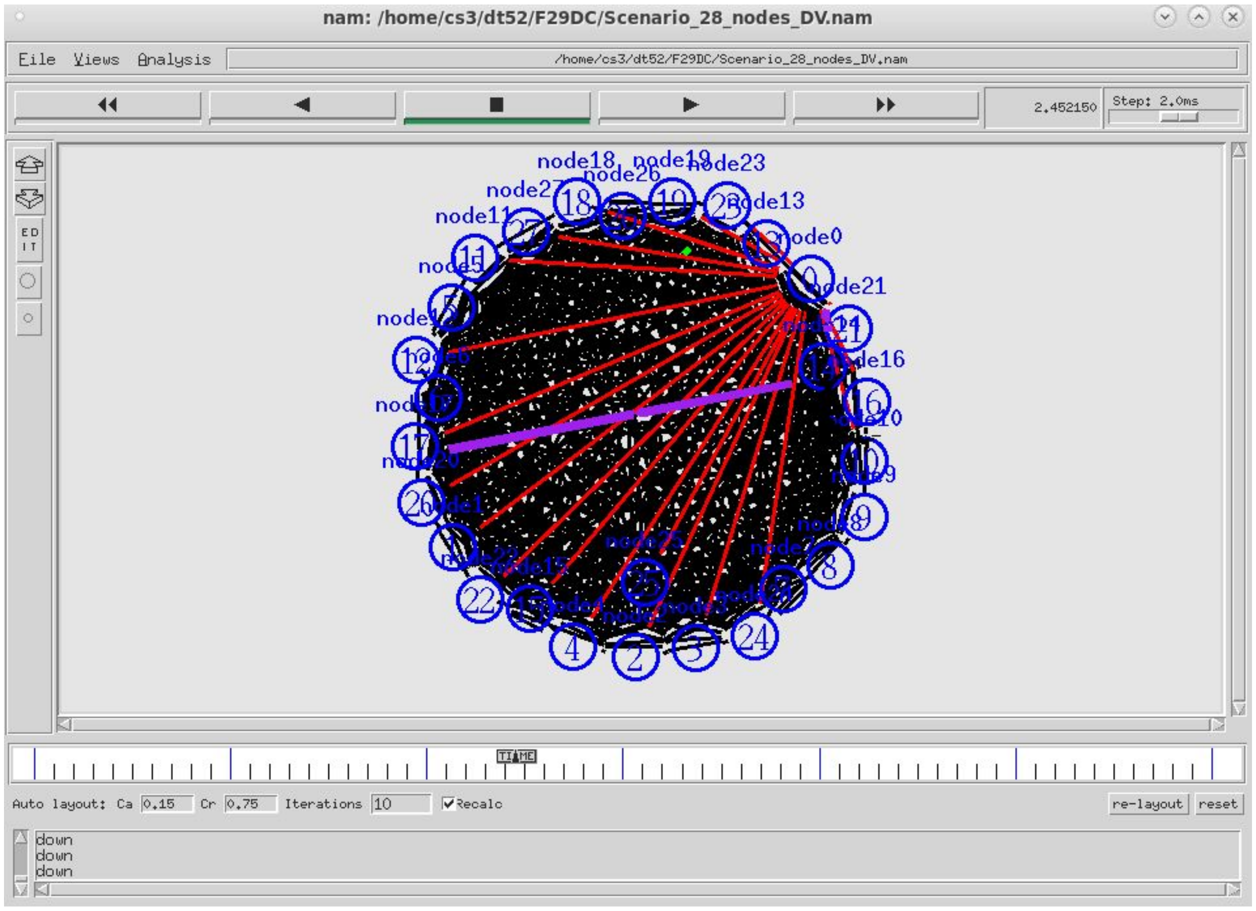




Compared to the 10 node topologies, the differences between the 2 protocols are a lot more distinct here. Whilst the LS protocol still seems to have resulted in less dropped packets overall, the end-to-end delay is even higher compared to the DV 20 node topology, presumably because of the larger amount of ping requests and having to take into account the larger number of nodes.

The DV protocol also seems much more efficient and consistent in forwarding and sending data than the LS protocol in addition to having a superior throughput. I believe that this is due to the LS protocols 20-node topology dropping more ping requests as seen in the respective topology animation, however it is definitely safe to say that the DV protocol is superior when handling medium-sized networks such as a 20-node network given the evidence above.

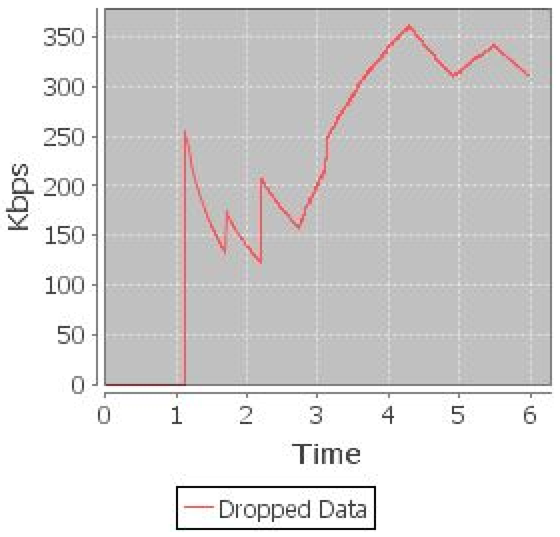
<u>28-node Comparisons (“large” topology)</u>

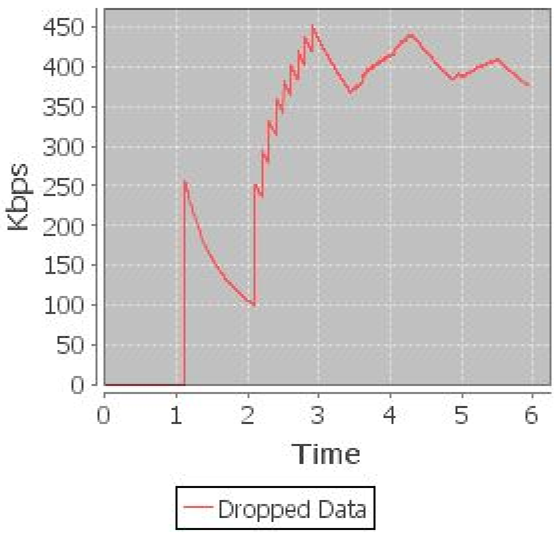

nam: /home/cs3/dt52/F29DC/Scenario_28_nodes_DV.nam
File Views Analysis
/home/cs3/dt52/F29DC/Scenario_28_nodes_DV.nam
2.452150
Step: 2.0ms
Auto layout: Ca 0.15 Cr 0.75 Iterations 10 Recalc
re-layout reset

```
down
down
down
```

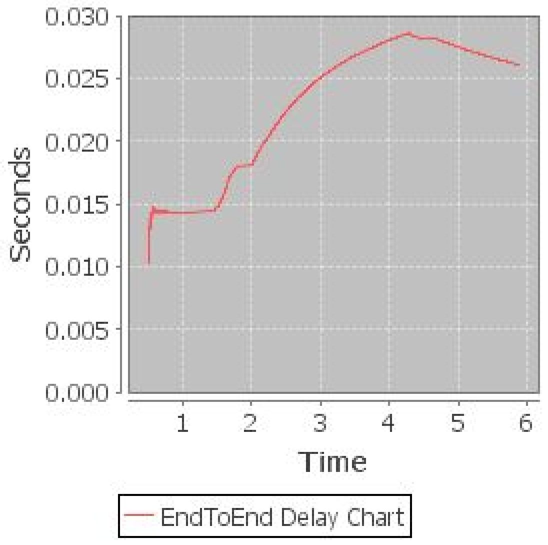
The LS topology is identical to the DV topology seen above in appearance. Simulation results are as follows (DV protocol graphs are on the left hand side and LS protocol graphs are on the right hand side):

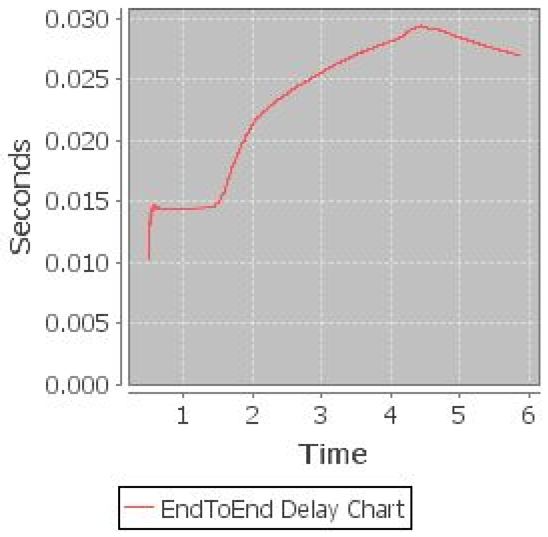
<u>Dropped Data</u>



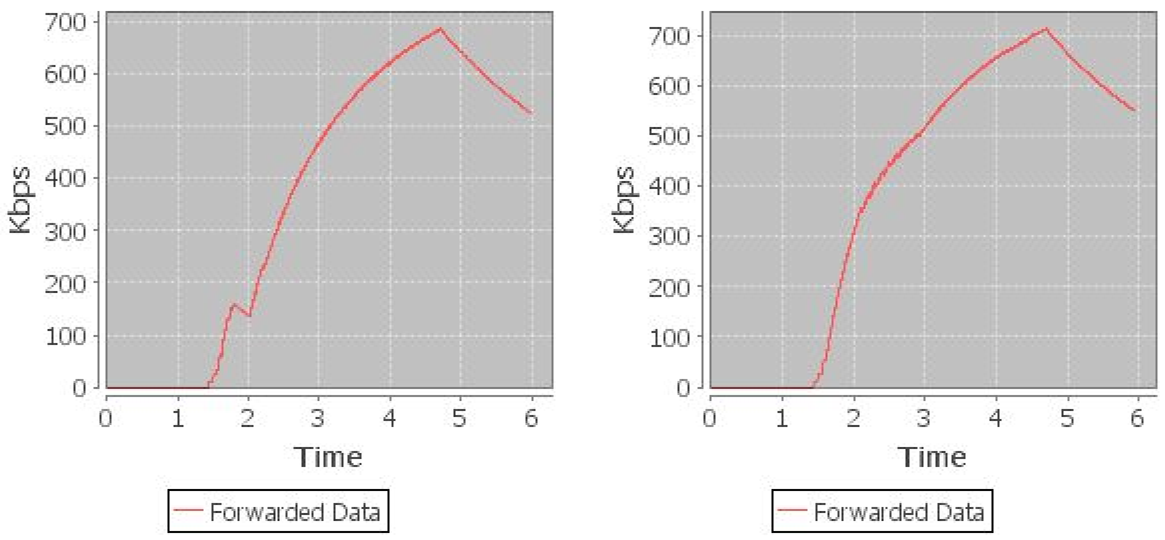


<u>End-to-End Delay</u>

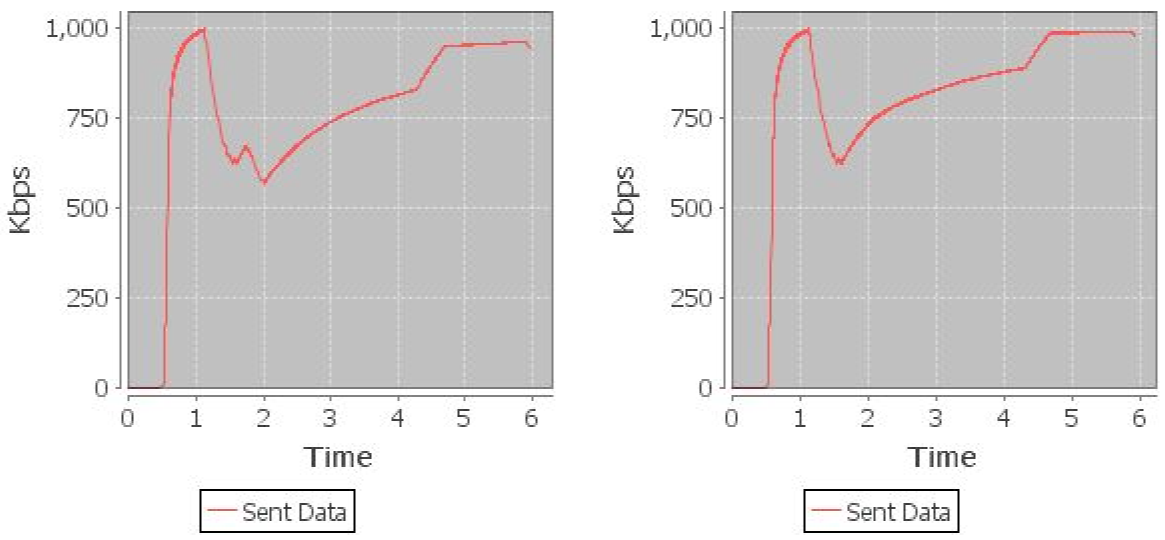




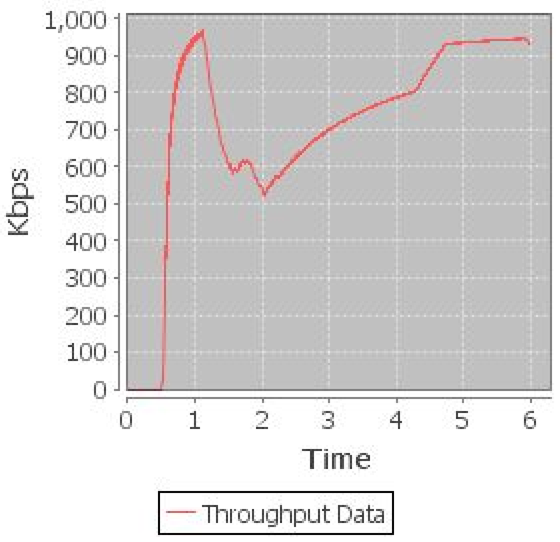
<u>Forwarded Data</u>

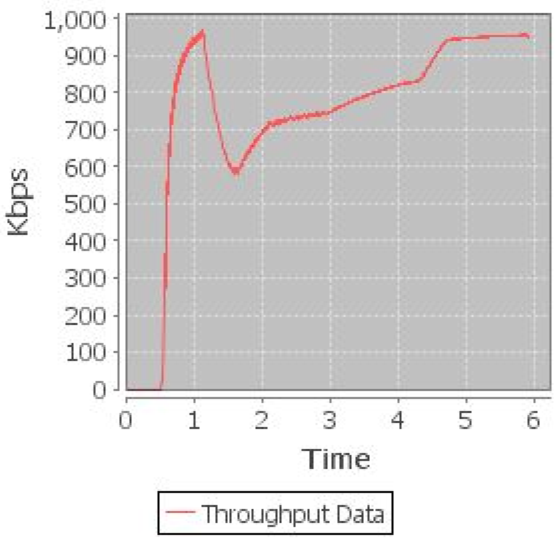


<u>Sent Data</u>



<u>Throughput</u>





Interestingly, the results are different to what I was expecting. I believed that the 28-node topology would provide more extreme variations to the 20-node topologies, but it seems as if the results are more balanced out, with the glaring exception of the LS protocol dropping a significantly larger portion of packets compared to the DV protocol peaking at just above 450 KBps of dropped data at 3 seconds of simulation time compared to the approximately 365 KBps of dropped data at approximately 4.25 seconds, which reflects very poorly on the LS protocol.

Aside from that, the LS protocol has a slightly better throughput than the DV protocol, presumably as a result of sending and forwarding more packets as seen in the above graphs. It also has a similar end-to-end delay when compared to the DV protocol here.

Whilst this seems positive for the LS protocol, the fact that it has so many dropped packets compared to the DV protocol which has a lot less (even with slightly less sent packets compared to the LS protocol) is inexcusable due to how the dropped packets could affect the communications in the network. Due to this, it is safe to say that the DV protocol is better for handling these large topologies.

Question 3 + 4 Conclusion

In summary, whilst the LS protocol may be better for handling small-scale networks such as the 10-node topologies outlined earlier due to its superior throughput and larger amount of data sent in said networks, the DV protocol definitely seems to be a more efficient and reliable choice for medium-large networks due to a lower risk of dropped packets and lower delay in receiving packets.