Introduction: This project is based on the second project where you implemented a virtual network infrastructure. If you are not happy with your implementation, a standard implementation from someone else will be made available to you after you hand in project 2. There is a required part to this project (implementing either multicast or a reliable transport protocol) and then some optional parts that can be done for bonus credit or to get out of taking the final.

1 Part I: Supporting Multicast

Note: Only one of Part I and II required.

In this part you will support multicast in your overlay. With multicast, we will have one or more multicast groups, each with several members. When you send to the group as destination, all the members in the group should receive the packet. You can implement any multicast algorithm that builds a spanning tree (i.e., anything but the brute force methods such as unicast to all group members or broadcast to everyone). The easiest would probably be Core based Tree which is equivalent to sparse mode PIM. In addition, for one of the bonus parts, you will need Core Based Trees. I’ll describe how you could go about implementing that.

In the configuration file, add the groups which a node is member of at the end of its line. These groups could have id of 1000 or more so that you do not mistake them for regular connections. For example, the line for node 1 may become:

1 bingsuns.binghamton.edu 7000 7001 3 4 1001 1002

to say that its connected to 3 and 4 and a member of groups 1001 and 1002. You may assume that the RP for a group is known (for example, by taking some hash of the group number such as groupNumber mod totalNodes).

You should have a multicast forwarding tree that includes an entry for every multicast group. Recall that an intermediate node may have to forward a multicast packet to multiple outgoing links; the multicast forwarding table entry should have for each group we are a member of the set of links it should forward the packet on. That is, there are multiple next hops possibly for each multicast entry. So, your data structure should have a way to include multiple hops (a linked list, or a fixed number array of next hops).

Each next hop should have its own TTL entry. The TTL is set whenever a join request comes in on that link. It is decremented with every heartbeat.

Joining and Leaving: Periodically, each node should send a join unicast message to the RP for each of its groups. As the join packet is forwarded towards the RP, at each intermediate node it is processed this way:

1- If node is already part of the tree: when we receive a packet for a group we are already part of the tree for, we add the receiving link to forwarding table entry for this multicast group. Also, we generate an ACK back to the source of the join to indicate that the join was successful.

2- If node is not part of the tree, it adds itself to it by creating a forwarding table entry for the group, with one outgoing link (the link it received the packet from). The node should forward the join packet towards the RP.
Leaving is implemented by stopping the send of the Join message. When TTL expires at intermediate nodes, the node will be removed from the multicast tree automatically. So, really you don’t have to do anything for leaving.

**Forwarding multicast packets:** Assume that only multicast members can send a packet to the group. When a node sends a packet to the group, the packet should be flooded in the tree. This is done by sending the packet out on all the next hops for the multicast tree other than the one that the packet is received on. So, when you receive a multicast packet from neighbor X, walk the next hop list for this multicast group and send the packet out on every hop except the one leading back to X.

2 Part II: Implement a Reliable end-to-end protocol

Note that only one of part I and part II is required.

In this part of the assignment you have to implement a reliable end to end protocol. What you implemented so far is the equivalent of IP. Recall that an end-to-end protocol provides a process to process connectivity. This is done by demultiplexing from network protocol (what you implemented last project) to the appropriate protocol using a protocol number field in the header (don’t forget to remove the network header). Once your end to end protocol receives the packet, it should demux to the appropriate “socket” using a port number field. To build an end to end protocol on top of it, you have to first include a transport protocol field in packet header and demux to the appropriate protocol. For each protocol you support, have a limited size circular buffer that you place packets in. You have to implement a producer consumer circular buffer object (or data structure) that is safely protected using locks. A circular buffer is simply a fixed size buffer. Most Operating Systems textbooks have examples of how to implement that.

When a packet is received by the forwarding thread that is destined for this node, instead of discarding it as we did before, we should now decapsulate it and pass it to the end to end protocol. This is done by taking the data part of the packet (i.e., removing the header for the network protocol) and placing it at the end of the buffer for the protocol (which specified as the demux key in the network protocol header). If the buffer is full, throw the packet away. If the buffer does not exist, this means that the protocol is not supported and you can just drop the packet (this case should not really happen here).

You can implement a transport protocol as a separate thread (Its enough to implement a single transport protocol.) The thread reads packets from its buffer and processes them. The data part of the original packet which is placed by the forwarding thread in the protocol buffer includes the header for your transport protocol. At a minimum this should include the information in the UDP header (check the book for that). This includes source and destination node id and port numbers. Use the destination port number to demux to the appropriate socket buffer. Again, this is a circular buffer that is written to by the transport protocol and read by the application thread that owns this socket.

To implement reliability, you should have some form of sliding window algorithm running. The flavor of the sliding window algorithm you want to implement is up to you. You have to generate acknowledgements for correctly received packets. You have to figure out how to implement timers and retransmissions. Stop-and-wait at a minimum is expected. You can do Go-back-N or Selective Repeat for extra credit. For timers, you can use software timers – that is, set the retransmit timer to be some multiple of the heartbeats of the transport thread. Every time it wakes up it decrements the timer. If it becomes 0, retransmission is initiated. If an ACK is received, the packet is freed and the retransmit timer cancelled.

You should create one or more application threads that then send packets to each other. You can just hard code the ports they are running on for now, but you should clearly show that you can run multiple connections from the same host and not get them mixed up.
3 Part III: Bonus Parts

I’ll list several ideas here and how much credit they will be worth. To get out of the final, you have to do 60 points worth of extra credits. If you do both parts above, this is enough to get out of the final. Or you can do any combination of 60 points from below. You can also count the bonus parts from the previous project. Come talk to me if you need help getting started:

- 30 points: Implement flow control and in order delivery in your transport protocol
- 30 points: Implement Go-Back-N or Selective Repeat for transport protocol
- 30 points Implement functionality similar to the specialized trees in PIM. That is, if there is an active source in the multicast group, you can dynamically construct a tree specialized for it as per the PIM algorithm.
- 15 points: implement applications that use your transport protocol. Each is worth 15 points. e.g., chat, ftp ...
- 30 points: Implement a DNS service for your overlay network
- 30 points: Implement a peer-to-peer lookup service for your overlay network. That is given the name of a document, you should be able to find out what node it resides on