Model Worm Propagation

Project for ECE 6110: CAD for Communication Networks

Elza Mathew
Karan Kashyap
Ebin Mathews
Chandan Jyoti Sharma

Objective:

To create a worm model in ns-3, and perform a simulation study of the rate of spread of worms as a function of:

1. Scan rate.
2. No. of Connections.
3. Payload.
4. Bandwidth.
5. ICMP unreachable response enabled/disabled.

Design

1. <u>Topology (Random tree)</u>

We created a random tree topology with scalable number of nodes. For each node on each level, child nodes were assigned based on a simple probability comparison, that is a generated random number was less than a threshold, the child was created else not. The tree topology contained a variable number of two level trees which were all joined, through their home node, to a single infected node.

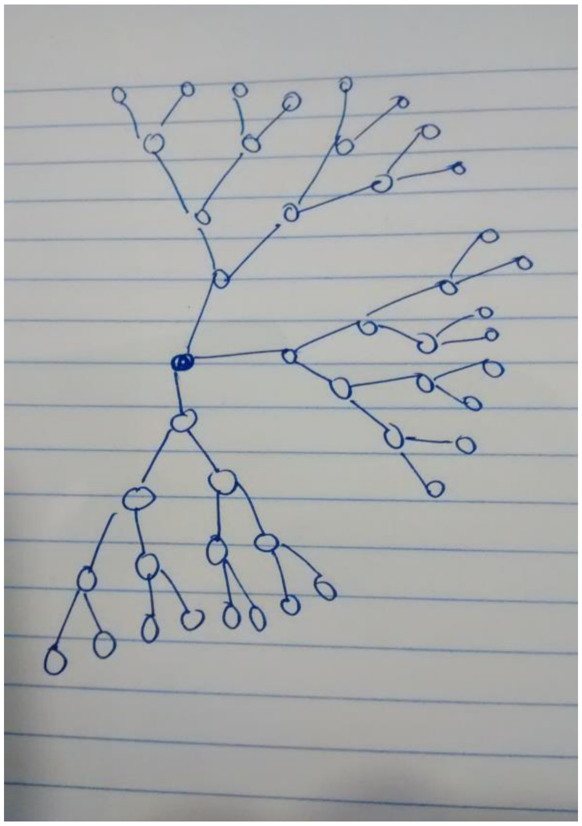


Figure 1: Basic tree topology

2. Assigning IP Addresses

The net devices connecting the root node to the head of each of the trees was assigned the base address of 1.0.0.0. The net devices connecting the head node of each tree to the first level of nodes in the tree was assigned the base address of 192.1+j.0.0 where j represents the node id of the corresponding first level node. The net devices connecting each first level node of the tree to the corresponding second level was assigned the base address of 192.1+j.x.0 where j represents the node id of the corresponding first level node and x was computed using the node id of the corresponding second level node.

3. The worm application:

The worm application was created by inheriting the Application class present in ns3. Most of the basic functionality of the Application class is implemented using private data members and member functions. These are not inherited in the derived class. So we had to implement these private functions on our own by using the code of ns3 Application class as reference.
The worm application also had a configurable parameter called m_nconnections, which specifies the number of destination that the application can connect to, and send packets simultaneously.
Another parameter called m_infected was used to enable transmission of worm packets in case it was set to true.

Setup

In the basic setup, a worm application was installed on the root node and it was set to be infected. The worm application and a packet sink were attached to each lead node. The initial configuration of these leaf nodes was set as uninfected. The worm application on the home node sent worm packets with respect to the assigned scan rate/connections to randomly generated IP addresses. Our aim was to infect only the second layer/ leaf nodes and if the worm packet reach any one of the leaf nodes, the worm application on the particular leaf node was activated and the worm application generates its own set of random IP addresses and send worm packets. The number of infected nodes were tracked using a counter. The color in NetAnim was changed from red to black to indicate the node was affected. A probabilistic vulnerability factor controls whether a node is affected when it receives a worm packet.

Flow:

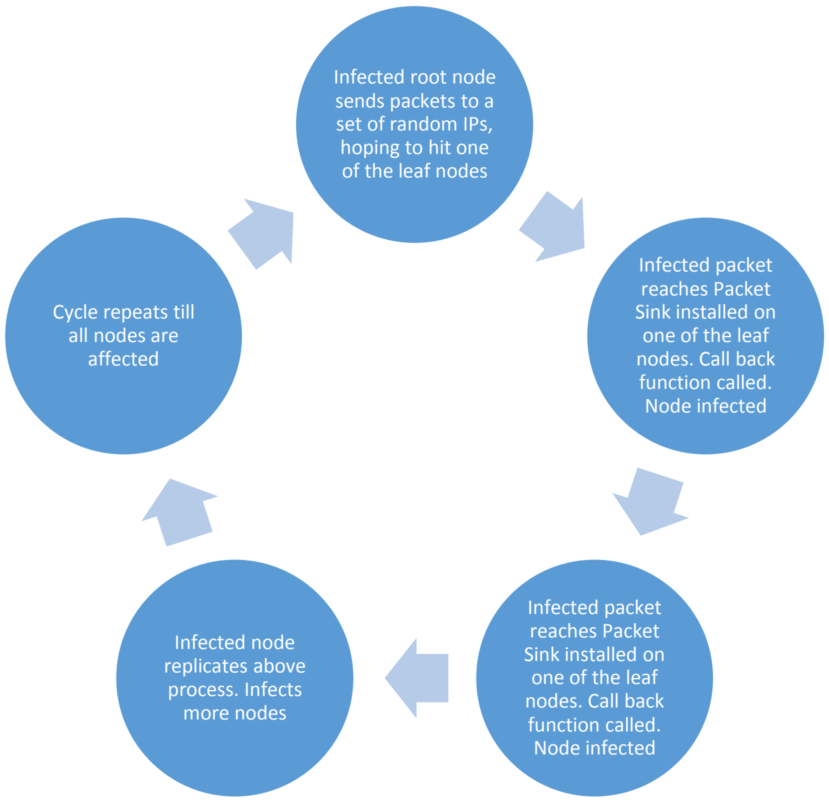

Infected root node sends packets to a set of random IPs, hoping to hit one of the leaf nodes
Infected packet reaches Packet Sink installed on one of the leaf nodes. Call back function called. Node infected
Infected packet reaches Packet Sink installed on one of the leaf nodes. Call back function called. Node infected
Infected node replicates above process. Infects more nodes
Cycle repeats till all nodes are affected

Figure 2: Flow diagram

Challenges and Solutions:

1. <u>Assigning IP addresses with no associated nodes</u>

   Since the Ipv4Helper class could not be used for our experiment, we had to think of an IP Address scheme and assign Ipv4 addresses manually to our created nodes.

2. <u>Dropping the packet in case of non-existent destination:</u>

   On sending a packet to a non-existent IP address, the packet was seen to shuttle back and forth at the last subnet hop till the number of retransmissions specified in the ns3 TCP protocol was completed. This caused unnecessary delay in the spread of the worm. So we decided to modify the default timeout period and the number of retransmission attempts This helped as to model the propagation of worm more accurately as we were running our simulations for 60 seconds and the number of created nodes were <500. So the packet was dropped as soon as the router found that the IP address does not exist. But we ensured that it would traverse the network till the last subnet hop to mirror real life networks.

3. Finding the ID of calling node of callback function

   The callback function is common for all the nodes of the network. But we needed to know which node's PacketSink called the function, so we had to modify the ns3 source files to extract this information. In the PacketSink.cc source file, we changed the payload that was being provided to the call back function. Inside the PacketSink, it was possible to call GetNode() to find Node ID. Then we wrote this ID to the packet which was being passed to the callback function as a parameter.
   Inside the callback function, we extracted the NodeID from the packet, and using this NodeID, it was possible to set this particular node to infected.

4. Disabling ICMP

   In the Icmpv4L4Protocol class, there is a portion where the route for ICMP message is generated. Setting this route to NULL achieved the goal of effectively disabling ICMP.

5. Changing node color

   Using the node ID returned in the call back function, we used AnimationInterface class's UpdateNodeColor() function to change the color of the infected node.

Experiments:

To model worm spread through a network, we conducted the following experiments:

1. Keeping track of the number of infected nodes with infection time with increasing number of connections for TCP, and increasing scan-rate for UDP
2. Keeping track of the number of infected nodes with infection time with increasing link bandwidth
3. Keeping track of the number of infected nodes with infection time with increasing payload.

Parameters:

1. The worm application data rate was set at 500kb/s
2. The simulation was run for 60 seconds.
3. All the links in the network have uniform bandwidth
4. No of TCP retries is set to 1
5. TCP timeout is set to 2 seconds
6. The number of leaf nodes was set to 200.

NetAnim implementation:

NetAnim screenshots with a representative toplogy containing 21 leaf nodes. It was very difficult to capture of the packets flowing in the actual topology with 200 nodes.

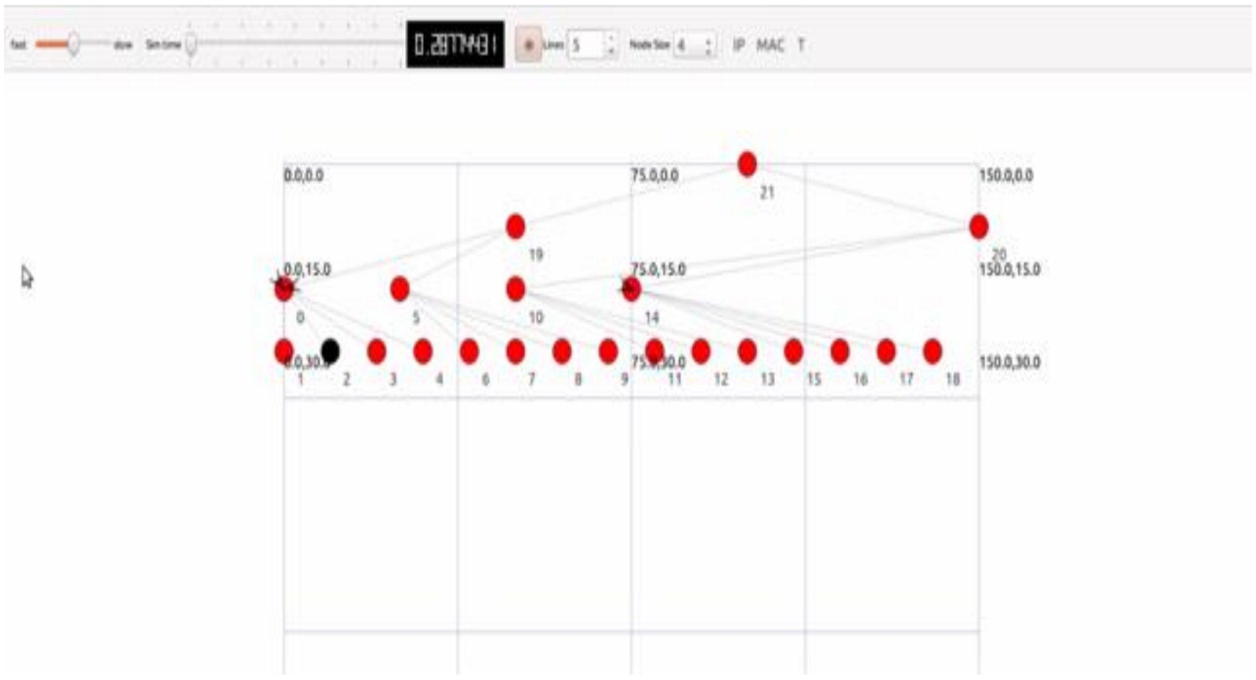


Figure 3: Some time into the simulation. Only one node (indicated in black) has been infected

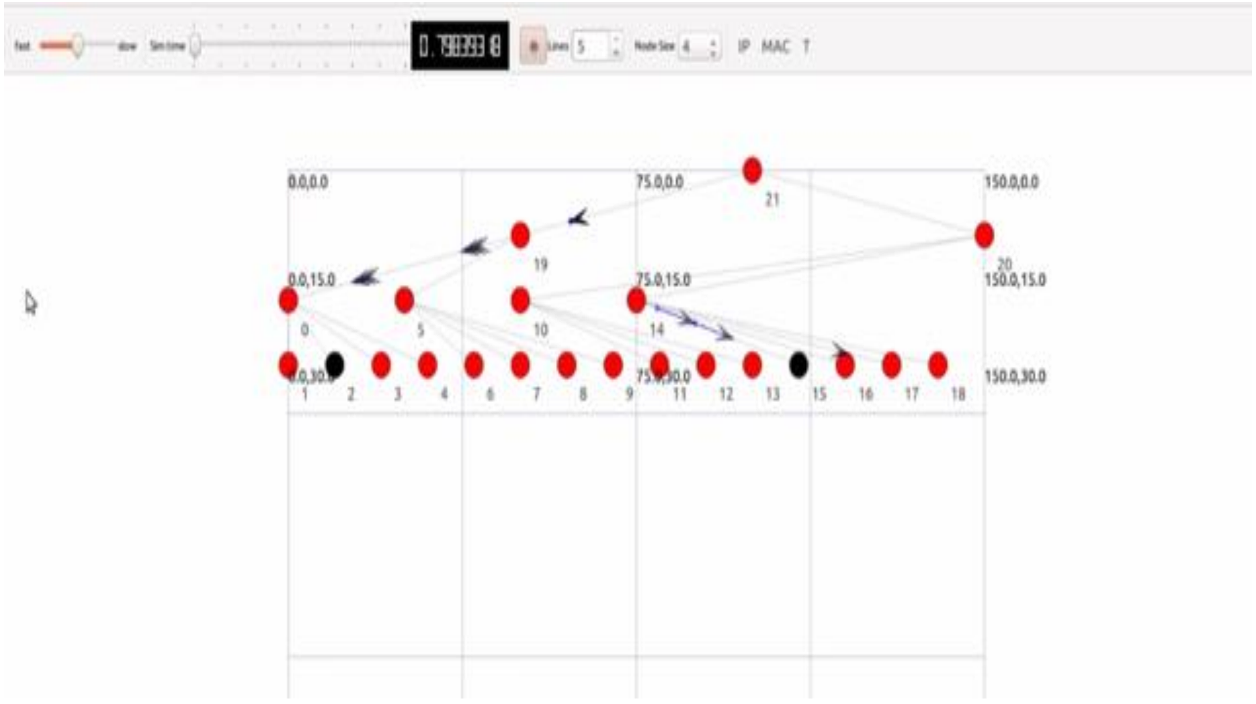


Figure 4: More nodes infected

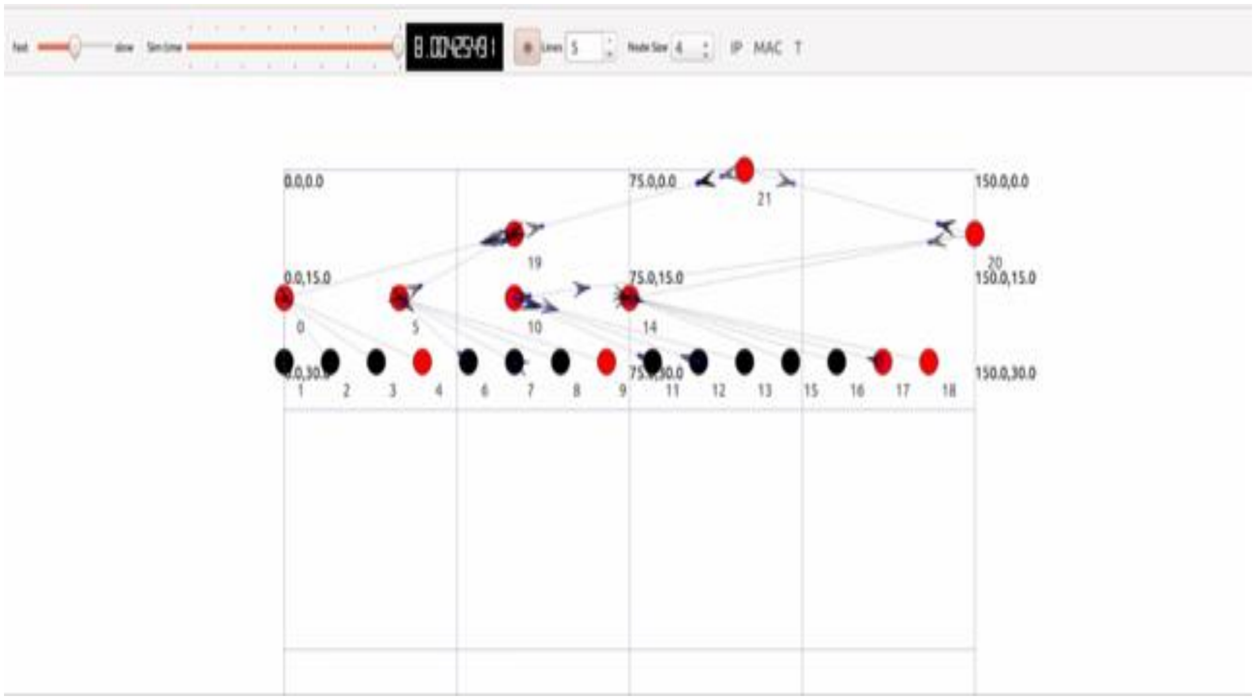
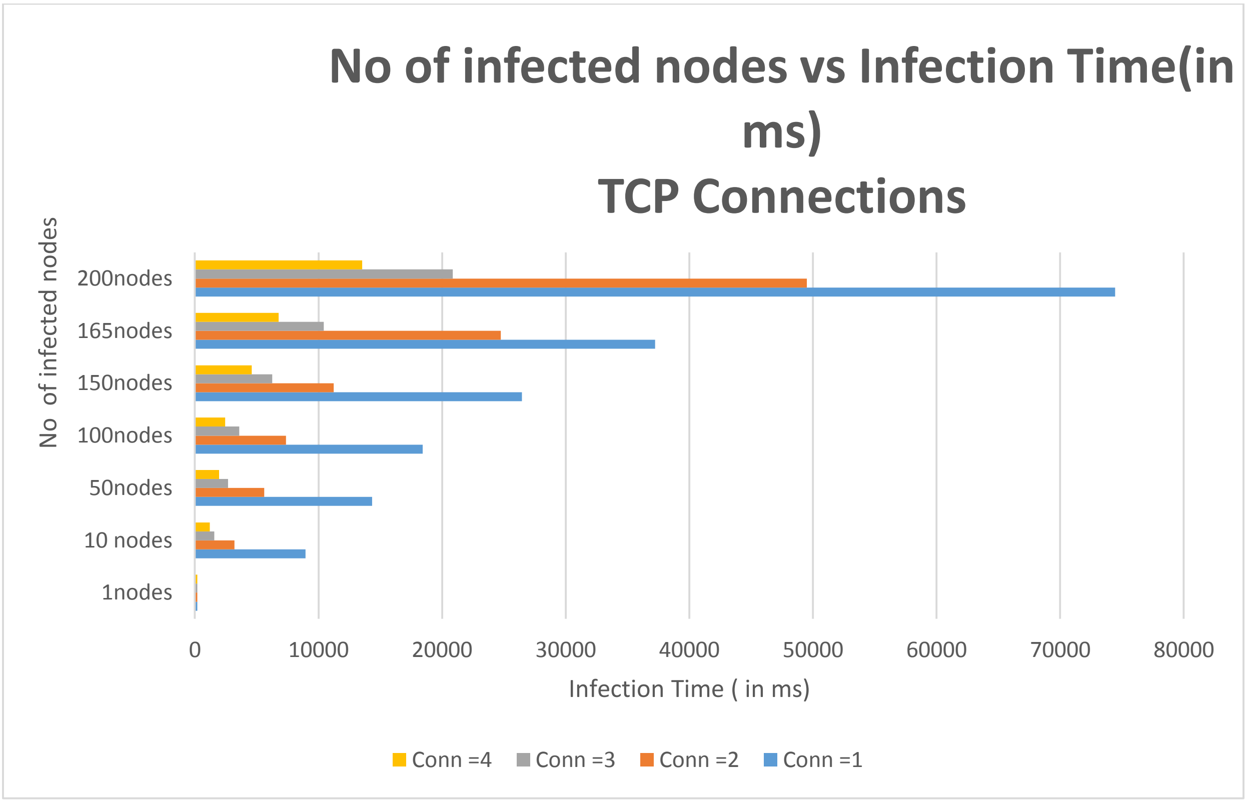


Figure 5: Towards the end of simulation, all vulnerable nodes have been infected.

Results and analysis:

1. <u>Varying number of connections in TCP</u>

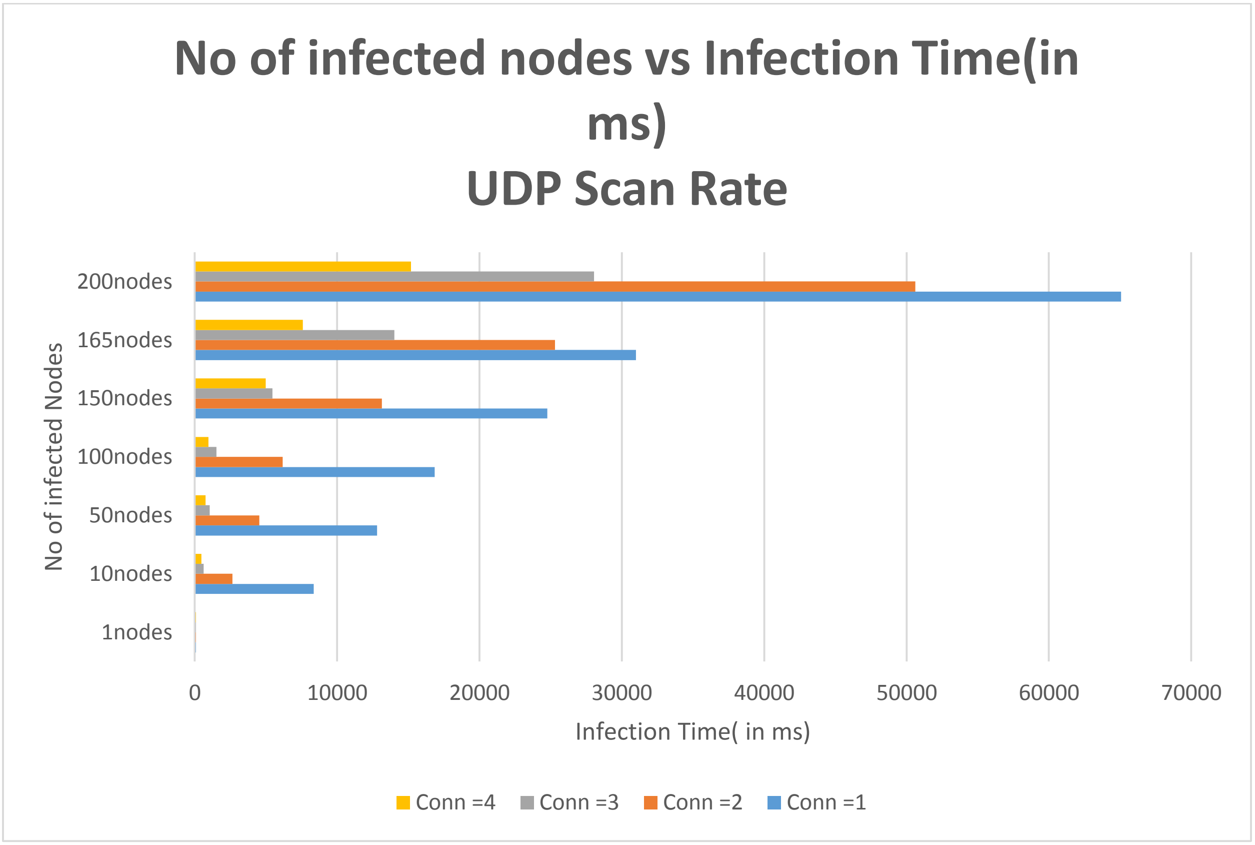


Since TCP connections involve a 3-way handshake for setting up, they exhibit a significantly less throughput if an IP address is non-existent. In this scenario, an intuitive way to increase the infection rate of the worm would be to have multi-threaded propagation through the network.

As can be seen from the graph, an increase in the number of connections increases the infection rate. This can be attributed to the fact that the probability of an infected packet being received by a node increases substantially with increase in the number

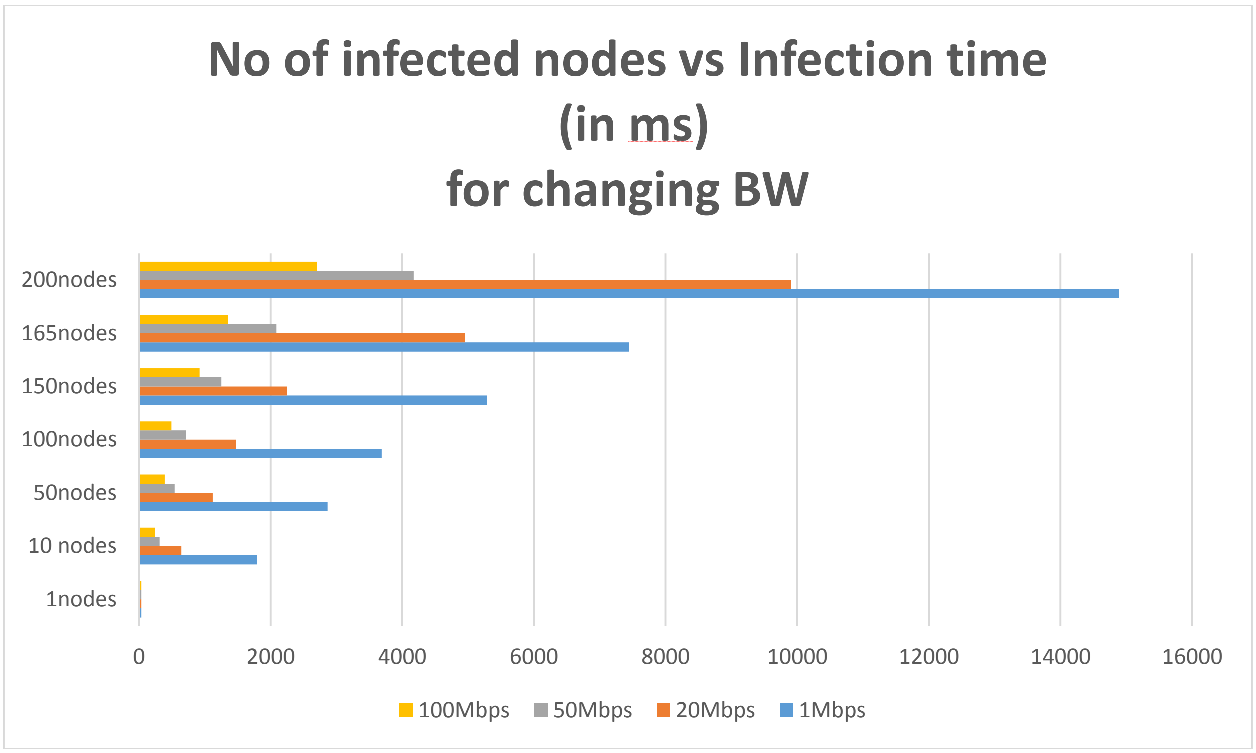
2. Varying scan rate

Scan rate is defined as the number of infection packets sent per second.



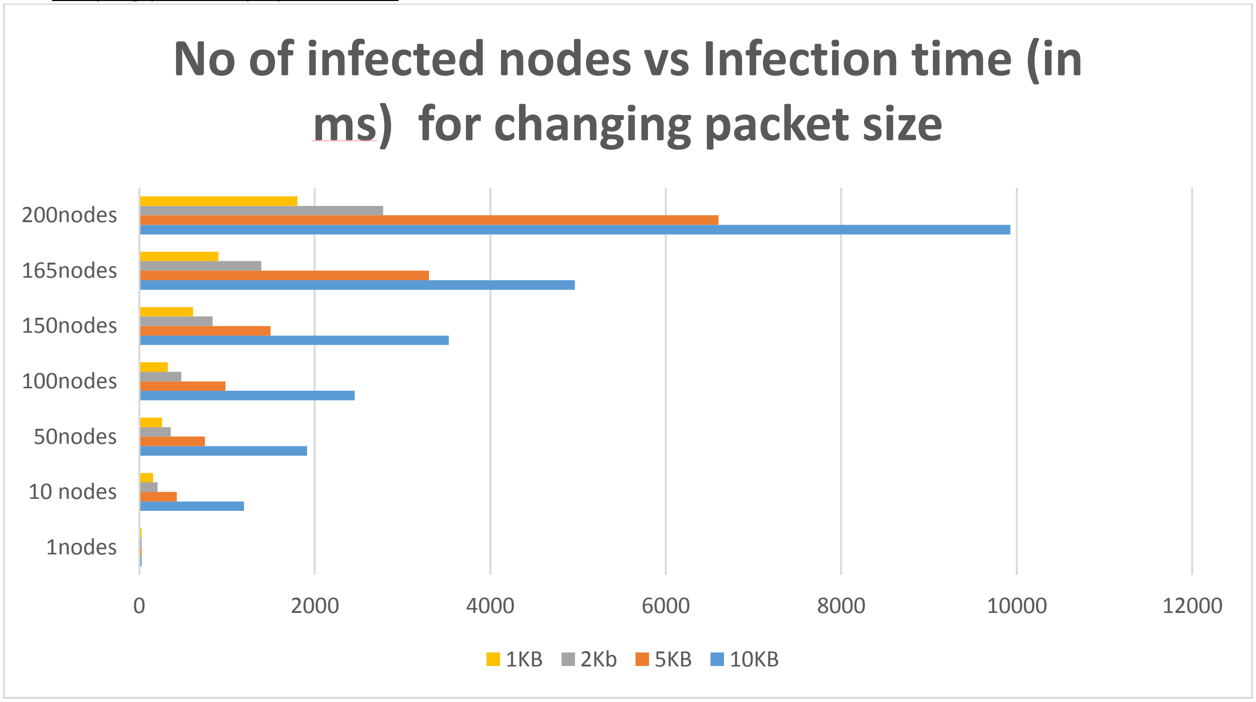
As the scan rate increases, the infection rate increases as there are no defense mechanisms in place. The higher the scan rate, the larger the number of connections attempted and higher the infection rate.

3. Varying link bandwidth



It is very intuitive that as the link bandwidth increases, more and more infection packets will be able to pass through the links. Therefore, the infection time for all the nodes will be significantly less.

4. Varying packet (payload) size



As packet sizes are varied from small to large, it is seen that the infection time actually increases. This is because small packets involve far lesser forwarding since their round trip time is less. Hence, they infect the topology faster.

5. <u>Varying ICMP messages</u>

An ICMP message indicates that a particular destination is not present in the current network. This information can be used by a learning worm by keeping track of all the un-reachable addresses. However in our case, we did not have enough time to do experiments on the ICMP packet and extract various kinds of information from it. Hence, the worm we built was not intelligent and ignored ICMP messages anway. So the in our experiments, we found that disabling the ICMP packet did not have any effect on the worm spread rate.

Conclusion:

UDP performs better than TCP as it doesn't require an initial three way handshake and can keep on sending packets even if the random IP generated is non-existent. UDP worms do not wait for an acknowledgment and thus are largely independent of round trip times. On the other hand, in case of TCP if the random is nonexistent then the worm has to wait for a timeout to send the next worm packet decreasing the infection rate. Increasing the scan rate for UDP and increasing the number of parallel threads for TCP can increase the infection rate. Increasing the bandwidth of the network links and decreasing the payload also has the same effect on the rate of spread of worms.

Future Work:

1. NIX vector routing can be used to decrease the simulation time.
2. To improve the speed of simulation, features like Routing Proxies and Breadth First Search can be incorporated.

Though we managed to get the basic topology and WormApp up and running, we ran short of time and were unable to implement these two modifications that would have speeded up our simulation.

Acknowledgements:

We would like to thank Professor George Riley for all his guidance during the implementation of this project.

References:

Simulating Internet Worms: George F Riley, Monirul I. Sharif, Wenke Lee