EECS 325/425
Fall 2019
Due before: Oct. 28, 2019 11:59pm

Project 1: Building a decentralized P2P network

In this project, you will implement a P2P network consisting of peer nodes that will connect to other peers to share files according to the following scheme:

P2P Network Aspect	**325**	**425**
Peer topology/discovery	Neighbors are hard-coded into configuration file	A new peer discovers other peers using broadcast of a UDP discovery msg, starting from an existing peer
Peer connection (for queries)	TCP	TCP
Query pattern	Flooding	Flooding
File transfer	Ad hoc TCP	Ad hoc TCP
Peer status check	Heartbeat over peer connection	Heartbeat over peer connection

System description and instructions:

The peer lives in ~/p2p directory. The peer is started by typing on the command line, while in ~/p2p working directory:

"java p2p"

Besides the peer, this directory contains the following:

- Three config files: ~/p2p/config_peer.txt which specifies the local ports the peer will be using, ~/p2p/config_sharing.txt which lists all files to be shared, and, for 325, config_neighbors.txt which lists all neighbors (IP address and port numbers, make sure these ports agree with the ports from the corresponding config_peer.txt files). Important: when specifying the config_neighbors.txt for each peer, make sure that the resulting graph of neighbor relationships is connected. Also, please specify two neighbors for each peer.
- A directory ~/p2p/shared which contains all files to be shared with other peers. Make sure the contents of this directory is consistent with the config_sharing.txt configuration file.
- A directory ~/p2p/obtained which contains all files that have been obtained from other peers. Note that obtained files are private — they are not shared further by this peer.

On start-up, a peer peer opens two TCP ports as listed in the config_peer.txt configuration file: one for neighbor connections (over which queries and responses are sent) and one for file transfers. The 425 peers also open a UDP port (also listed in the config_peer.txt) to exchange peer discovery messages. The peer then waits for input from the standard input.

- 325: ***After starting all the peers***, the user types command "Connect" in each peer's standard

input. The peer will read the config_neighbors.txt file and establish TCP connections to listed peers.

- 425: ***After starting all the peers,*** the user types command "Connect <peer IP addr> <peer Port# >" where the arguments specify an exiting peer *already in the network* (make sure the existing peer is in-network unless it's the first peer, or you will create isolated islands). The peer runs a peer discovery protocol starting from the specified peer using UDP messages. After collecting Pong messages for a certain time, the peer picks two peers and establishes TCP connections with them.
- After the above steps, the peer waits for user input. The peer also (a) waits on its established TCP connections with the neighbors for file queries; (b) periodically sends heartbeat messages to its neighbors on the same connection; and (c) listens on a welcome socket for any ad-hoc TCP connections for file requests.

During operation, a user can type the following commands on the standard input of any peer:

- "Get <file name>". The peer runs the file query protocol to find and download this file. The peer stores the downloaded file in ~/p2p/obtained directory.
- "Leave". The peer closes all connections with its neighbors.
- "Connect" – the same command as above
- "Exit". The peer closes all connections and terminates

Finally, when a peer does not receive the heartbeat messages from a neighbor for a long time (provide the timeout in the connects) peer closes the connection with this neighbor.

Requirements:

1. Peer code must run on Linux servers provided for the class. You must use Java for this project.
2. Your P2P network must contain six peers, with one peer on each of the six provided servers. Topology (that is, which the peers are connected to which peers) is at your discretion.
3. Each peer must simultaneously (1) wait for user's requests for files from the command line (2) periodically check the status of their connected peers and respond to status checks from others, (3) listen for and respond to file queries from peers, and (4) listen for and respond to file requests from peers.
4. Each peer must print a line of output upon
   - starting
   - connecting to a peer
     - Print a line before this peer attempts to connect to another peer and another line reporting the outcome of the attempt (success/failure)
   - accepting a connection initiated from another peer
   - sending a query to another peer
   - receiving a query
     - Print a line upon receiving a query
     - Print another line indicating the query result (does this peer have the requested file or not?)
   - sending a heartbeat to a neighbor
   - receiving a heartbeat from a neighbor
   - closing the connection to a neighbor when the heartbeat does not arrive for a timeout value
   - requesting a file transfer from another peer
   - receiving a file transfer request

- completing the file transmission to the requesting peer

Protocols

1. The query protocol:
   - A query message will be plain text in the format: "Q:(query ID);(short string)" where short string represents file name.
   - The response message will be plain text in the format "R:(query ID);(peer IP:port);(filename)"
   - Note 1: the responses are sent back to the immediate neighbor that sent the corresponding query, not to the original peer that initiated the query flood. In other words, responses retrace backward the query paths.
   - Note 2: when an intermediate peer forwards a query on behalf of a neighbor, the intermediate peer may receive multiple responses from the peers that own the file. Make sure your application handles this correctly and explain how you handle this in your README file.
2. The transfer protocol:
   - The request message will be plain text in the format "T:(filename)"
   - The response message will be the plain text containing the file, followed by the connection closure
   - The transfer protocol operates over a separate ad-hoc TCP connection between the recipient and provider of the file.
3. (425) The peer discovery protocol:
   - The ping message will be plain text UDP message in the format "PI:<joining peer IP addr>:<joining peer port>. This message is flooded through the p2p network; each peer that receives this message for the first time responds to the original sender with a pong message. Make sure that a peer discards silently duplicate ping messages.
   - The pong message will be plain text UDP message in the format "PO:<responding peer IP address>:<responding peer Port # for neighbor TCP connections>"

Resources

1. **Ports:** You will be allocated a port range of 20 ports reserved for your use. You can do all development on a single host (e.g., your laptop) but must be able to communicate between provided hosts to be graded.
2. **Files:** A zip file of public domain books and titles is provided with the assignment, along with an index file mapping filenames to titles and filenames to peers.

Deliverables

before the due date/time, please

1. Submit your code as one zip file to Canvas.
2. Deploy a peer (but don't start it – we will) and the peer's set of files on each Linux host

Grading Rubric (325: 85 points + 15 bonus points for 425 part; 425: 100 points total)

- Fundamentals

    - Code, configuration, and data (files) deployed as required on assigned servers and uses correct port number range: 5 points
    - Code is readable, well-structured, and free of hard-coded values: 10 points
- Peer Program and Protocols
    - Fully connected network is created by calling “connect” after all peers are started: 10 points
    - 425 only: peer selection via UDP responses is implemented: 15 points
    - Query protocol floods overlay network: 15 points
        - Are broadcast storms prevented?
    - Query response travels overlay network links to origin: 15 points
        - Do intermediate peers handle duplicate responses for a query correctly?
    - File transfer executed via direct P2P connection: 10 points
    - Heartbeat protocol and timeout implemented: 10 points
    - All status messages printed to standard out as required: 10 points