EE 542 – Laboratory Assignment #5: Fast, Reliable File Transfer with Custom TCP/IP

Instructor: Young H. Cho

Due date: Sept. 25 (Demo Only – No Report)

1. Improving TCP Performance Over Lossy Links

In this course, we have examined the performance of TCP and UDP under different network conditions. In laboratory 4 you developed a solution to transfer large data over a high-latency lossy link. In this lab, we want to explore ways to improve TCP performance so that any TCP application can take advantage of your solution. You will present the results of the implementation in your demo.

1.1. Congestion vs. Lossy Links.

The acknowledgment 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 1 and 2 need to exchange data using TCP. The nodes are connected by high latency, lossy link (such as satellite or long-range wireless uplink).

2. Initial Experiments

The goal for this lab is to 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 using the standard FTP program(SCP) to move a 1GB (1024 byte) file from one node to another.

Using the same node configuration and the commands used in Lab 4, set the link between the routers to 100Mbps with an initial delay of 10ms with no loss. Create a 1GB file and use FTP between nodes to transfer the 1GB file. Record the performance obtained for your report.

Explore the problem space by varying the delay from 0 to 200ms (RTT) and 0 to 25% packet loss (TX, RX path) (i.e 20ms and 5% step sizes). What throughput can FTP achieve under these conditions?

You may want to present this as a 3D graph, or one graph with multiple lines.

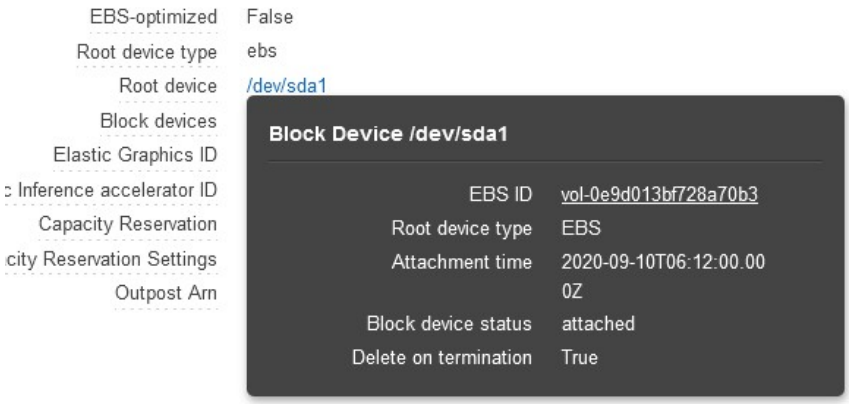
3. Improving TCP Performance

Now we need to consider ways to improve the performance of TCP under these conditions. The easiest way to modify or implement your TCP stack is to do it on Linux. There already exist several plug-able TCP implementations for Linux [3]. 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.

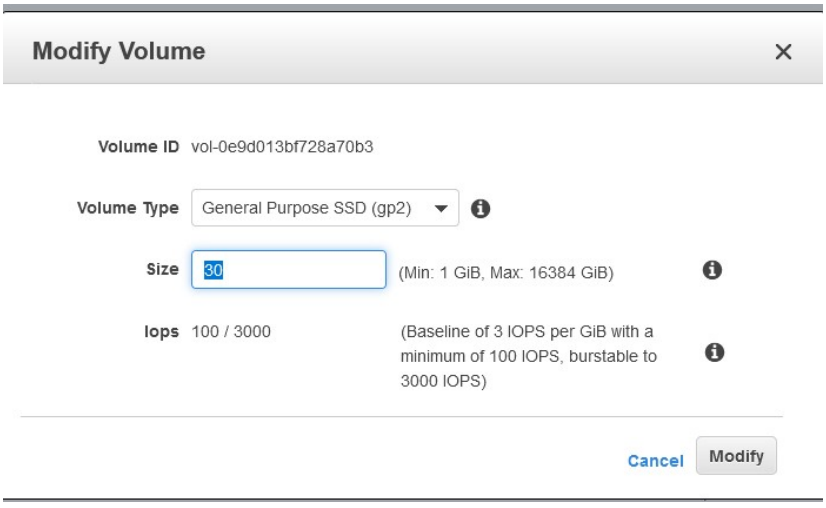
We suggest that you start early given the short deadline as well as the limited resource.

PROCEDURE TO BUILD UBUNTU KERNEL

In AWS, ubuntu uses a custom kernel optimized for AWS to support Xen para-virtualization. Since the compilation of the kernel produces a lot of object files, it is necessary to increase hard drive space to at least 40Giga bytes. We will do this only for one instance and then copy the final resulting binary kernel image to another host and install it there (server, client). You can also build it under
VirtualBox with correct kernel version for AWS and copy the binaries to AWS instance. For increasing HDD space you need to identify your EBS volume at AWS. Click on the root device and then it will show your EBS volume id. (if you are trying it on VirtualBox you have to create a new 16.04 ubuntu instance with 50 GB HDD space)



Select that EBS ID and then go to actions->modify Volume and increase the size to 30 GB. You need to do these steps when your instance is in a stop state. (only 30GB is free under the free tier summing all instance HDD usage, So you would be charged a minor amount for using extra space). You can also refer below for detailed steps:
`https://medium.com/findworka/how-to-increase-disk-size-for-an-ec2-instance-on-aws-b82181df6215`



After above step is done you have to create swap space in you HDD. Follow instructions from below link to create a swap size of 1 GB. This is done as as physical memory is only 512 MB for t2.micro instance.
`https://linuxize.com/post/create-a-linux-swap-file/`

Now install git by executing:

```
# sudo apt-get install git
```

```
Also configure git to use low memory by executing:
# git config --global pack.windowMemory "25m"
# git config --global pack.packSizeLimit "25m"
# git config --global pack.threads "1"
```

You must checkout AWS ubuntu kernel code from repository. Methods of doing this will change over time. For example, type the following command to clone aws-5.4 version of the kernel.

```
git clone git://git.launchpad.net/~canonical-
kernel/ubuntu/+source/linux-aws/+git/bionic -b aws-5.4
```

Also edit contents under file: `/etc/apt/sources.list` by uncommenting all lines starting with `deb-src` which is needed for below commands to work.

Then install all libraries required for building by executing below command under AWS or VirtualBox:

```
# sudo apt-get build-dep linux linux-image-$(uname -r)
# sudo apt-get install libncurses-dev flex bison openssl libssl-dev
dkms libelf-dev libudev-dev libpci-dev libiberty-dev autoconf
```

Now to compile there are two ways.

1. First is the ubuntu advised way which does not support incremental build. (on EC2 micro instance it takes 10 hours to build)

Procedure is as follows:
Execute from root of your checkout repository:

```
LANG=C fakeroot debian/rules binary
```

After compilation *.deb binaries would be visible in cd ../ (in parent of your directory)

Now install kernel by executing:

```
sudo dpkg -i *.deb
```

Verify that you can see your version of vmlinuz and initrd under /boot directory.

Detailed instruction is present at: *https://wiki.ubuntu.com/Kernel/BuildYourOwnKernel*

2. For incremental build follow: (you should concentrate on this method)

https://davejingtian.org/2019/11/07/ubuntu-kernel-build-again/
https://wiki.ubuntu.com/KernelTeam/GitKernelBuild.
https://help.ubuntu.com/community/Kernel/Compile

```
# sudo make oldconfig
# sudo make -j1 bindeb-pkg  (-j4 -> if VirtualBox (assumed 4 virtual
core present)
# cd ..
# sudo dpkg -i *.deb
```

Now you need to change GRUB bootloader configuration to pick your new kernel to boot and start ubuntu.

First go to file /boot/grub/menu.lst and edit the configuration to look like below. Your two entries would be present at the bottom of the file. In that move the first entry to top as shown below and add default, fallback and timeout string (this is not applicable for VirtualBox). For more details refer:

https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/UserProvidedKernels.html

```
default=0
fallback=1
timeout=0

title           Ubuntu 16.04.6 LTS, kernel 4.4.0-1013-aws
root            (hd0)
kernel          /boot/vmlinuz-4.4.0-1013-aws root=LABEL=cloudimg-rootfs ro console=hvc0 console=tty1 con
sole=ttyS0 crashkernel=384M-2G:64M,2G-:128M
initrd          /boot/initrd.img-4.4.0-1013-aws

title           Ubuntu 16.04.6 LTS, kernel 4.4.0-1113-aws
root            (hd0)
kernel          /boot/vmlinuz-4.4.0-1113-aws root=LABEL=cloudimg-rootfs ro console=hvc0 console=tty1 con
sole=ttyS0 crashkernel=384M-2G:64M,2G-:128M
initrd          /boot/initrd.img-4.4.0-1113-aws
```

Now modify /boot/grub/grub.cfg config. You must edit the entry above the “Advanced” substring search in first occurrence (applicable for both aws and VirtualBox).

```
  else
    set linux_gfx_mode=keep
  fi
else
  set linux_gfx_mode=text
fi
export linux_gfx_mode
menuentry 'Ubuntu' --class ubuntu --class gnu-linux --class gnu --class os $menuentry_id_option 'gnulinu
x-simple-b7109b09-d3aa-418a-982a-04e9f04ddd76' {
        recordfail
        load_video
        gfxmode $linux_gfx_mode
        insmod gzio
        if [ x$grub_platform = xxen ]; then insmod xzio; insmod lzopio; fi
        insmod part_msdos
        insmod ext2
        if [ x$feature_platform_search_hint = xy ]; then
          search --no-floppy --fs-uuid --set=root  b7109b09-d3aa-418a-982a-04e9f04ddd76
        else
          search --no-floppy --fs-uuid --set=root b7109b09-d3aa-418a-982a-04e9f04ddd76
        fi
        linux   /boot/vmlinuz-4.4.0-1013-aws root=UUID=b7109b09-d3aa-418a-982a-04e9f04ddd76 ro  console=
tty1 console=ttyS0 nvme.io_timeout=4294967295 nvme_core.io_timeout=4294967295
        initrd  /boot/initrd.img-4.4.0-1013-aws
}
submenu 'Advanced options for Ubuntu' $menuentry_id_option 'gnulinux-advanced-b7109b09-d3aa-418a-982a-04
e9f04ddd76' {
        menuentry 'Ubuntu, with Linux 4.4.0-1113-aws' --class ubuntu --class gnu-linux --class gnu --cla
ss os $menuentry_id_option 'gnulinux-4.4.0-1113-aws-advanced-b7109b09-d3aa-418a-982a-04e9f04ddd76' {
                recordfail
                load_video
```

Change **vmlinuz and initrd.img** to your kernel version and appropriate path under /boot directory. Rest all string remains same. Mine was 4.4.0-1-13 here. (Any failure at these steps would result in ec2 not bootable, only way to recover then would be to delete this instance and repeat on new one).
Now save the change to this file and execute **sudo reboot.** After system comes up verify that your kernel is loaded by executing uname -a. It should **show your kernel version and build time at which kernel is created**.
One thing to note here is that every time you execute `dpkg -i` command, these files get overwritten. So, you would have to repeat all these steps again.
The files interested in this lab are located under net/ipv4/tcp*
For your coding/testing you can try it first on VirtualBox and then run it on AWS for actual measurements. This way you would be able to avoid additional charges of AWS. All the above procedures should work

on both AWS and VirtualBox. Only thing to note is to check out the correct version of kernel from git as per your base kernel (closer version) currently running.

Modifying TCP Module

Start by reading the paper "Removing Exponential Back-off from TCP" [1]. Start with the standard TCP stack for Linux and remove the exponential back-off 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 over a network that is configured to be 100 Mbps with 200ms RTT latency (100ms delay each way) and 20% packet loss (bi-directional).

Egress rate limiting: (only on router)

```
# sudo tc qdisc add dev eth0 root handle 1:0 tbf rate 100mbit latency 0.001ms burst
901555
# sudo tc qdisc add dev eth1 root handle 1:0 tbf rate 100mbit latency 0.001ms burst
901555
```

For RTT of 200ms, drop of 20%: (only on router)

```
# sudo tc qdisc add dev eth0 parent 1:1 handle 10: netem delay 100ms drop 20%
# sudo tc qdisc add dev eth1 parent 1:1 handle 10: netem delay 100ms drop 20%
```

4. Evaluation

Perform the step 2 experiment of this Lab using your modified TCP stack in place of the default TCP. 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 back-off 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.

[4] http://vger.kernel.org/~davem/skb.html

[5] http://vger.kernel.org/~davem/skb_data.html

[6] http://amsekharkernel.blogspot.com/2014/08/what-is-skb-in-linux-kernel-what-are.html

[7] https://git-scm.com/docs/gittutorial

[8] https://makefiletutorial.com/