Lab 6

Duc Viet Le
CS536

December 6, 2016

Problem 1.
The result is similar to the result of lab4:

- For `myping.c`: since it takes longer for udp packets to travel between overlay nodes, the ping result increases as I increase the number of overlay nodes. I used 3 overlay routers which are `sslab02, sstlab04, sslab06`, and I run `mypingd.c` at `sslab01` and `myping.c` at `sslab08`. The ping results are:

  ```
  1.997 ms
  2.225 ms
  1.792 ms
  2.132 ms
  2.023 ms
  1.989 ms
  ```

  Ping result without using overlay router is: `0.447 ms` on average.

- For `traffic_send.c`: Since we test the app on multiple sslabs with pretty much bandwidth, there is no significant differences when using overlay routers. However, if there is a bottle neck node in one of the overlay node, I would expect the bandwidth to be decrease. Below is result using same overlay nodes (i.e `sslab02, sslab04, sslab06`) `traffic_snd` :

  ```
  Portnumber: 21806
  payloadSize: 1000
  Package Count: 1000
  Package Spacing: 1000
  Completion Time: 1.132008 s
  Package Per Second (PPS): 883.386047 packages/s
  Bit sent: 8440000
  Bits Per Second (BPS): 7455778.000000 bps
  ```

```
traffic_rcv

        Port Number: 30000
        payloadSize: 1000
        Start listening ...
        First Package arrived.
        End of transmission
        Package Count: 1000
        Completion Time: 1.133577 s
        Bits received: 8440000
        Package Per Second (PPS): 883.722412 packages/s
```

- For testing multiple clients, I do not see any significant changes.

Problem 2.
Sliding Window Algorithm and *Negative/Cumulative Acknowledgment* are used for this problem. For each UDP packet, I used extra 4 bytes as sequence number. Similar to what discussed in lectures, each sender and receiver keep track of 3 variables. For sender, those are *send window size* (SWS), *Last Acknowledgment Received* (LAR), and `Last Frame Sent` (LFS). Similarly for receiver, those are *largest acceptable frame* (LAF), *Last Frame Receive* (LFR), and `Receiver Window Size` (LFS).
When a frame arrive at receiver, the `SequenceNum` is checked if it's between *LFR* and *LAF* and is discarded if otherwise. An cumulative ACK is sent to sender when all frame in between window are received, so that sender can update its sliding window. Also, when receive detects a missing packet in between a negative ACK is sent to sender, the sender then resends the missing package and wait until receiving cumulative ACK from receiver. For simplicity, we assume there is no corrupt package since performing checksum may significantly reduce running time.

Problem 3.
Greedy-TCP: For this problem, the tcp sender can reduce package spacing by half or double/triple its sending rate every RTT to take all available bandwidth (i.e. similar to slow start discuss in class). However, to avoid self-congestion, the TCP sender only does that until its receiving the first package loss; the tcp sender will reduce it sending rate by a fraction. We can adopt the equation discussed in class as follow:

$$\begin{aligned} &if\ Q(t) = Q^*\ then\ \lambda(t+1) \leftarrow \lambda(t) \\ &if\ Q(t) < Q\ then\ \lambda(t+1) \leftarrow \lambda(t) \times \alpha \\ &if\ Q(t) > Q\ then\ \lambda(t+1) \leftarrow \gamma \times \lambda(t) \end{aligned}$$

Where $\alpha > 1$ (i.e $\alpha = 2$) and $0 < \gamma < 1$ (i.e. $\gamma = .5$)