Aalto University
School of Electrical
Engineering

S-38.3610 Network Programming

Spring 2015

Phase 1 Report

27.02.2015

Group 3: FINDME

Raghavendra M S, raghavendra.mudugoduseetarama@aalto.fi
Anteneh Adem, anteneh.adem@aalto.fi
Raul Morquecho, raul.morquecho.martinez@aalto.fi
Jaume Benseny, jaume.benseny@aalto.fi

1. Overview and overall architecture

Overview

FindMe is a name resolution application, in which one can query the information of a node (user) having only a piece of his/her information. For example, a query can be about knowing the complete information about a node just by sending its name. A query not necessarily has to be the name of the node, instead an email address can be used, and also location or street address information can be sent as part of the query to know rest of the information.

Architecture

FindMe can be divided into two separate applications:

1) **Client application:**

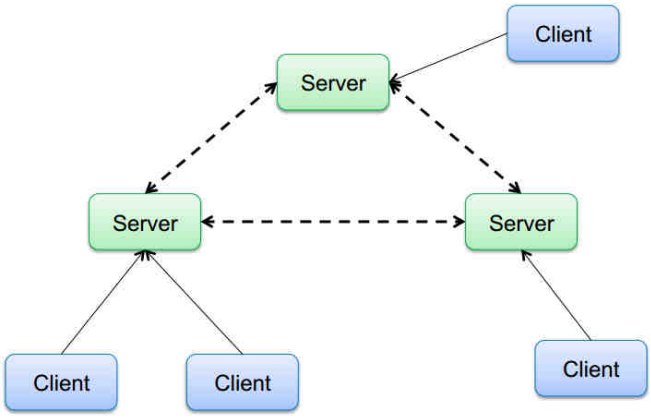
Client construct the query based on the input of the user, send the query to the server and waits for a reply, once the reply is received, the client displays the information to the user.

2) **Server application:**

Server is responsible for resolving the received queries from clients. The server is divided broadly into three modules.

a. Module 1. Data processing module
b. Module 2. Server interactive module
c. Module 3. Client interactive module

Overall architecture



2. Requirements description

General requirements

Major requirements to be implemented as part of FindMe:

- Uniquely identify clients
- Handling name resolution query from clients
- Handling an information update message from clients
- Data synchronization between the servers

Server requirements and associated modules

Server behaviour / required features:

- The servers store about clients:
  - Name
  - Email Address
  - GPS Location
  - Postal Address
- Servers synchronize information to keep updated data about clients
- Uniquely identifying the clients
- Handle resolution queries from clients
- They must be resilient to one server failure
- OPTIONAL FEATURE - Store queries that cannot be processed yet

Module 1. Data processing module

Module is responsible for

- Generate client IDs
- Receiving queries from clients and encode response
- Receiving messages from server and change particular entry

Inter module communication requirements

- Communications with Module 3 and Module 2 to manage data update petitions.

Module 2. Server interactive module

Module is responsible for

- General objective: Synchronizing database between servers
- Sends database updates, received from local database, to other servers
- Receives database updates from other servers
- Sends database updates, received from others servers, to local database

Network communication requirements

- Periodic communication among servers to keep integrity of client's data.

Inter module communication

- Communications with Module 1 to manage data update petitions.

Module 3. Client interactive module

Module is responsible for:

- Open socket server and listen to clients
- Replay to clients about other client position or related data
- Make sure petitions are delivered to Module 1.
- OPTIONAL - Replay to clients about position correctly updated

Network communication requirements

- Asynchronous communication containing small data fields

Inter module communication

- Communications with Module 1 to manage data update petitions.

Client requirements

Client behaviour / required features:

- Query servers about other clients
- Receives replays from the server and prints it to the screen
- Send the location information of the user to the server each time the user opens the application for sending a query.
- This module is also responsible for interacting with the user; a dialog through a terminal will enable the user to send a query in order to retrieve the desired information.
- OPTIONAL FEATURE - Ask about client location within a time frame

Communication requirements

- Asynchronous communication containing small data fields

3. Instructions

(To be updated)

4. Communication protocol

FindMe communication has two differentiated channels:

- Client to server communication
- Server to server communication

Client to server communication

Client to server communication is implemented using UDP protocol because communication is connectionless and asynchronous.

NEW CLIENT

- When connecting for the first time, it sends a zero as an ID to indicate that is a new client, once it is assigned an ID by the server; it uses it to identify itself on future queries.
- CLIENTID$TYPEOFCOM$SERVICE$<string>$LOCATIONUPDATE\n
- example: 0$\n

QUERY MSG

- The query message sent by the client is composed by 5 fields plus a terminating end-of-line to recognize when the string is terminated
- The first field is the client ID assigned previously by the server
- The second field is the type of request, it could be QUERY or UPDATE
- The third field indicates what information the string contains
- The fourth field is the information itself
- The fifth update is always the current location of the client
- CLIENTID$TYPEOFCOM$SERVICE$<string>$LOCATIONUPDATE\n
- example: 000001$QUERY$NAME$raghu$10.19.19\n

UPDATE MSG

- The update message is used when a client wants to update its own information on the system, for example, if its email address has changed.
- CLIENTID$TYPEOFCOM$SERVICE$<string>$LOCATIONUPDATE\n
- example: 000001$UPDATE$EMAIL$rahgu@aalto.fi$2890182\n

Server to server communication

Communication among servers is implemented by TCP protocol. Constant notification of updates is shared through the network and therefore connection oriented approach is required.

HOW TO SYNC WHEN ONE SERVER IS BACK ON¿

NEW CLIENT MSG

- SERVERID$TYPEOFCOM$CLIENTID$SERVICE$<string>\n
- example: 000001NEW000001$NAME$raghu$10.19.19\n

UPDATE MSG

- SERVERID$TYPEOFCOM$CLIENTID$SERVICE$<string>\n
- example: 000001$UPDATE$000001$NAME$raghu$10.19.19\n

5. Implementation description

Module 1. Data processing module

Architecture:

- Data information is stored in memory when servers are up
- Communication with other modules is carried out with message queues

In order to identify a client uniquely across all the servers, the server will generate an identifier for the client. Every server has a range of numbers and server can assign a number from that range to a client. This helps to keep the identifier unique across the servers. Once a client gets assigned by a number the same number will be used every time the client tries to contact the server.

Module 2. Server interactive module

Architecture:

- Each server is connected to its local database
- Direct connection between each server.
- Server can be a client or a server depending on who started the connection

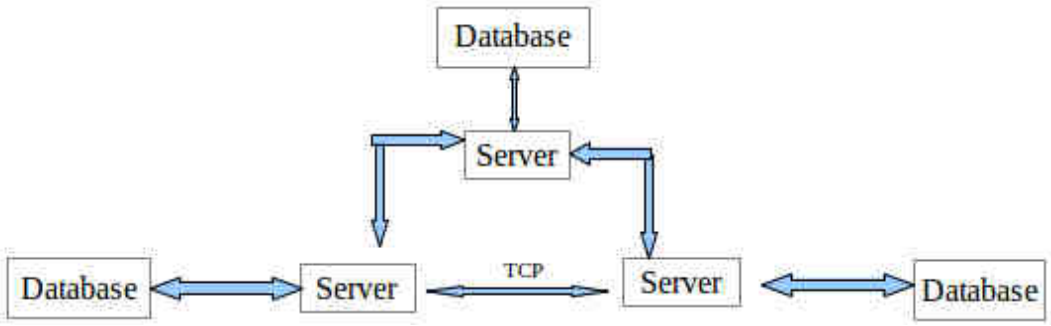


figure: sample architecture for three servers

Implementation:

- TCP is used for connection between two servers
- Servers will be configured to know all peer servers and synch operation should be restricted to only these configured peers.
- When server starts it creates connection with configured peer servers and the connection will be open as far as both servers are running.
- if a server is unable to connect to other known server or if remote server disconnects, server should wait in listening mode for incoming connections
- Severs could work in either client or server mode depending on who started the connection, if a remote peer started the connection the remote peer is the client otherwise the remote peer is the server.
- servers have a unique server id

Module 3. Client interactive module

- We are talking about a UDP server receiving requests from multiple clients.
- The number of clients can be large but the traffic exchanged is small
- No need for threads. Select with 1 thread can be enough. Max parent + 1 pre-fork child.
- We assume work in IPV6 network with no firewalls.

Module 4 - Client

What kind of server architecture better addresses the client requirements?

- The server must support sending and receiving data over UDP protocol and the data will be sent/received in string format, as long as the interface is available with this characteristics, the rest of the server architecture doesn't concern the client.

With what other modules the client shares code, features or libraries?

- The client is stand alone, it does not need to share code, memory or libraries with any other modules, due to the fact that the client runs in a separate computer, the only contact between the client and the rest of the implementation on this project is through the network sockets.

Time line execution / flow diagram
(to be updated later)

Is there any module the client needs to validate its logic?

- Yes, it needs the server to be working properly in order to validate a successful working estate, this can be tested on the last stage of the project, however, the server is not expecting to be working properly during the first two thirds of the project, because of that, a simple test server will be created to validate client behaviour, this test server does not

have any logic or any complicated structure, it just receives strings through the network socket and prints them to the screen.

- In the final integration, and before the testing part of this project is when the client needs the server to be properly working in order to be validated.

6. Quality assurance

(To be updated)

7. Known defects and other shortcomings

(To be updated)

8. Distribution of work

Implementation roles:

Server Side:

- Module 1 . Data processing module - Raghu
- Module 2. Server interactive module - Anteneh
- Module 3. Client interactive module - Jaume Benseny

Client Side:

- Complete client implementation - Raul

Work plan for phase 2:

- week 1 (2-8) – Each component develops basic features of its module and tests them.
- week 2 (9-15) – Integration of code and general test
- week 3 (16-20) – Fixing of general test small improvements.

Testing:

- Develop tests along with main code.
    - Every developed will be responsible for the testing of its own code.
- “Tester” software component
    - **We need to define someone responsible for the Tester software component.**
    - Standalone executable that executes a thorough set of tests that cover all software functionality
    - Network tests: send protocol messages in automated sequence
    - Could also include unit tests directly on functions
    - Test also high load, erroneous protocol messages, network disconnections (may need to be simulated)

Documentation:

Person responsible for the documentation is the project Manager Jaume Benseny. He is also in charge of keeping meeting notes and logs. The plan is to fill the present document as the project moves forward.

9. References