**CS4283/5283: Computer Networks**

Vanderbilt University, Fall 2022; Instructor: Aniruddha Gokhale

**Programming Assignment #2**

Handed out 9/22/2022; Due 10/07/2022 (11:59 pm Central in Brightspace)

# **Assignment Theme:** Impact of Network Perturbations on Application Performance

**Overview**

This assignment (and all subsequent assignments) will reuse our application layer, which is the Smart Refrigerator protocol and its serialization capabilities in JSON/FBUFS. However, unlike Assignment #1, where we focused on the application-level protocol development and tested the basic client-server capabilities under normal conditions, this assignment focuses on evaluating the end-to-end performance of the request-response under different network perturbations and under multiple refrigerator clients sending their requests periodically to the grocery order and health status servers.

**Expected Operation: What remains same as before/What changes?**

In Assignment #1, our client (refrigerator.py) sent the request directly to either the grocery order server or the health status server alternately. However, in this assignment, we are now going to involve the network layer. So, instead of sending the request directly to the final recipient (grocery or health server), we are now going to send the information hop-by-hop by consulting a route given to us. Although Mininet will take care of sending our packet from the sender to the receiver, we are going to create our own overlay of routers by picking a subset of the hosts from the Mininet topology and letting them behave as routers. To that end, we will need to build an additional Python file called router.py whose job in life is to receive a request and simply forward it to the next hop by consulting the supplied route. If a router is the last router in the path, it sends the packet to the true destination and awaits a reply from that destination, propagating the reply in the reverse direction.

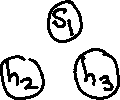
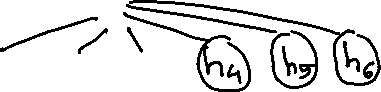
Thus, we will need to add one more entry to our config.ini as follows along with some number representing the topology (you will need to make sure that a corresponding Mininet topology is created corresponding to the route:

**[Network]**

Route=<some file that has the routing table for that experiment (see example)>

As an example, consider the following Mininet topology obtained using the command:

sudo mn --topo=single,6



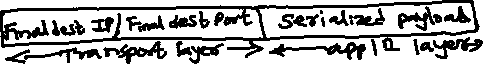
Now suppose that we run our refrigerator client on host h1, grocery order server on h5 and health status server on h6. Suppose that hosts h2, h3 and h4 are behaving as routers. Suppose we are now told that to reach host h5 from h1, we must follow the route h1->h2->h3->h5 and to reach host h6 from h1, we must follow the route h1->h2->h4->h6. By default, Mininet starts assigning IP addresses in monotonically increasing order. Thus, h1 gets 10.0.0.1, h2 gets 10.0.0.2 and so on.

The routing database given to us will be a file with contents like this. You can specify it as csv file or any format that you are able to quickly parse.

|  |  |  |
| --- | --- | --- |
| **Host/Router** | **Destination** | **Next hop** |
| H1 | 10.0.0.5 | H2 |
| 10.0.0.6 | H2 |
| H2 | 10.0.0.5 | H3 |
| 10.0.0.6 | H4 |
| H3 | 10.0.0.5 | H5 |
| 10.0.0.6 | H6 |
| H4 | 10.0.0.5 | H5 |
| 10.0.0.6 | H6 |

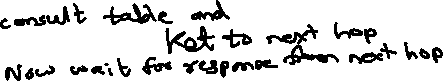
**Where/How to start working on the assignment?**

In PA1, our application-level business logic passed the IP address and port number of the grocery and health servers, which was passed on all the way down to the network layer, which then used ZMQ to create a REQ socket and directly connect to the final destination. In this assignment, rather than connecting directly to the final destination IP address, we now use the incoming destination IP address and consult our routing table to see how our next hop is for the given destination. We then connect to that IP address. In this way a chain of connections gets setup for the given destinations at each intermediate routers. What this also means is that you will need to create a router.py file with the basic business logic that it consults its routing table for an incoming destination and forwards to next hop depending on some small header that is added to the transport layer which includes the real destination IP address and port of the final application. Thus, the packet that gets pushed to our network layer will look something like this:



Once all of this is setup, then start running the actual transfer of application packets from refrigerator to the respective servers. You may be able to run multiple refrigerator clients and they all follow the routing table.

The logic of each router may look something like this:



**Milestones (and hence the deliverables and expectations):**

Please use the Slack Channel for programming assignments for discussions/questions.

***Milestone 1:***

* Addition of a small header at transport layer that keeps track of destination IP and port, which is then consulted by network layer to determine next hop
* Ability to specify the routing info in the file according to the specified scenarios and then be able to parse it as part of the config parser.
* Router logic which does the next hop forwarding by consulting the supplied routing info. Assume that only one router runs on the specified hosts and each router always runs on a well-defined port like 1234 or something like that.
* Submit to Brightspace what all have you accomplished up to this point and what was not accomplished, and difficulties, if any.

***Milestone 2 (final):***

* Run one refrigerator, one grocery server and one health server for each of the topologies under (a) no loss (b) some percentage loss
* Repeat the same but now with 2 refrigerator clients, say on the same host, and see what is the impact on performance.
* Collect end-to-end latency data for all cases without loss and with loss, and see the impact on performance. Remember that we are relying on existing TCP doing the necessary retransmission, etc.
* Plot graphs
* Create video for the grader
* Submit the code, plots etc to Brightspace

**Mininet topologies to be used**

1. Six hosts and one switch: *sudo mn --topo=single,6 --link=tc,delay=10ms,loss=5 (note that in the first run, do not specify loss while in the second run, specify the loss)*

Will create six hosts: h1 through h6 all connected to a single switch. Each link will have a base delay of 10 msec. Deploy your client on h1, grocery server on h5, and health server on h6. Use the route as specified in the example above.

1. Six hosts and six switches: *sudo mn --topo=linear,6 --link=tc,delay=10ms,loss=5 (note that in the first run, do not specify loss while in the second run, specify the loss)*

Will create six hosts: h1 through h6 all connected to a single switch. Each link will have a base delay of 10 msec. Deploy your client on h1, grocery server on h5, and health server on h6. Use the route as specified in the example above. In this, we should expect some more degraded performance.

1. 27 hosts and 13 switches: *sudo mn --topo=tree,depth=3,fanout=3 --link=tc,delay=10ms,loss=5*

Creates a Tree of depth 3 and each intermediate node (i.e., switch) having 3 children each. Deploy grocery server on h1 and health server on h27. Run one or more clients on host h1; grocery server is on h19, and health status server on h27. Say that to reach h19 from h1, we follow a subset of routers, say h1->h5->h7->h15->h18->h23->h19. Say to reach h27 from h1, we follow a subset of routers, say h1->h6->h10->h17->h24->h27.

**Rubrics (for grading after the final milestone)**

* Correctness (program works): 40%
* Experiments in all scenarios 20%
* Plots of results 20%
* README file explaining how to run the code: 10%
* Zoom-demo or self-explanatory video to grading peer: 10%