DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master Practical Course
Computer Network Simulation

14.12.2022

Assignment 2
Part 2 - Implementation

Group 2

Zafer Tan Cankiri
Erdem Ege Marasli

Description of the Application:

We designed a robust messaging app for people stuck in bunkers after a catastrophe like a nuclear warfare.

The application has two major aims as follows:

1. Finding the location of a person in a bunker.
2. Communicating with the other people in the bunkers via text messages.

The application runs on an infrastructure that connect multiple bunkers to each other via Ethernet connections. Each bunker has a bunker router that acts as a gateway between the people in that bunker and the other bunkers.

There is a main router that has connection to all of the bunkers. All of the traffic between bunkers goes through this main router. There is a server connected to the main router, which acts as a rendezvous point.

At regular intervals, each person sends a HeartBeat signal to the server automatically. The server understands that this person is still at the given bunker and notes to its database.

When someone wants to learn if the desired person is at one of the bunkers, it sends a lookup request to the server. When the server receives this lookup request, it checks its database. If the server finds a valid record in its database for the desired person, it sends back a positive response with the desired person's location. Otherwise, it sends a negative response. When the user gets a positive response from the server, it saves all the information to its address record for future use.

Users can also try to send a text message to another user. When someone tries to send a message, it first looks at its address book. If the user has the information about the receiver in its address book, it directly sends the message to the given IP address in the address book. Otherwise, it sends a lookup request to the server to learn if the receiver is at one of the bunkers. If the user receives a positive response from the server, it saves the information to its address book and sends the text message to the given IP address.

Implementation Details:

There are 3 main applications in our scenario:

1. HeartBeatApp
2. ServerApp
3. ClientApp

HeartBeatApp:

HeartBeatApp works on all hosts. This app basically sends HeartBeat packets to the server periodically as mentioned above. For simulation purposes, this periodic interval is given randomly to each host.

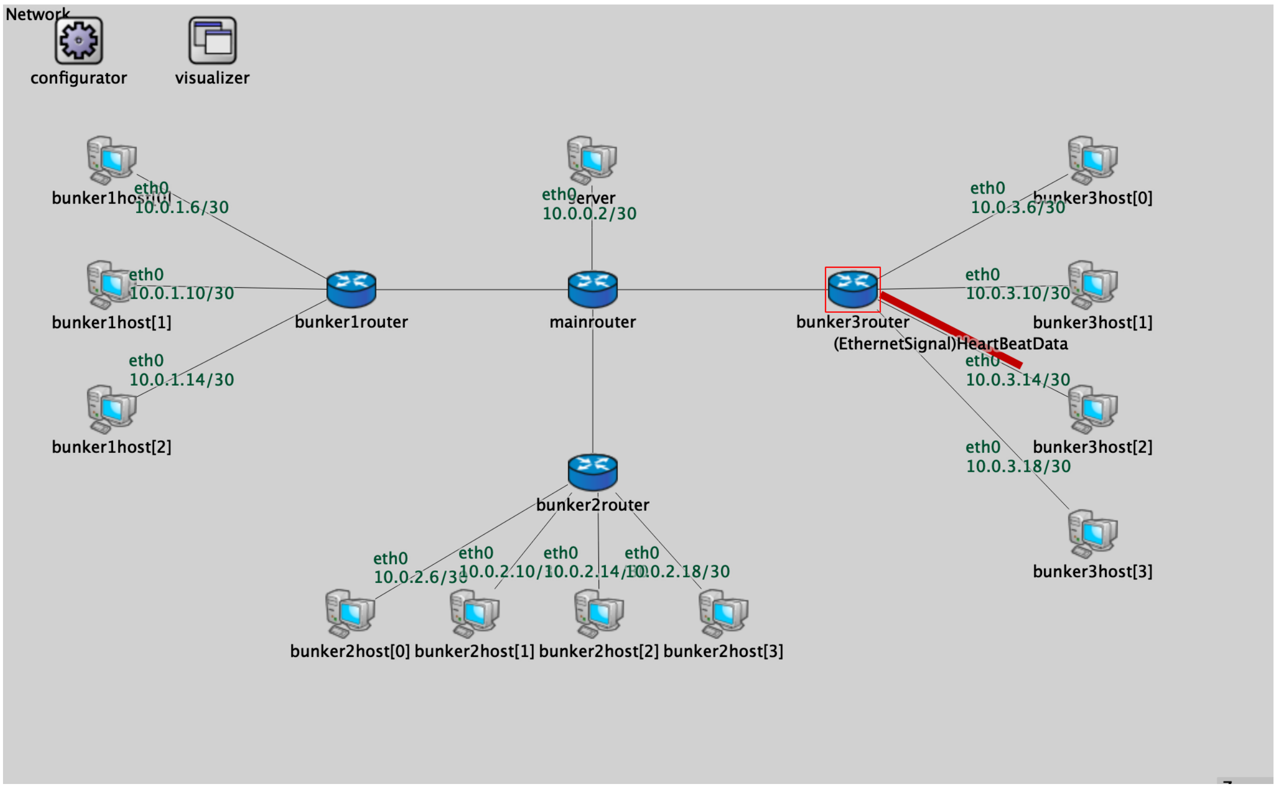

Figure 1: Bunker3host[2] sending an HeartBeat request to the server

ServerApp:

ServerApp runs only on the main server. This app resolves lookup requests and receives heart-beat signals. It has a database to store the locations, IP addresses, and timestamps of the users. Locations and IP addresses are gathered by the heartbeat packets which are sent by the users. The timestamp of a record is updated each time the server gets a heartbeat from the corresponding host. According to that timestamp, the server can track if the user is still in the bunker or not. If the server cannot get a heartbeat signal for a long time from the user, it deletes its record from the server database. According to the database, the server resolves lookup requests come from the users.

ClientApp:

ClientApp works on all hosts. This application sends/receives text messages in various (random) packet sizes to/from users and sends lookup requests to the server as mentioned above. A client can try to send a text message or a lookup request at a random time.

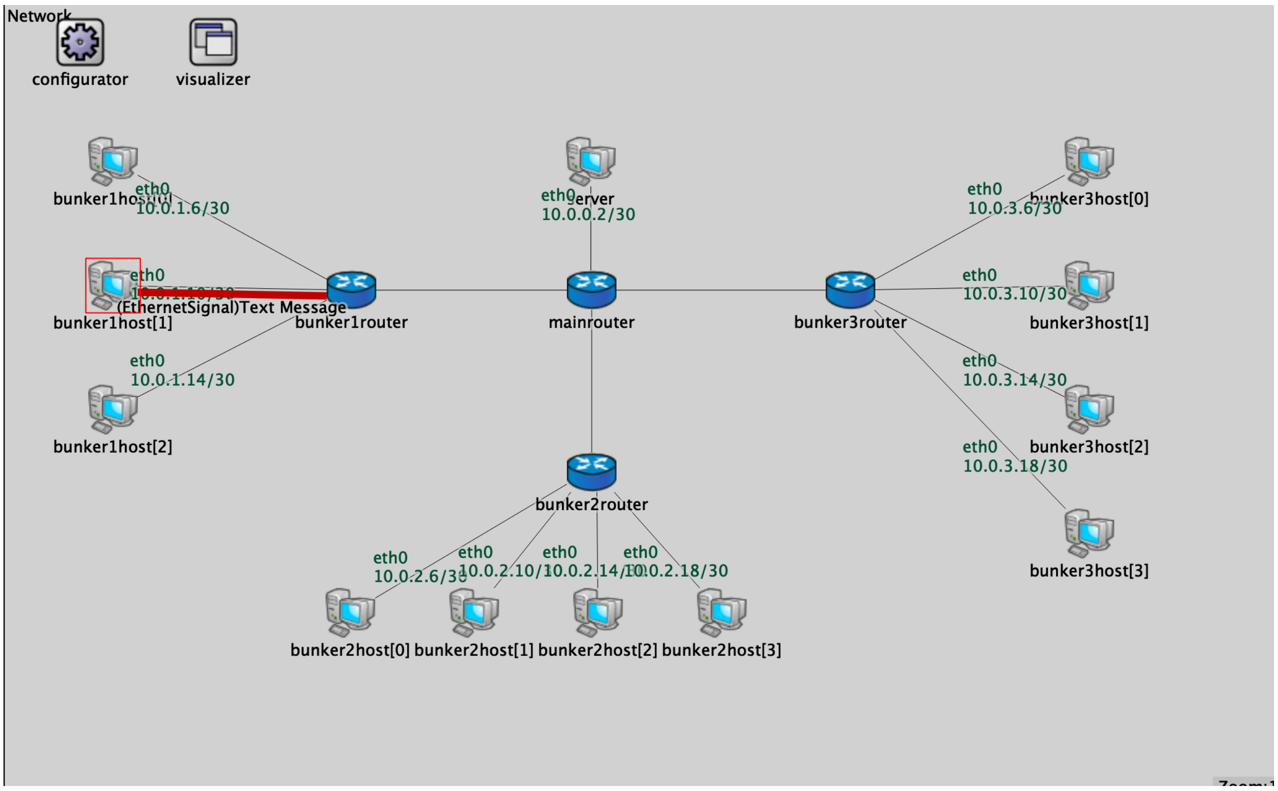

Figure 2: Bunker1host[1] sending an Text Message to Bunker3host[0]

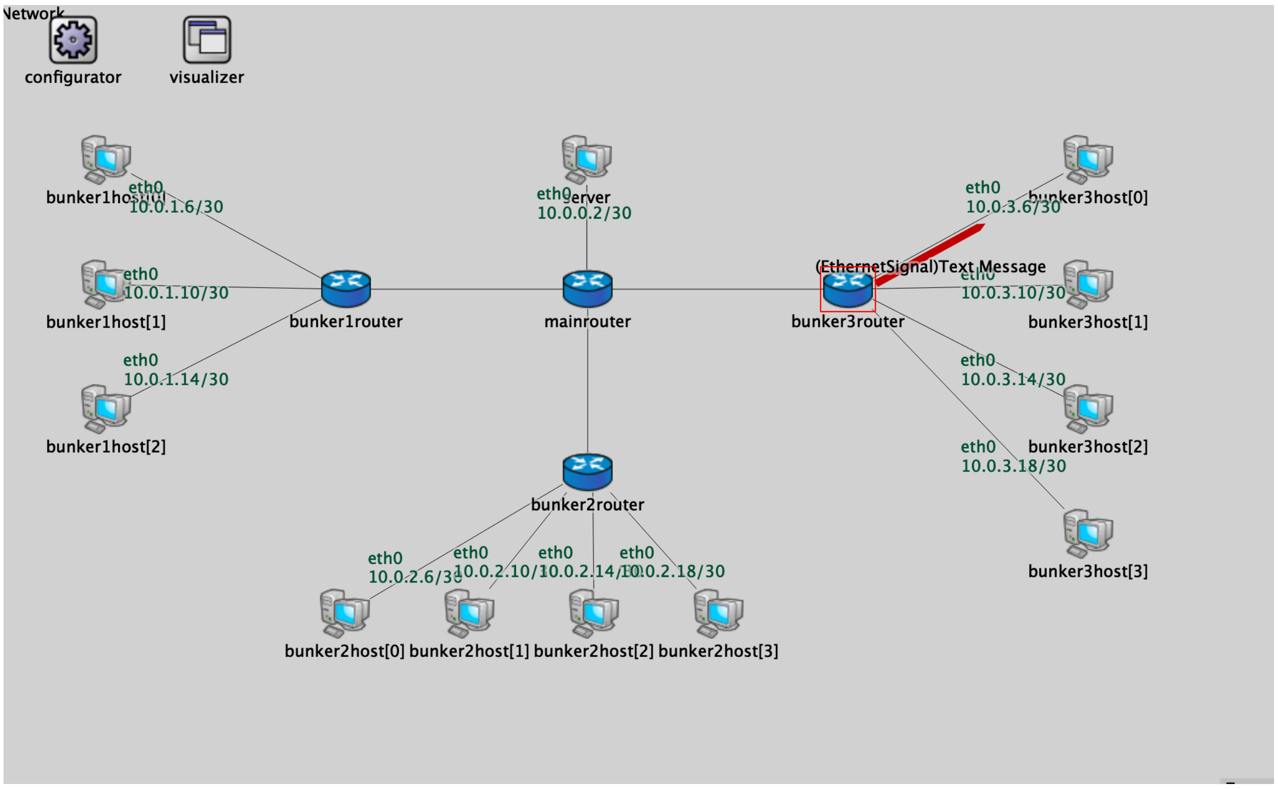

Figure 3: Bunker1host[1] sending an Text Message to Bunker3host[0]

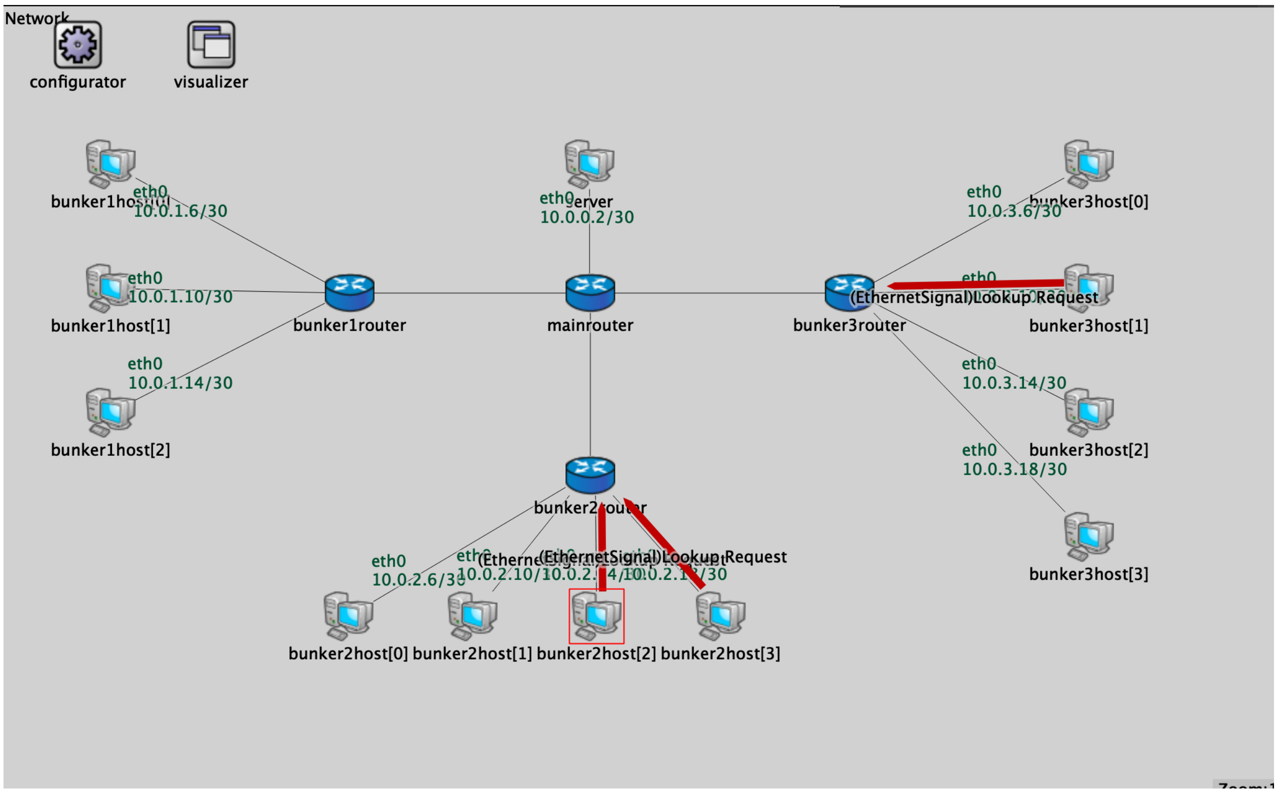

Figure 4: Bunker3host[1] sending an Lookup request to the server

Updated Network Conditions:

1. From now on (Assignment 2), the number of bunkers is also dynamic and can be set from the INI file. Bunkers are created dynamically and each bunker gets a bunker router that is connected to the main router directly. The hosts are also distributed to the bunkers dynamically.

2. The cables that connect the nodes became modifiable. Their lengths and bandwidths can be set with parameters.

3. Each router gets a limited size of buffer for packet processing. This buffer size can be set via parameters.

TCP Implementation:

1. In lieu of utilizing UDP for our application, we commenced the development of a TCP-based implementation. The current iteration can be found in the "src/tcp" directory.

2. In the INET examples, we discovered implementations of "TcpEchoApp", "TcpSessionApp", and "TcpBasicClientApp". We attempted to apply similar methodologies to our project, but due to its heightened complexity, significant modifications were required. As such, we have decided to suspend the TCP implementation for the time being, as it does not impart any notable benefits to the application. Our application primarily employs small textual messages that can be accommodated by individual UDP packets, rendering the need for sequencing unnecessary. Furthermore, the wired network infrastructure of our system reduces

the likelihood of data corruption during transmission, obviating the necessity of transmission control (reliability). Additionally, in forthcoming stages, we plan to expand our server capacity through the addition of multiple server instances. A database synchronization mechanism for these servers will necessitate a multicast/broadcast-based discovery mechanism, which will necessitate the utilization of UDP.

3. However, we are also contemplating potential scenarios in which we may transition to a wireless infrastructure, and therefore TCP remains under consideration for its reliability features.

4. Ultimately, we may opt for a UDP/TCP hybrid solution.

Data Collection Pipeline Implementation:

1. We have developed a Python script, dubbed "pipeline.py", to manage the pipeline.

2. This script reads the "omnet.ini" file and employs regular expressions to parse it and extract the defined configuration names.

3. The script subsequently executes each configuration by running our simulation in Cmdenv mode with the parsed configuration names in sequence.

4. The script subsequently invokes "opp_scavetool" with the requisite parameters to convert the vector and scalar files generated by each run into JSON files.

5. The script parses the JSON files, extracts the required data, and reorganizes it into a more useful data structure, subsequently saving it as a clean JSON file.

6. The script also performs aggregation tasks, such as averaging collected metrics, and generates plots of these metrics using mathplotlib.

7. These operations are performed iteratively for all configurations.