Electrical & Computer ENGINEERING

18-345: Introduction to Telecommunication Networks
Spring Semester, 2018

Project 3: Peer Network

Deadline: 11:59pm EST, April 6th

Questions? Email TAs

1. Introduction

In this project, we will explore the underlying network system, which forms our P2P network. You will implement a simple link-state routing protocol for the network. We could then use the obtained distance metric to improve our transport efficiency.

Note that it should be possible to complete most part of this project even if your previous projects were not fully operational.

2. Project 2 modification / Interface

2.1 Associate peer node with content using peer identifier or name

/peer/add?path=<contentpath>**&peer=**<peer UUID>**&rate=**<kbps>

This is exactly the same as /peer/add command specified in project 2. However, we should be able to specify a peer identifier (see 3.1) as the content holder instead of full host and port information. This request could be called multiple times with different peer identifier if the content could be found on multiple peers.

2.2 Kill peer process

/peer/kill

We would like to be able to remotely shutdown your peer process. When the frontend receive the URI, your process should attempt to terminate (System.exit or exit)

2.3 JSON Object Notation

To facilitate testing, in section 3, we ask you to provide additional command URIs which return JavaScript object notation (JSON - http://www.json.org/) as a result. Such result is only a text document

containing a well-formed string (without any header/footer in the content) The HTTP content-type should be **application/json**. For example:

{"metric":10} represents an object with a field called 'metric' and an integer value of 10. [10,20,30] represents an ordered array of length 3 containing 3 numbers 10,20, and 30 respectively.

Any number of whitespaces could be inserted (but not necessary) between any quotes, braces or commas. Note that you do not have to parse these notations; you only have to print them out.

You could use any library found on the web page (e.g. cJSON, Jackson, or simply printf) to generate the result. If you use a library, please include its source/header and modify your makefile to compile them correctly.

3. Project Tasks

3.1 Configuration File

Your node should be able to read a configuration file (by default) "node.conf" located in the same directory in which we execute the server. Alternatively, you should be able to take any configuration file provided with **–c** option when we execute the peer process (e.g. with a command-line **./vodserver –c node.conf**) Here is an example of a node.conf:

```
uuid = f94fc272-5611-4a61-8b27-de7fe233797f
name = node1
frontend_port = 18345
backend_port = 18346
content_dir = content/
peer_count = 2
peer_0 = 24f22a83-16f4-4bd5-af63-9b5c6e979dbb,pi.ece.cmu.edu,18345,18346,10
peer_1 = 3d2f4e34-6d21-4dda-aa78-796e3507903c,mu.ece.cmu.edu,18345,18346,20
```

These are the option details. You could assume that the configuration file will be well-formatted. However, all fields are optional. **If the uuid option does not exist, you must generate a new UUID** (see 3.1) **and update the configuration file** (either node.conf or those specified in –c parameter) with the generated identifier.

- **uuid** – The node's unique identifier (see 3.1, default: not-specified)
- **name** – A user-friendly name used to call the node
- **frontend_port** – Front-end port of the node (HTTP server, default:18345)
- **backend_port** – Back-end port of the node (UDP-Transport backend, default:18346)
- **content_dir** – The content directory for this peer (default: content/)
- **active_metric, extended_neighbors** – (see extra credit, default: 0)
- **peer_count** – the number of neighbors specified in this configuration (default: 0) This option will follow by exactly peer_count lines providing peer information.

- **peer_x** = uuid,hostname(or ip address),frontend port,backend port,distance metric
  A comma-separated value listing initial neighbors for this node and their distance metrics

3.2 Node Identifier

Each peer node should have a unique identification. A Universally Unique Identifier is a 16-byte (128-bit) number, which could be used to uniquely, identified information without the need of central coordination. The UUID should be generated upon the creation of each peer node. It should be persisted in the configuration file so that we can identify the same node across restart.

You could execute 'uuidgen' command to generate a new UUID. For this project, you could use java.util.UUID.randomUUID or uuid_generate/uuid_unparse provided by libuuid (include /usr/include/uuid/uuid.h and compile with -luuid). The libraries are available on the cluster.

Request URI
/peer/uuid

Response: the uuid of the current node
{"uuid":"f94fc272-5611-4a61-8b27-de7fe233797f"}

3.3 Reachability

You should add additional message to your backend; A Keepalive message. The purpose of such message is to notify your neighbor that you can still reach them. For example, each node could send out a keepalive message to its neighbor every 10 seconds. A neighbor could be declared as unreachable if we miss three consecutive keepalive messages.

Request URI
/peer/neighbors

Response: a list of objects representing all active neighbors
[{"uuid":"24f22a83-16f4-4bd5-af63-9b5c6e979dbb", "name":"node2", "host":"pi.ece.cmu.edu", "frontend":18345, "backend":18346, "metric":10}, {"uuid":"24f22a83-16f4-4bd5-af63-9b5c6e979dbb", "name":"node3", "host":"mu.ece.cmu.edu", "frontend":18345, "backend":18346, "metric":20}]

Request URI
/peer/addneighbor?uuid=e94fc272-5611-4a61-8b27-de7fe233797f**&host=**nu.ece.cmu.edu**&frontend=**18345**&backend=**18346**&metric=30**

Response: unspecified

Action: Add the given node with the given uuid, host, frontend port, backend port, and distance metric as your new neighbor (There will not be any newline/whitespace character in the request URI)

Your node should start performing reachability check on the given node. If the node is active, subsequent calls to /peer/neighbors should contain the information about this node.

3.4 Peer discovery and link state advertisement

In this part of the project, we will use our initial neighboring peers to discover more information about the rest of the network.

In an analogy to the actual network infrastructure, we could view each peer node as a network router. Initially, our nodes only know information about their neighbors. It will later discover the rest of the network by performing a 'link-state' routing protocol.

In a link-state routing protocol, only information about the network link is exchanged between routers. A complete map of the network is then reconstructed on every router, and a routing table will be calculated based on the map. This is in contrast to a distance-vector routing protocol where each node tries to share its routing table with its neighbor.

For this part of the project, you could implement a **link-state advertisement** protocol. The goal is to distribute the same network map to every peer. Note that this mechanism may not be feasible on a large-scale P2P network where there are millions of nodes. But it is simple and sufficient on a small scale deployment.

A simple version of a link-state advertisement protocol could be explained as followed:

Periodically, or when there is a change in a node's neighbor, each node creates a link-state advertisement which contains:

- The identity of the node which produce the advertisement
- The identity of all of its neighbors and their connectivity metric
- A sequence number which increments when the source node create a new advertisement

The advertisement is then sent to all neighbors.

Note that each node should remember, for every other node, the sequence number of the last link-state advertisement it received. Upon receiving a new advertisement message, the receiver should compare the advertisement's sequence number with what it had.

- If the sequence number is lower, the message should be discarded.
- If the sequence number is higher, the receiver should update its network map and forward a copy of the advertisement to its neighbors.

Note that the above protocol is only one of the suggestions. The goal for this part of the project is to distribute the network topology and its connectivity metrics to every node in the network.

Request URI
/peer/map

Response: An object representing an adjacency list for the latest network map. It should contain only active node/link. The object's field name should be node's name (by default), or UUID (if a node name was not specified.)

For example, this response should be generated for graph in figure 1 (assuming all nodes are active).
{ "node1":{"node2":10,"node3":20},
"node2":{"node1":10,"node3":20},
"node3":{"node1":20,"node2":10,"node4":30},
"node4":{"node3":30} }

3.5 Priority Selection

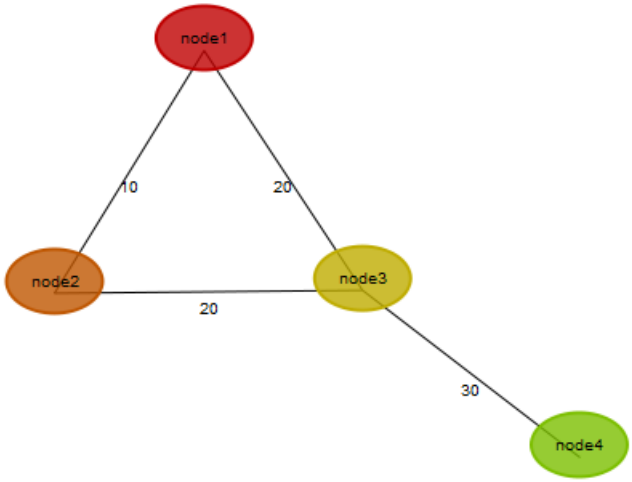

Figure 1 A simple network topology

For this part of the project, we will construct a routing table for the network. Using the network topology and connectivity metrics provided from the previous part. We could then use the routing information to calculate the distance metric between a node and the content requested.

Given a set of *n* potential content nodes $\{c_1,c_2,\ldots,c_n\}$ with /peer/add, your task is to **rank the preference of these content nodes based on their shortest distance metric between the current node**. When assigned properly (see extra credit), such metric could reflect the network distance between nodes.

Request URI
/peer/rank/<contentpath>

Response: an ordered list (sorted by distance metrics) showing the distance between the requested node and all content nodes.

For example, the following response will be generated if the request URI was made to node1 (in figure 1) and if all other nodes have the content.
[{"node2":10}, {"node3":20}, {"node4":50}]

4. Deliverable Items

The deliverable items are enumerated as follows.

1. **The source code** for the entire project (if you use external libraries unavailable on the cluster, provide the binaries needed for compilation of your project). Please submit a zipped folder containing all the source code. The folder is to be named as andrewid_project4.

2. **Makefile** - You should prepare a makefile for C make (or build.xml for Java Ant) which generates the executable file for the project. We expect the file to work from your submission directory. For example, the following calls should generate an executable vodserver (for C) or VodServer.class (in edu.cmu.ece package directory for Java) respectively. The following commands will be attempted on your submission directory, depending on your choice of programming language.

   …/<andrewid>/project3$ **make**
   …/<andrewid>/project3$ **ant**

3. **Do not submit the executable files** (we will DELETE them and deduct your logistic points). However, we must be able to invoke the one of the following commands (after invoking make/ant) from your submission directory to start your server with the given configuration file (if no config file was specified, use node.conf or create a new one).

   $ **vodserver –c node.conf**
   $ **java VodServer –c node.conf**

4. **A brief design document** regarding your server design in design.txt / design.pdf in the submission directory (2-3 pages). Apart from the team information, tell us about the following,
   a. Did you use the backend protocol (udp) or the frontend (http) server to handle advertisement/reachability messages?
   b. What method did you use to perform link-state advertisements and priority selection?
   c. Libraries used (optional; name, version, homepage-URL; if available)
   d. Extra capabilities you implemented (optional): Please specify in this section if you have implemented any extra features and how you achieved it
   e. Extra instructions on how to execute the code.

5. Grading

Partial credits will be available based on the following grading scheme:

Items	Points (100 + 30 – extra)
Logistics	**15**
- Successful submission & compilation	5
- Design documents	10
Setup	**25**
- Parse configuration file	10
- Generate UUID (/peer/uuid) and save it to config file	5
- Add neighbor (/peer/addneighbor)	5
- Peer kill (/peer/kill)	5
Peer Routing	**60**
- Reachability (/peer/neighbors)	10
- Link state advertisement (/peer/map)	20
- Priority Rank (/peer/rank)	30
Extra Achievements	**30**
- Active distance metric	10
- Extended neighbors	10
- Transport integration	10

Extra Credit:

Active distance metric (enabling configuration: active_metric = 1): Right now the distance metric provided by initial configuration is static. Implement a 'better' distance metric based on active or passive probe. Tell us why such metric is useful and should be more preferred in this situation. However, if you do implement this, we should be able to switch the metric back to static (to verify your code correctness) by specifying active_metric = 0 in the configuration file.

Extended neighbors (enabling configuration: extended_neighbors = 1): In an actual P2P network, you want to be able to always remain connected to the network even if your neighbors aren't... Come up with a scheme, which can provide this functionality and implement it. What would you do when you restart your peer and none of your neighbors were there?

Transport integration: Use the priority rank from 2.4 to assist your transport backend. You should be able to transfer most of the content from the node which ranks at the top of your metric and proportionally smaller amount of traffic from subsequent nodes. Show us a case where this actually improves your transport layer.