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

Project 4: Content Search

Deadline: 11:59pm EST, April 27

Questions? Email TAs

1. Introduction

For the last part of this project, we will implement a simple gossip protocol which performs a distributed search for content.

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

2. Project 3 modification / Interface

2.1 Project 3 dependency

We expect you to complete the configuration file part (Project 3 section 3.1) of project 3 before proceeding. Additional configuration options should be added (please see section 3.3 for detail):

- **search_ttl** – default TTL for a search request (default: 15)
- **search_interval** – time interval (in milliseconds) between each gossip rounds (default: 100)

2.2 JSON Object Notation

To facilitate testing, 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.

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 Content Search Interface

The only task left for our system is to perform a network-wide content search. To achieve this task, we will add an additional peer/search URI interface. The following request should initiate the search protocol described in 3.2.

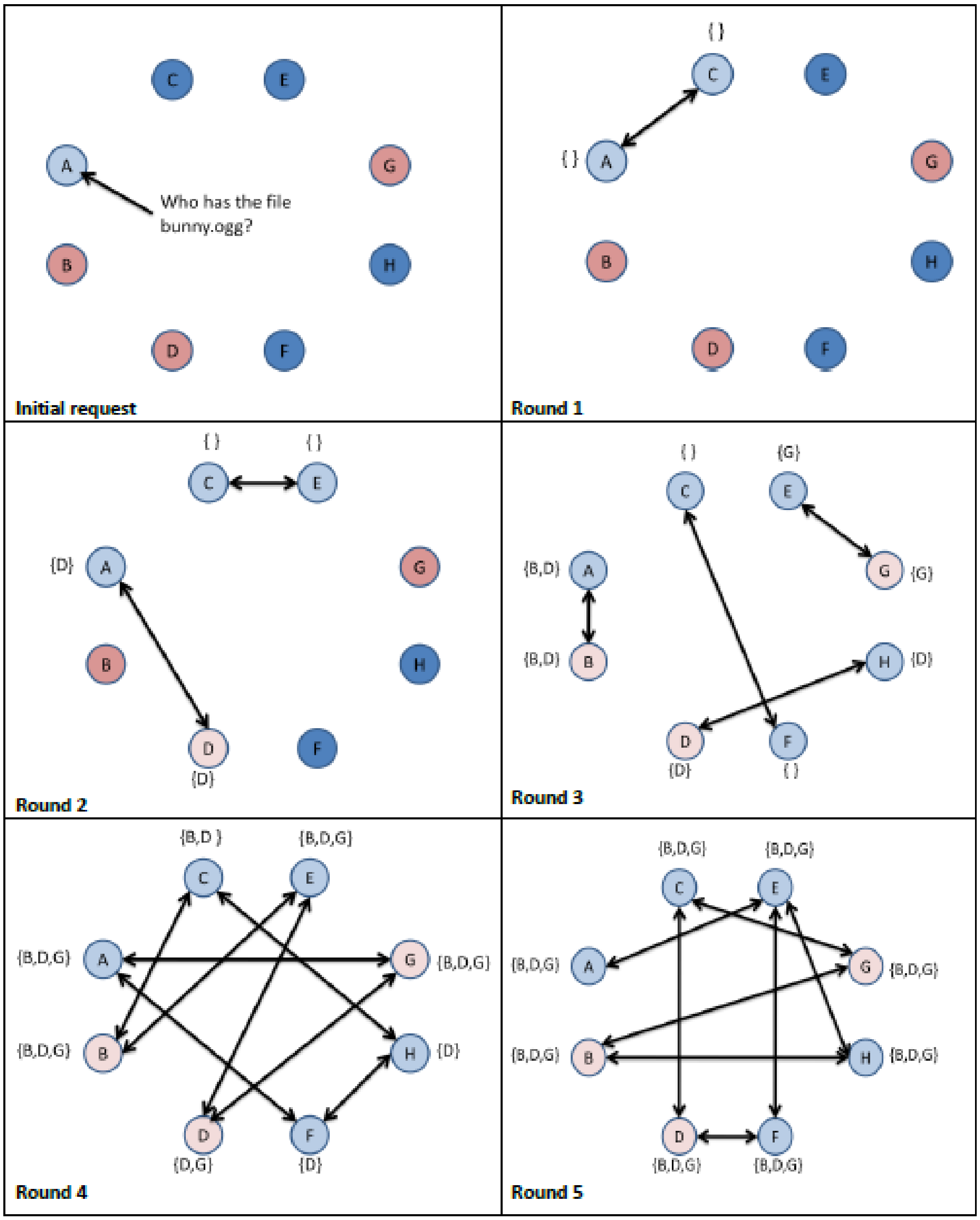
Request URI:
/peer/search/<name(relative to content_dir) of content>
Example:
/peer/search/video.ogg
Response: A list of matching contents (see extra credits) and the list of peer UUID which have the content (for the above example, given the configuration ‘content_dir=content/’, the peers should have file “./content/video.ogg”). The following response lists all the peers which contains video.ogg.
[{“content”:“video.ogg”**, “peers”:[**“f94fc272-5611-4a61-8b27-de7fe233797f”, “24f22a83-16f4-4bd5-af63-9b5c6e979dbb”, “e94fc272-5611-4a61-8b27-de7fe233797f”**]}]**

3.2 Gossip Protocol

Gossip protocol is a class of protocol which is very similar to a network broadcast. However, the difference is the communication occurs in multiple rounds, with a bounded-size of the total message exchanges. For each round, a peer will exchange information with one of its neighbor, hence learning new information about the network. This new information will be exchanged with another neighbor on subsequent rounds. A simple gossip protocol for content search could be described as followed:

- Upon receiving a peer/search request, add the request to the list of search requests. Initiate a time-to-live(TTL) to search_ttl rounds.
- For each round (every search_interval ms):
    - For each search request:
        - decrease the TTL, do not proceed if TTL = 0
        - Pick a random neighbor, and send an exchange message (containing the content path, TTL, and last known list of peers with content) to the neighbor.
- When a node receives an exchange message:
    - If the node has not seen the same request for the content before, initiate a new search request with the given path, TTL and peer list.
    - Check if the node has the content, if so, add itself to the exchange’s peer list
    - Merge the list of peers received in the message with the node’s known list of peers

The figure shown in the next page illustrates the above protocol. We assume that the network has full mesh connectivity (every node can talk to every other node – note that the above protocol does not require such assumption to work). The figure show how peers exchange information about a content file. The red nodes designated nodes with the content file. Highlighted nodes indicate that the nodes have received the search request.



3.3 Integration

Once you have the search protocol (and the rest of the project) implemented, we should have a working P2P video-on-demand network. Try putting a file named video.ogg in one of the peer's content directory and sending the following URIs to another peer.

http://viewer:8345/peer/search/video.ogg

http://viewer:8345/peer/view/video.ogg

You should be able to view the video from any of the nodes in the network using the above URIs.

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)

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>/project4$ make
   <andrewid>/project4$ 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 <u>no config file was specified</u>, use node.conf or create a new one).**

   ```
   $ vodserver –c node.conf
   $ java edu.cmu.ece.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. What are the drawback/limitations of the given protocol?
   b. Libraries used (optional; name, version, homepage-URL; if available)
   c. Extra capabilities you implemented (optional): Please specify in this section if you have implemented any extra credit options and how did you achieve it
   d. Extra instructions on how to execute the code, if needed

5. Grading

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

Items	Points (100+20)
Logistics	**20**
- Successful submission & compilation	10
- Design documents	10
Implementation	**80**
- Parse configuration file	10
- Peer search (/peer/search)	60
- Integration (/peer/search then /peer/view)	10
Extra credit	**10**
- Partial search	10
- Content Portal	10

Partial Search – It is possible to extend the provided protocol to perform partial matching of content. Can you perform a partial matching showing any content whose name contains the given request? e.g. /peer/search/vid should return a list of contents/peers objects

[

{“content”:“video.ogg”, “peers”:[“f94fc272-5611-4a61-8b27-de7fe233797f”, “24f22a83-16f4-4bd5-af63-9b5c6e979dbb”, “e94fc272-5611-4a61-8b27-de7fe233797f”]},

{“content”:“myvideo.ogg”, “peers”:[“f94fc272-5611-4a61-8b27-de7fe233797f”, “e94fc272-5611-4a61-8b27-de7fe233797f”]}

]

Since both “video.ogg” and “myvideo.ogg” both contains “vid”, and afterward /peer/view/video.ogg should instantly play “video.ogg”

Content Portal – We, as users, usually prefer browsing instead of searching (pure conjecture)… Give us a homepage (reroute your homepage http://vodserver:port/ to this portal) that show the list of latest and greatest contents (very subjective) on your P2P network! If we put a file on a content folder on one of the node, it should show up on the homepage almost immediately. Note that merging the dynamic content folder may be actually harder than it appears.