**NodeRunner Network Abstraction**

<https://github.com/arcpu-net786/NodeRunner>

**1. Introduction and Description**

This document is a comprehensive guide to the NodeRunner Network Abstraction programming framework. It is a simple TCP client-server protocol written entirely in Python and its associated frameworks and libraries. Upon initialization, users (or their corresponding programs) will be able to communicate with each other by sending messages without programming any client/server code.

A picture containing text, design

Description automatically generated

Figure 1. The connections between each abstracted node in the network.

**2. Required Packages and Software**

This project should run on any platform that is able to support the following packages:

* Python version >= 3.6

No display is needed. However, an internet connection run by the TCP/IP protocol is needed in order to have multiple nodes communicate. Additionally, the gateway must allow incoming public traffic to a specific device in order to communicate between devices on different subnets. If you use a cloud service instance like AWS EC2 or a GCP compute engine, make sure that the instance’s security protocol allows incoming and outgoing TCP traffic.

**3. Running the Code**

Run the following command in a Linux (or Mac) terminal to initiate a node:

python3 node.py

Running the above command will prompt the user to input an IP address, port, and a name. Enter the server’s public IP address as the IP address. By default, this will resolve to 0.0.0.0. Enter any unreserved port number for the port. Finally, enter a string that is unique among all parties in the communication protocol.

A screenshot of a computer code

Description automatically generated with low confidence

Figure 2. The terminal display of the initialization and connection between node1 and node2.

Figure 2 above demonstrates how two nodes can communicate with each other immediately after initializing and connecting. The top terminal is for node1, and the bottom terminal is for node2. node1 initializes at public IP address 18.221.21.36 and port 55555. node2 initializes at public IP address 3.27.131.196 and port 55556.

**4. Interacting with the Program**

There are only two commands in this protocol: *connect* and *write*.

connect [IP address] [Port] – bidirectionally connects the current node to the node at the specified IP address and port.

write [node name] [text] – sends the written text to the specified node.

Once a node has been connected to a different node by specifying an IP address and port, subsequent communication can proceed by just specifying the name of the node, which is sent as a connection confirmation. A ‘*connect’* request can only be done to a previously unconnected node once. Subsequent communication can only be ‘*write’* requests.

**5. Architecture Principles**

A traditional client-server connection treats the client and the server as separate entities, such that the server is a foundational program that binds to the system’s port and IP address. Consequently, the client is a program that is detached from the system’s port and IP address and instead connects to the server by referencing the port. As a result, multiparty communication architectures can be constructed by implementing a singular centralized server that is connected to multiple clients. The server’s job is to accept all communication and disseminate any messages sent by one client to all clients. This is because clients cannot connect to other clients directly. A picture containing text, screenshot, diagram, number

Description automatically generated

Figure 3. Traditional Client-Server Architecture vs. Abstracted Client-Server Architecture

What is actually going on? Truly, only the server is bound to the port and IP address. Therefore, the goal is to effectively have servers talk to each other. However, they cannot connect to another server without binding to an IP address, which is why their client counterpart is needed. A single “home” client connects to the server and is treated as the affiliated client. The home client is only one of the clients connected to the server, just with the special property that it gets all the info that the server gets. Clients only connect to servers. Therefore, all other nodes that want to talk to a node, will have the node’s client component connect to the server component of that node. The server shares the information that it received to the node’s affiliated client. The node’s client can then respond by communicating to the other node’s server component, which in turn shares its message with its node’s client component. This gives the abstraction of a single node entity talking to a different node entity.

A diagram of a server

Description automatically generated with low confidence

Figure 4. Client-to-Server communication protocol under-the-hood.

**6. Credits and Acknowledgements**

First and foremost, this work would not have been possible without the decades of networking research and open-source projects that are available for students to learn.

[1] A general overview of TCP/IP protocol was derived from the Spring 2022 iteration of CS 168 at UC Berkeley lecture videos:

<https://cs168.io/>

This project is also open-sourced for education purposes, in the GitHub link at the top of this document.