CSCI 558L Fall 2014
Laboratory 6: Fast Reliable File Transfer over TCP/IP

Instructor: Young H. Cho – younghch@usc.edu
T.A.: Siddharth Bhargav – ssbharga@usc.edu

1. Improving TCP Performance Over Lossy Links

In this course we have examined the performance of TCP and UDP under several scenarios. In the earlier file transfer laboratory we developed a specific solution to transferring data over a high-latency lossy link. In this lab, we want to explore generic solutions to improve TCP performance so that any TCP application can take advantage of your solution. You will present the results of the implementation to the TA.

1.1. Congestion vs. Lossy Links.

The acknowledgement and back-off mechanisms of TCP were developed to avoid congestion collapse [2]. However, in general, mechanisms that are in use today for TCP cannot tell the difference between a packet lost due to congestion and a packet lost due to an unreliable link. The exponential back-off algorithm is the appropriate algorithm to deploy when TCP flows are sharing a congested link, however, as you have seen in the prior lab, reducing the sending rate and window may not be the proper response under lossy-link conditions.

1.2. Experimental Set-up

Nodes n0 and n1 need to exchange data using TCP. The nodes are connected by a high latency, lossy link (such as satellite or long range wireless uplink).

2. Initial Experiments

The goal for this lab is recognize that the packet drops in this network are from a lossy link and not from congestion, and thus improve TCP performance. The test for this lab will be to use the standard FTP program to move a 1GB (1024 byte) file from n0 to n1.

Begin by writing a NS file for this experiment. You may use any OS you'd like, but you may need to consider how you'll go about implementing the changes (see below for more details). Set the link between the routers to 100Mbps with an initial delay of 10ms with no loss. Create a local filesystem using mketrafs and a 1GB file. Use SCP between n1 and n1 to transfer the 1GB file. Record the performance obtained for your report.

Explore the problem space by varying the delay and loss up to 200ms delay and 25% loss (both directions, so 400ms RTT and 50% total packet loss). What throughput can FTP achieve under these conditions? You may want to present this as a 3D graph, or one graph with multiple lines.

3. Improving TCP Performance

Now we need to consider ways to improve the performance of TCP under these conditions. The easiest way to modify or implement your own TCP stack is to do it on Linux. There already exist a number of plug-able TCP implementations for Linux [3]. In order to use them you'll need to figure out how to

build a new kernel on your chosen Linux distribution and include these modules. It should then be fairly simple to modify the modules to improve performance.

We suggest that you start early and break the group into two subgroups to work in parallel given the short deadline as well as the limited resource. One way or another, the goal of this lab is for the whole team to understand both aspects of the solution.

Modifying TCP Module

Start by reading the paper "Removing Exponential Backoff from TCP" [1]. Start with the standard TCP stack for Linux and remove the exponential backoff algorithm as described in the paper. Make sure to understand the implicit packet conservation principal. For the full credit, you will need to achieve at least 10 Mbps performance of FTP over TCP/IP from n0 to n1 over 100 Mbps/200ms delay/20% packet loss.

4. Evaluation

To evaluate your improvements to TCP perform the same experiment as above with SCP, except include your modified TCP stack and/or router. If you do more than one modification, make sure to test and evaluate each modification separately (and together) to see which change improves performance under what conditions.

5. References

[1] Amit Mondal and Aleksandar Kuzmanovic. Removing exponential backoff from tcp. SIGCOMM Comput. Commun. Rev. , 38:17–28, September 2008.

[2] John Nagle. Congestion control in ip/tcp internetworks. SIGCOMM Comput. Commun. Rev. , 14:11–17, October 1984.

[3] Ren´e Pfeiffer. Tcp and linux' pluggable congestion control algorithms.