Problem 1:
Here are the results for the ping and traffic generator applications when client ran on sslab01, router 1 on sslab02, router2 on borg00, and server on borg01.
+ For my the app:

Test	Elapsed time on overlay network	No overaly network
1	1.29 ms	0.72 ms
2	1.50 ms	0.72 ms
3	1.56 ms	0.71 ms
4	1.49 ms	0.79 ms
5	1.54 ms	0.78 ms

As we can see that the elapsed time on overlay network is larger. This is reasonable since in overlay network the path are longer, it has to go through intermididate routers to reach the destination.

+ For traffic generator app:

Test	Throughput on overlay network	No overaly network
1	868 pps	871 pps
2	867 pps	866 pps
3	879 pps	871 pps

The throughputs at receiver on both cases are almost equal. In this problem, no matter what path the packets take, the receiver still achieves the same throughput.

Problem 2:
Here is the design for my UDP-cased reliable file transfer. To make UDP reliable, I used window sliding algortihms. I maintain follwing variables:
+ SWS : sender window size
+ LAR: last acknowledged frame
+ LFS: last frame sent
+ RWS: receiver window size
+ LAF : largest acceptable frame
+ LFR: last frame received
+ Each sender and receiver has a buffer of lenth SWS, RWS respectively
In my implementation, I maintans 2 variants: LFS – LAR <= SWS, and LAF – LFR <= WRS.
When sender (sever) receives an ACK with sequen number X, he moves LAR to X, flushes buffer upto LAR, sends up to (SWS – LFS + LAR) frames, and updates LFS.
When receiver receive data with sequence number Y, he updates LFR, sends cumulative ACK (i.e, LFR+1) , flushes buffer to file, and updates LAF = LFR + RWS.
Also, I used negative ACK to make the performance better. Whenever receiving a negative ACK, sender retransmits the packet. Further more, the sender is intially in slow start with a cogestion control of 1. For every valid acknowledgement received, it increases window size by 1, which is 1000 bytes (block size) . Thus, for the first ACK, window grows from 1 to 2 and 2 packets are sent in the next iteration. Both the acks will be received in the same RTT initially, so the window grows from 2 to 4, effectively doubling each time. This continues till the server either times out on an acknowledgement or the SS threshold is hit. Whenever the server times out or gets 3 consecutive duplicate acknowledgements, it moves from slow start to congestion avoidance phase.
In this phase, the congestion window grows linearly rather than exponentially.

The payload of packets from server now also contains 4 bytes sequence number. The receiver sends eitther cumulative ACK nor NAK back to server.
However, I did not completed the implementation for the design since I had some troubles and time limited.