Deliverables:

* Challenge/Scenario to solve
* Description of the System
  + Components:
    - Registration Server
      * Purpose:
      * Function
    - Client
      * Purpose
      * Function
      * Interaction with Other Components
    - Test Cases
      * Purpose
      * Function
      * Interaction with Other Components
* Network System Architecture Diagram
  + Layers (UI, Network layer, DB Layer) accompanied with description.
  + Network Nodes, Edges, Databases, etc.
* Implemented Network Concepts with justification
  + Examples:  
    TCP/IP programming, implementation of network protocol, network error handling, etc.
* System testing (2 Unit and 2 Stress Test Cases)
  + Including development of tests
* Program
  + Each person is in charge of 1 layer.

Network System Engineering

[Name of the System]  
*Revision 1.0*

**Table of Contents**

[1. Introduction 6](#_Toc160524775)

[1.1 System Purpose 6](#_Toc160524776)

[1.2 Challenges 6](#_Toc160524777)

[1.3 Solution Overview 6](#_Toc160524778)

[1.3.1 Protocol Design 6](#_Toc160524779)

[2. Network System Design 6](#_Toc160524780)

[2.1 Functional Specifications 6](#_Toc160524781)

[2.1.1 Address Registration 6](#_Toc160524782)

[2.1.2 Address De-Registration 6](#_Toc160524783)

[2.1.3 Address Retrieval 7](#_Toc160524784)

[2.1.4 Client-Client Testing 7](#_Toc160524785)

[2.2 Network Components 8](#_Toc160524786)

[2.2.1 Application Domain Protocol 8](#_Toc160524787)

[2.2.2 Registration Host / Server 9](#_Toc160524788)

[2.2.3 Client 10](#_Toc160524789)

[2.2.4 Test Cases 10](#_Toc160524790)

[2.3 System Architecture 11](#_Toc160524791)

[3. System Testing 13](#_Toc160524792)

[3.1 Unit Test Cases 13](#_Toc160524793)

[3.2 Stress Test Cases 14](#_Toc160524794)

[4. Appendix A: Application Test Cases 15](#_Toc160524795)

[4.1 Template for Unit Test Case 15](#_Toc160524796)

[4.2 Template for Stress Test Case 15](#_Toc160524797)

[5. Appendix B: Screen Shots and Source Code Snippets 17](#_Toc160524798)

[5.1 SERVER\_RegistrationServer.py 17](#_Toc160524799)

[5.2 PROTOCOL\_Request.py 17](#_Toc160524800)

[5.3 PROTOCOL\_Response.py 17](#_Toc160524801)

[5.4 CLIENT\_Client\_Info.py 17](#_Toc160524802)

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Name | Reason For Changes | Date |
| 1.0 |  | Network System Engineering | dd/mm/yy |
| 2.0 |  |  | dd/mm/yy |
| xx |  |  | dd/mm/yy |

Approved By

Approvals should be obtained for project manager, and all developers working on the project.

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Signature | Department | Date |
| Jorge Ramirez de Diego | Jorge Ramirez de Diego | Development |  |
|  |  |  |  |

# Introduction

## System Purpose

## Challenges

## Solution Overview

### Protocol Design

* Application Layer:
  + A domain-specific application-layer protocol will be created for structuring our messages.
  + Binary Encoding and Decoding over the transport and network layer will be used for simplicity.
    - Sample Messages:   
      “REGISTER:{client\_info}”  
      “REQUEST:{Nickname}”
  + Both server and client require the capability of encoding and decoding the binary messages.
    - Server will take a different action depending on the received message type.
* Transport Layer:
  + TCP will be utilized for transmitting and delivering data over a network (Same Subnet).
* Network Layer:
  + IP will be utilized for sending and receiving over a separate network.

# Network System Design

## Functional Specifications

### Address Registration

* Client establishes a TCP/IP connection to the server and sends a registration request with the relevant metadata.
* Server responds with an acknowledgement message and the unique ID of the registration.

### Address De-Registration

* Client establishes a TCP/IP connection to the server and sends a deregistration request with the relevant metadata.
* Server responds with an acknowledgement message.

### Address Retrieval

* Client establishes a TCP/IP connection and sends a retrieval request.
* Server responds with the requested information over TCP. Client’s application layer must interpret the received information.

### Client-Client Testing

* Client performs various test cases over the address retrieved from the server.
  + The goal of these test cases is to retrieve information about the infrastructure of the network.

#### Test Cases

## Network Components

### Application Domain Protocol

*PROTOCOL\_Request.py*

*PROTOCOL\_Response.py*

*CLIENT\_Client\_Info.py*

* **Purpose:**
  + Provides a framework for application-application communication between Host and Clients.
  + Enables Protocol Users to simply convert domain objects for transmission as serialized and encoded objects through TCP.

#### System Components

* **Components:**
  + Request
    - Types of requests all users of the Application Protocol should be capable of handling or transmitting. Standardization allows for Protocol to be utilized in multiple domains.
    - Structure:
      * RequestHeader
        + RequestTypes:

NULL = 0

REGISTER = 1

DEREGISTER = 2

RETRIEVE = 3

RETRIEVEALL = 4

* + - * RequestBody
        + Any String or Serialized Object
    - Serialized Format:
      * “{RequestHeader Type}:{RequestBody String}”
  + Response
    - Types of responses all users of the Application Protocol should be capable of handling or transmitting. Standardization allows for Protocol to be utilized in multiple domains.
    - Structure:
      * ResponseHeader
        + ResponseTypes:

NULL = 000

SUCESS = 100

BAD\_REQUEST = 200

SERVER\_ERROR = 300

* + - * ResponseBody
        + Any String or Serialized Object
    - Serialized Format:
      * “{ResponseHeader Type}:{ResponseBody String}”
  + Client\_Info
    - Domain-specific object standardized across client and server. Both are capable of handling the object as required.  
      Provides all information required for Client->Client communication.
    - Structure:
      * Dictionary {‘Nickname’, ‘IP’, ‘Listening\_Port’}
    - Serialized Format:
      * {‘Nickname:””,  
        ‘IP’:””  
        ‘Listening\_Port:””}

### Registration Host / Server

*SERVER\_RegistrationServer.py*

* **Purpose:**
  + Acts as a central hub for storing and providing host client registrations to requesting host clients. Its intermediary role aims to allow for “consensual” device discovery.
* **Constrains:**
  + The server must be hosted on a static IP so that clients can know where to find it. The server’s address will be hard coded into the clients.
    - Could be solved by using DHCP for local clients.

#### Core Functionalities

* **Functionalities:**
  + A client provides the required information for performing a registration in the host. The host then stores the provided data for future use.
  + Client provides the registered devices information on request.
  + A client can deregister from the host.
  + Must be able to encode and decode UTF-8 Strings into/from binary.
  + Capable of storing and retrieving information from the assigned Storage File.

#### System Components

* TCP Socket
  + Simple TCP Socket that enables any Client from sending an encoded request.
* File-Based Storage
  + Server writes and stores and retrieves Client Registrations (REGISTER requests) from the file.
    - Format (JSON):

{

“Nickname”: {

“Nickname”: "",

“IP”: "",

“Listening\_Port”:””

},

“Nickname”: {

“Nickname”: "",

“IP”: "",

“Listening\_Port”:””

}, …

}

* Server Application
  + Decodes any request received by the socket in UTF-8 and handles the request accordingly.
    - If format is incorrect, error is sent to sender.
    - Response resulting from the request is sent to sender.

### Client

#### Core Functionalities

#### System Components

* **TCP Client:**

Each client will implement a TCP client that is capable of establishing connection with the server.

### Test Cases

* Test Case:
  + Purpose
  + Functionality
  + Implementation

## System Architecture

The system should provide the capability of accessing desired target clients over a subnet. This will be achieved via the use of public Ips when dealing with inter-network communication, in which case the router device will act as a Layer 3 device (a router), and intra-network communication, in which case the router device will act as a Layer 2 device (a switch).

A computer screen shot of a computer server

Description automatically generated

Figure 1 - Application's Network Diagram

The procedure for sending messages between client and server is well standardized. Any message type will follow the same basic procedure, simplifying any future development of program expansion.

A black background with white text

Description automatically generated

Figure 2 - UML Sequence Diagram - General Procedure

Given the objective of the application, the procedure for obtaining a target client is also well defined. It is up to the test case to then specify exactly what the description of the test will be.

A black screen with white text

Description automatically generated

Figure 3 - UML Diagram - General Testing Procedure

# System Testing

## Unit Test Cases

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Project Name: |  | Test Designed by: | Jorge | Ramirez de Diego | |
| Module/Component Name: | *SERVER\_RegistrationServer.py* | Test Designed date: | 05 | 03/2024 | |
| Release Version: | 1.0 | Test Executed by: | Jorge | Ramirez de Diego | |
|  |  | Test Execution date: | 05 | 03/2024 | |
|  |  |  |  |  |  | |
| Pre-condition: | RegistrationServer class and its dependencies (‘Client\_Info’, ‘Request’, ‘RequestHeader’, ‘RequestBody’, and ‘RequestTypes’) are correctly implemented and are available.  Python ‘unittest’ framework is used for the Unit Test.  Python ‘unittest.mock.patch’ replaces ‘socket.socket’ class to simulate network interactions without an actual socket communication. | | | | | | |
| Postconditions: | Temporary JSON file (‘temp\_test\_clients.json’) is created at the start and removed after the test.  Test environment is cleaned up and returned to the same state as before it was ran. | | | | | | |
| Test Steps: | 1. ***Mock Setup:*** ‘socket.socket’ objects are placed with ‘MagicMock’ objects to simulate client-server interaction. 2. ***Simulate Client Registration Request:*** Create a ‘Request’ object that mimics a client registration request, which would include the client information (nickname, ip, and listening port). 3. ***Invoke Server Handling:*** Call the ‘handle\_client’ method of the ‘RegistrationServer’ through the mocked socket to process the registration request. 4. ***Verification:*** Check the server’s assigned storage (‘temp\_test\_clients.json’) to verify that the client’s information was added. | | | | | | |
| Test Data: | * Client Nickname: “TestUser” * Client IP: “192.168.1.1” * Client Listening Port: “5005” * Server IP: Default Value (localhost or local/public IP Address is Assigned by Switch/Router) * Server Listening Port: Default Value (9999) * Server Storage File: ‘temp\_test\_clients.json’ | | | | | | |
| Expected Result | * Client’s information is successfully added to the server’s assigned storage.   + Message "TEST SUCCEEDED: TestUser is found in data file" printed on success.   + Message "TEST FAILED: TestUser is not found in data file" printed on failure. | | | | | | |
|  |  | | | | | | |
| Test Case# | Test Title | Test Summary | | Expected Result | |
| 1 | Unit Test Implementation Test | Test ran to ensure that the Unit Test was implemented correctly. | | Successful (OK) | |
|  |  |  | |  | |
|  |  |  | |  | |

## Stress Test Cases

# Appendix A: Application Test Cases

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Template for Unit Test Case | | | | | |  |  |
|  | Project Name: |  | Test Designed by: |  |  |  |  |
| Module/Component Name: |  | Test Designed date: |  |  |  |  |
| Release Version: |  | Test Executed by: |  |  |  |  |
|  |  | Test Execution date: |  |  |  |  |
|  |  |  |  |  |  |  |  |
| Pre-condition |  | | | | | | |
| Postconditions: |  | | | | | | |
|  |  |  |  |  |  |  |  |
| Test Case# | Test Title | Test Summary | Test Steps | Test Data | Expected Result | Status  (Pass/Fail) | Notes (if any) |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Template for Stress Test Case | | | | | |  |  |
|  | Project Name: |  | Test Designed by: |  |  |  |  |
| Module/Component Name: |  | Test Designed date: |  |  |  |  |
| Release Version: |  | Test Executed by: |  |  |  |  |
|  |  | Test Execution date: |  |  |  |  |
| Test Case# | Test Title | Test Summary | Test Data | Breakout Point | Safe  Point | Status  (Pass/Fail) | Notes (if any) |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

# Appendix B: Screen Shots and Source Code Snippets

## SERVER\_RegistrationServer.py

[](https://github.com/JRamirezDD/Term-Project/blob/main/Program/SERVER_RegistrationServer.py)

<https://github.com/JRamirezDD/Term-Project/blob/main/Program/SERVER_RegistrationServer.py>

## PROTOCOL\_Request.py

[](https://github.com/JRamirezDD/Term-Project/blob/main/Program/PROTOCOL_Request.py)

<https://github.com/JRamirezDD/Term-Project/blob/main/Program/PROTOCOL_Request.py>

## PROTOCOL\_Response.py

[](https://github.com/JRamirezDD/Term-Project/blob/main/Program/PROTOCOL_Response.py)

<https://github.com/JRamirezDD/Term-Project/blob/main/Program/PROTOCOL_Response.py>

## CLIENT\_Client\_Info.py

[](https://github.com/JRamirezDD/Term-Project/blob/main/Program/CLIENT_Client_Info.py)

<https://github.com/JRamirezDD/Term-Project/blob/main/Program/CLIENT_Client_Info.py>