README:

Contents:

1. sch_dycbq.c – This is the new queuing discipline “dycbq”. This is modification of “cbq” queuing discipline, where root can have two sub-classes i.,e children where each has dedicated space and shared space that is allocated based on incoming traffic.
2. sch_dyfifo.c – Actual functions which adjust shared space and enqueue to a FIFO queue and dequeue from a FIFO queue are written in this file. These functions are called by sch_dycbq, whenever a packet to respective class arrives. Because of this dependency, always insert sch_dyfifo module first and then sch_dycbq. Similarly, sch_dycbq must be removed first and then sch_dyfifo while removing module.
3. q_dycbq.c – This file adds support for dycbq in tc program. This enables tc program to identify new queuing discipline and pass its parameters to kernel space, where sch_dycbq.c will receive using netlink protocol.
4. Makefile – This contains instruction to create kernel modules sch_dycbq.ko and sch_dyfifo.ko
5. DemoSlice – GENI Slice request that is used during project demonstration.
6. initRouter – Instructions to install required packages and sources (iproute2 and linux kernel) in router. These are required to “make” modules and enable support for dycbq by tc program.
7. platinumRoutes – adding routes in platinum node to enable pinging to other nodes.
8. regularRoutes – adding routes in regular node to enable pinging to other nodes.
9. routerRoutes – adding routes in router node and also enabling forwarding for this node.
10. serverRoutes – adding routes in server node to enable pinging to other nodes.
    - While adding routes please take care respective interfaces are correctly entered.
11. dycbqCommands – Contains commands that attach dycbq queuing discipline to root class and create two classes platinum (prio 1) and regular (prio 7) with respective rates.

Installing support for DYCBQ in Router

1. Open exoGENI and import DemoSlice request and submit slice
2. After all nodes become active, log-in to router node.
3. execute “vi initRouter” and copy contens of initRouter to this file. Click “ESC” and enter “:wq” to save this file. In terminal, now execute “ chmod 777 initRouter” to give access permissions.
4. execute “./initRouter”. Now all the necessary packages will get installed/downloaded. Enter “Y” whenever console requests to continue.
5. Now add necessary routes in router as given in “routerRoutes” file.
6. Navigate to iproute2 sources directory that was downloaded and extracted in step 4.
7. copy q_dycbq.c to <iproute2Sourcedir>/tc/
8. In iproute2Sourcedir, execute “make TCSO=q_dycbq.so” to create object that can be dynamically linked. Now execute “export TC_LIB_DIR=’./tc’” which enables this terminal session to use tc tool present in iproute2 source directory.

9. Now navigate to “~/<linux-kernel-sources>/net/sched/” directory. Copy sch_dycbq.c and sch_dyfifo.c files and replace existing Makefile with Makefile submitted.
10. execute “make” and then insert modules using “insmod sch_dyfifo.ko” and “insmod sch_dycbq” in exact order.
11. navigate to “~/<iproute2sources>/

Configuring Platinum Node:

In platinum node, add routes so that it can ping regular as well as server nodes. Add routes based on content in “platinumRoutes” file.

Configuring Regular Node:

In regular node, add routes so that it can ping regular as well as server nodes. Add routes based on content in “regularRoutes” file.

Configuring Server Node:

In server node, add routes so that it can ping regular as well as server nodes. Add routes based on content in “serverRoutes” file.

Testing DYCBQ:

After configuring all nodes, ping from each node to other to verify connection. Now in router node, in iproute2 sources directory, execute commands present in dycbqcommands file. This will attach dycbq queuing discipline to root node.

Commands example:

- tc/tc qdisc add dev eth1 root handle 1: dycbq bandwidth 10Mbit avpkt 1514 totalSharedSpace 250
- tc/tc class add dev eth1 parent 1:0 classid 1:1 dycbq rate 600kbit allot 1514 variableSpace 200 limit 400 prio 1
- tc/tc class add dev eth1 parent 1:0 classid 1:2 dycbq rate 100Kbit allot 1514 variableSpace 50 limit 300 prio 7
- tc/tc filter add dev eth1 protocol ip parent 1:0 prio 1 u32 match ip src 172.16.5.15 flowid 1:1
- tc/tc filter add dev eth1 protocol ip parent 1:0 prio 1 u32 match ip src 172.16.5.10 flowid 1:2

Above set of commands, specify output link rate is 10Mbit (10 megabits per second) and total shared space is 250 packets. Each packet on average is of length 1514 bytes that root class can receive.

It attached two classes (platinum prio 1 -> 1:1 and regular prio 7 -> 1:2) to root (1:0). Each class output rate is provided as 600kbit and 100kbit respectively. Also, limit (dedicated space – 400 packets and 300 packets) and initial variable space (200 packets for platinum and 50 for regular).

tc filter statements take care that packets coming from particular source ip address are forwarded to classes provided as flowids.

1. Now in platinum node, execute “ping –f –n 100 –s 1514 –i 0.001” and in regular node execute “ping –f –n 100 –s 1514 –i 0.001”.
2. After ping is done, in router node execute “dmesg –c” to check how incoming traffic was to each class.
3. Executing “dmesg | grep -o -P 'csv.{0,40}'” will show entries in following order (Pt – Platinum & Rg –Regular)
   PtpacketsReceived,PtOccupancy,PtTotalSpace,RgpacketsReceived,RgOccupancy,RgTotalSpace
   Scrolling down the entries shows how totalSpace (dedicated + shared) is behaving according to incoming traffic. PtPacketsReceived -> This shows number of packets received of the total 50 packets that was considered while adjusting.

Similarly iperf tool can be used to analyze packet loss when UDP transmissions and throughput in case of TCP transmissions.