COMP3331/COMP9331

Assignment T3 2022

IoT Data Collection and Sharing

All details are in the specification

- **READ THE SPECIFICATION**
- **READ THE SPECIFICATION (AGAIN)**
- Information about deadlines, file names, submission instructions, marking guidelines, example interactions, and various other specifics are in the specification
- Choice of programming languages: C, Java, Python (2 or 3, but who's using 2?)
- This talk provides a high-level overview

Client-Server Architecture

- Server
  - Always on
  - Responsible for managing the network
    - Maintains state about the network
    - Performs computation for the edge devices
  - Implements a request/response API for interacting with client(s)
  - Prints informative updates to the terminal
- Client
  - Allows a device to interact with the server
  - Interaction is through the command line terminal
- Transport Protocol: TCP

Peer-to-Peer (CSE students only)

- Two clients can communicate directly to share files
  - Server facilitates that connection by telling one client where it can reach the other
- Transport Protocol: UDP

Design (1)

- Server must interact with multiple clients
  - We recommend a multi-threaded approach
  - Many implementation approaches possible, we do not mandate a particular approach
- Simple, sequential interactions with each individual client
  - You may assume that each interaction with a client is "**atomic**"

Design (2)

- You will have to define and manage a number of data structures (more on this later)
- You must design your own application layer protocol
    - This includes the syntax/semantics of the messages exchanged between the client-server and peer-to-peer, and the actions to be taken upon receiving each message
- You must be particularly careful about managing multiple threads and how they access the shared state (data structures, files, etc.)

Execution (1)

- Client(s) and Server executed on the same machine in different terminals
- **Server**
  - Command line arguments:
    - `server_port` : where server will be listening for clients, choose value in range [1024, 65535]
    - `number_of_consecutive_failed_attempts` : number of failed login attempts before account is blocked, choose value in range [1, 5]
  - Executed first – waits for client(s) to connect
  - Network is empty at start-up – no logs, no files, no connected devices
  - Server will keep executing forever

Execution (2)

- **Client**
  - Command line arguments:
    - `server_IP` : IP address of machine on which server is running (so "127.0.0.1")
    - `server_port` : should match the first argument for the server
    - `client_udp_port` : where client will receive data from other clients, choose value in range [1024, 65535] (CSE only)
  - Let the OS pick ephemeral ports to:
    - Initiate TCP connection with server
    - Send data over UDP to other clients (CSE only)

Device Authentication

- A credentials file will be available in the server working directory
  - Example is on the assignment page
  - Different file for marking, but same format
- To join the network, device must first be authenticated
- Client prompts to enter device name
- If server confirms the name is valid, client prompts for password
  - If invalid, prompt to retry
- If server confirms password is correct, authentication is successful
  - If incorrect, prompt to retry
  - After repeated failed attempts, account is blocked for 10 sec, client quits

IoT Network Operations

- Should support 7 commands (6 for non-CSE)
- "Device" is prompted to enter one of the commands
- Basic error checking for syntax, appropriate arguments, etc. should be implemented
    - We are more interested in observing if you have implemented the “normal” functionality, so our tests for error checking will be limited and won’t include bizarre edge cases
    - Many of these errors can be detected on the client-side and will not require communication with the server
- All client-server communication (authentication + 6 commands) use TCP
- UVF command is used to transfer files over UDP to another peer (CSE only)

Commands (1)

- **EDG** - Edge data generation
  - `EDG fileID dataAmount`
  - Client randomly generates integer data samples
  - Samples are saved in a text file
- **UED** - Upload edge data
  - `UED fileID`
  - Client transfers data file to server
  - Server appends upload log

Commands (2)

- **SCS** - Server computation service
  - `SCS fileID computationOperation`
  - Server performs some computation on previously uploaded data
  - Computations include `SUM`, `AVERAGE`, `MAX`, `MIN`
- **DTE** - Delete data file
  - `DTE fileID`
  - Server deletes its copy of the data file
  - Appends deletion log

Commands (3)

- **AED** - Active edge devices
  - `AED`
  - List all active devices (except device issuing command)
  - Read the log file and send relevant information to client
- **OUT** - Exit edge network
  - `OUT`
  - Device leaves the network, client quits
  - Server updates state/log about active devices

Commands (4)

- **UVF** - Upload video file (CSE only)
  - `UVF deviceName filename`
  - Client first confirms with server that the recipient device is active and gets recipients' peer-to-peer address
  - Recipient client should be "always on", listening for P2P file transfers
  - File is transferred directly between clients using UDP
    - In the context of this P2P exchange, the sender is initiating the exchange, so acting as a "client", while the receiver is always listening at a known address, so acting as a "server"

Commands (5)

- **UVF** ... *continued*
  - Receiving client should store file and display appropriate notification in terminal
  - P2P should happen concurrently with client-server
  - File names are case sensitive and don't include any whitespace
  - Make no assumptions on file format, just send as binary

Data Structures

- Server should maintain certain state information about the network, e.g. active devices, blocked devices, etc.
- Server will also have to manage several files – various logs and data files uploaded by devices
- Proper design of data structures is essential to ensure all functionality can be achieved
- Be careful about accessing the data structures across multiple threads
- Avoid arbitrary upper limits for variables
    - Dynamic memory allocation is recommended

How to start and progress

- Focus on client-server configuration
- Get the client to setup a TCP connection with the server
- Implement authentication
- Implement one command, test it thoroughly, move to the next
- Suggest the following order: OUT, AED, EDG, DTE, SCS
- Make sure server can communicate with multiple clients
- Then move onto P2P configuration and UVF command
- **STRONGLY SUGGEST TO DEVELOP YOUR IMPLEMENTATION IN VLAB ENVIRONMENT (NOT ON YOUR NATIVE MACHINE)**
    - Added benefit – CSE accounts are backed up

Report

- Program design
- Data structures
- Details of the application layer protocol
- Trade-offs considered
- Point out issues if program does not work under certain circumstances
- Reference any and all borrowed code

Testing (1)

- Test, Test, Test
- Server and client(s) executing on **same machine but different working directories**
- Emphasis on correct behaviour
- Basic error checking
- **MUST test in VLAB environment and through command line**
- If we cannot run your code, then we cannot award you any marks

Testing (2)

- Your assignment will be **MANUALLY** marked by your tutors
- Client and server must display meaningful messages in the terminal
  - Extensive examples are available at the end of the spec
  - You are NOT required to use the exact same text
- Detailed marking rubric is available in the specification
- SYSTEMATIC DEBUGGING + UNIT FUNCTIONS

Plagiarism

- **DO NOT DO IT**
- **Both parties involved are considered guilty**
- **DO NOT POST YOUR CODE ON A PUBLICLY ACCESSIBLE REPOSITORY HOSTING SERVICE (e.g. GitHub)**
    - **EQUIVALENT TO PROVIDING YOUR CODE TO THE ENTIRE WORLD**
- If caught:
    - You will receive zero marks (and there may be further repercussions if this is not your first offence)
    - Your name will be added to the school plagiarism register
- Every term the above has happened for a small group of students
    - Would be nice to not have to do it this term

Resources

- Many program snippets on the course site
  - Including multi-threading code snippets
- Your socket programming experience in Labs 2 and 3 will be useful
- Repository of resources is here

Seeking Help

- Assignment specific consultations (for all 3 programming languages)
  - In Weeks 5, 7, 8, and 9
  - Schedule is here
- Ed forum
  - Read posts from other students before posting your question
- **Read the spec - very often your answer will be in there**