Self-Healing Network Using ORBIT Testbed

Final Report

Team 36
Client and Advisor: Professor Mohamed Selim
Team members
Matthew Johnson
Amber Chin
Parnumart Hanthiradej
Adam Tiedman
Yuin-Choon Ng
Team Email: sdmay21-36@iastate.edu
Team Website: https://sdmay21-36.sd.ece.iastate.edu

Revised: 4/25/2021

Executive Summary

Development Standards & Practices Used

- Software Development Standards
    - C Coding Experience to design and implement the algorithm
    - Linux Proficiency to operate testbed servers
- Engineering Standards
    - IEEE 1703-2012
    - RFC 768
    - ANSI C12.22
- Networking Knowledge
    - Knowledge of wireless protocols
        - UDP
    - Knowledge of wireless nodes
        - WLAN
        - Support Servers
        - Interfaces
        - Network Address
    - Knowledge of network testbeds
        - ORBIT(Wireless Testbed)

Summary of Requirements

- The network needs to be able to:
    - Detect an outage
    - Diagnose the outage
    - Compensate for the damage while attempting a fix
    - Ensure server node are not overloaded beyond their bandwidth
- The central controller should be able to assess the network state with limited network congestion

Summary of Engineering Standards

- IEEE 1703-2012
  - Local Area Network/Wide Area Network (LAN/WAN) Node Communication Protocol
- ANSI C12.22
  - The American National Standard for Protocol Specification for Interfacing to Data Communication Networks
- RFC 768
  - User Datagram Protocol which can be described as the fundamental component of the internet protocols and standards.

Applicable Courses from Iowa State University Curriculum

- CprE 489
- CprE 308
- CprE 431
- CprE 430

New Skills/Knowledge acquired that was not taught in courses

- Network Design with more connected nodes
- C Coding language
- Writing Linux shell scripts
- Routing ethernet networks
- Interacting with ORBIT Testbed (OMF commands)
- User Datagram Protocol

Table of Contents

List of figures/tables/symbols/definitions (This should be the similar to the project plan)

5G Network: The 5th generation of mobile networks. This new standard of mobile network began its launch in 2018.

GENI Platform: Testbed that uses many virtual machines to simulate a network environment

Orbit Platform: A radio grid with multiple nodes used for communication testing and development

Self-Healing: A process of diagnosing and resolving without the need for human interaction

1 Introduction

1.1 Acknowledgement

We would like to acknowledge our current advisor is Professor Selim and we would like to thank him for helping guide us through the early stages of our project. He has provided us with large amounts of background knowledge as well as helped guide our research and overall project.

1.2 Problem and Project Statement

Current cellular networks will portion out a large amount of their budget to spend on repairing and resolving network problems and outages. The current method involves relying on human expertise to identify, diagnose, and resolve any issues with the network. This process has proven to not only be very costly, but also presents significant challenges in regards to the speed at which resolutions can be made. With 5G networks rolling out, the complexity and overall cell density of these networks will prove to be too much to handle with the current processes. To be able to keep next generation networks running, a new solution for resolving network issues must be created.

Our system can utilize an algorithm to detect a full or partial outage, diagnose the cause, and compensate for the outage. Using an algorithm to solve these issues will cause minimal user downtime during an outage while keeping the cost to maintain the network much low.

1.3 Operational Environment

The chosen testing tool to simulate a wireless network is a radio testbed called ORBIT. The ORBIT testbed we are using is sandbox9. This testbed has 7 nodes that we are able to configure. Each node in the testbed is connected to every other node but can have its interfaces configured so that a simulated network can be constructed. The network setup that was used for testing consisted of one central controller node, three server nodes, and three client nodes.

1.4 Requirements

Functional :

- The network needs to be able to detect an outage, diagnose the outage, and compensate for the damage while attempting a fix. This requires a detection and mitigation time under a second.
- The self-healing process should make sure network nodes are not overloaded beyond their maximum available bandwidth.
- The central controller should be able to assess the network state with limited network congestion.

Non - functional :

- Economical :
    - Free ORBIT platform to implement self-healing project.
- Environmental
    - The project should be functional across open-access networks.

1.5 Intended Users and Uses

Intelligent networks perform a variety of roles which improve our lives. For these networks to be efficient, they must be reliable.

Self healing networks are used when there is a failure in the network. The self healing network can resolve the problem without humans involved. It is a tool which can detect, remediate outages, failure and breaches.

The main user base of this is mobile network companies and users. These mobile network users should experience minimal loss of service due to partial outages.

1.6 Assumptions and Limitations

Assumptions:

- The end users of neighbouring base stations will not be affected by base station failure before the self-healing process.
- There will be no presence of interference issues.
- The algorithm of the project must have the functionality of being able to decide which base station can be involved in the healing process.

Limitations:

- The financial cost for this project has to be none, since we are using free platforms to conduct experiments on this project.
- The software development standards for this project should be restricted to using C and shell script for coding to implement the self-healing algorithms on the testbeds and linux to operate the testbed servers.
- Because of a lack of device drivers, we will not be able to wirelessly implement our design, we must therefore use the wired connections present in the ORBIT testbed to perform our algorithm

1.7 Expected End Product and Deliverables

The expected end product in the first semester is to implement a simple self-healing algorithm between two access points with a single user.

The expected end product in the second semester is to implement a more complicated self-healing algorithm which contains three servers and three clients. Every node contains three interfaces and each interface contains a different network address. Besides, a UDP socket is configured on each node to do the self-healing. The central controller will monitor the status of each node, if any server or client misses a check in, an alert will be

triggered, and compensation will be carried out.

2 Project Plan

- 2.1 Task Decomposition

First Semester

- Research Orbit Platform
  - Begin testing on simulation platform Geni
  - Run test scripts on real Orbit platform devices
- Research Self-Healing Algorithms
  - Do a research search for important self-healing algorithm descriptions
  - Break these topic down into important concepts
  - Choose desired algorithm and map out key block diagram
- Implement a self-healing algorithm on GENI
  - Code algorithm through Linux shell with simple client and server
- Implement a self-healing algorithm on Orbit platform
  - Code algorithms in Python
  - Implement algorithm on two access points
  - Implement algorithm on real-world outdoor test network
- Assess self-healing algorithm
  - Assess quality of chosen algorithm implementation
  - Decide if this algorithm meets our desired benchmarks
  - Make necessary changes to algorithm and retest
- Project Presentation
  - Prepare presentation on outcome of self-healing algorithm implementation
  - Give presentation

Second Semester

- Find the alternative platform that can work similar to GENI
  - Tried using WITestlab, but failed to due to Covid-19
- Working with ORBIT tutorials
  - Tried working with the wifi tutorial but failed to because there were 2 different omf versions on ORBIT, which are 5.4 and 6
- Discuss about ORBIT tutorials with ORBIT creator/developer
  - Met up with Ivan Seskar from Rutgers University for further insights with the ORBIT Testbed
- Implement a self-healing using UDP on ORBIT
  - Code the algorithm for the UDP client and server with the C programming language

    - Implement the algorithm based on the network topology consisting of 3 servers and 3 clients.

- Assess self-healing algorithm

    - Assess quality of chosen algorithm implementation
    - Decide if this algorithm meets our desired benchmarks
    - Make necessary changes to algorithm and retest
- Project Presentation
    - Prepare presentation on outcome of self-healing algorithm implementation
    - Give presentation

2.2 Risks And Risk Management/Mitigation

Implement self-healing algorithm on Orbit platform:

The Orbit platform may not work well for our implementation of a self-healing algorithm. Because this platform is well known and used for applications such as ours, the probability of this is fairly low at around .2. If this is the case we may need to select a different algorithm, or we may need to choose a different platform such as Geni.

Assess self-healing algorithm

The second risk factor is that our algorithm may not meet our expectations. This is fairly likely at about .6. If this is the case, then our plan will be to go back to the research stage and choose an algorithm with the knowledge gained through our first implementation. We would then need to go through the implementation task again, finally reaching the assess task once more. This risk can be slightly mitigated through a more lengthy research process, however, it can never be eliminated until we actually test the algorithm on the platform. We have factored into our timeline that this may occur, so the loss in time is not drastic.

2.3 Project Proposed Milestones, Metrics, and Evaluation Criteria

Research Orbit Platform :

The research on Orbit platform has been carried out. It takes 2 to 3 weeks to receive a response from the organization. Because of this, our team has planned to use another platform called GENI which is much similar to the Orbit platform. On GENI, team members are allowed to create their own slices (experiments) with different nodes (server or client).

Research about what self-healing is:

Our team has been doing a lot of research and reading IEEE articles on what self-healing is and also presenting powerpoint slides to our advisor to get feedback or additional information from him biweekly. We learnt that the most important thing about self-healing is that the end user should not have to experience any network failure from the base stations. Self-healing is a fundamental process in any network.

Implement a Self-Healing algorithm on GENI:

The implementation of the self healing algorithm on GENI platform included the use of several virtual machines set up to emulate a cell network. We were successful in writing a shell script that would detect an outage and reroute the client node so that a successful connection could once again be established. GENI allowed for a controlled environment that could be easily manipulated and would be very reliable.

Implement a Self-Healing algorithm on ORBIT:

The implementation of a self healing network on the ORBIT platform will be the first real world test of the developed algorithm. Using the ORBIT platform will be a step above the GENI tool as it allows for actual physical nodes rather than the use of virtual machines. With ORBIT the opportunity for tests including variables such as radio noise is now possible.

2.4 Project Timeline/Schedule

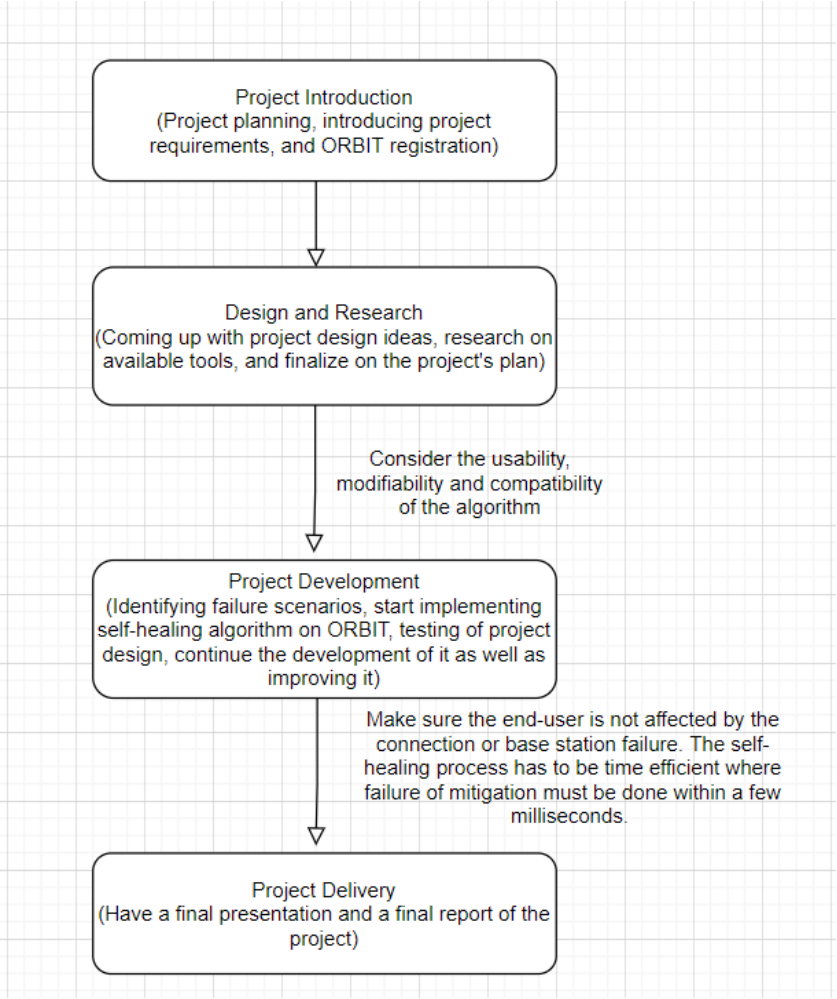

Figure 1

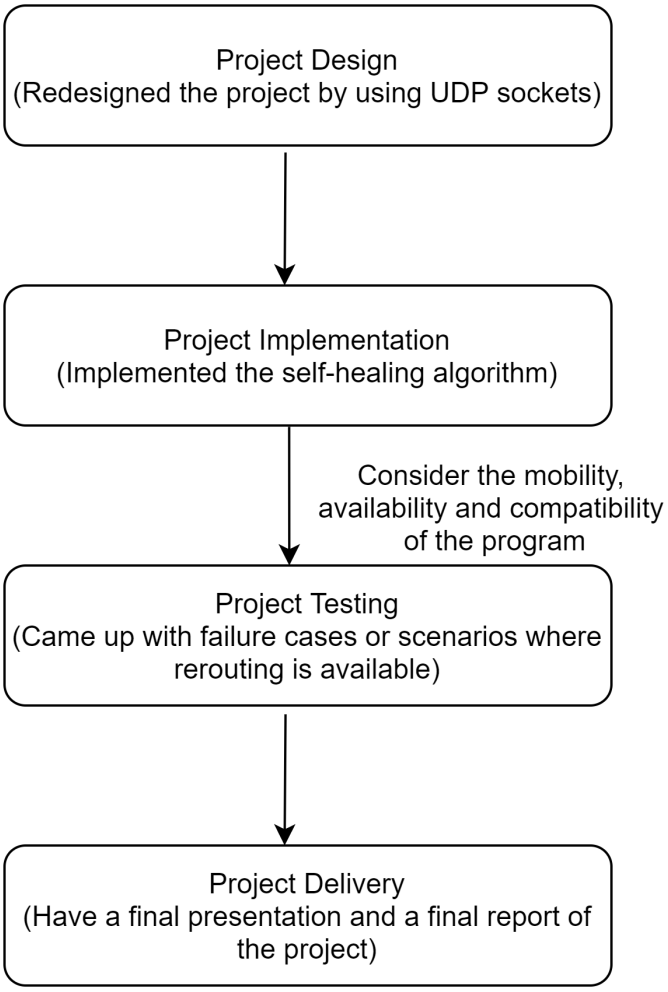
Project Design
(Redesigned the project by using UDP sockets)
Project Implementation
(Implemented the self-healing algorithm)
Consider the mobility, availability and compatibility of the program
Project Testing
(Came up with failure cases or scenarios where rerouting is available)
Project Delivery
(Have a final presentation and a final report of the project)

Figure 2

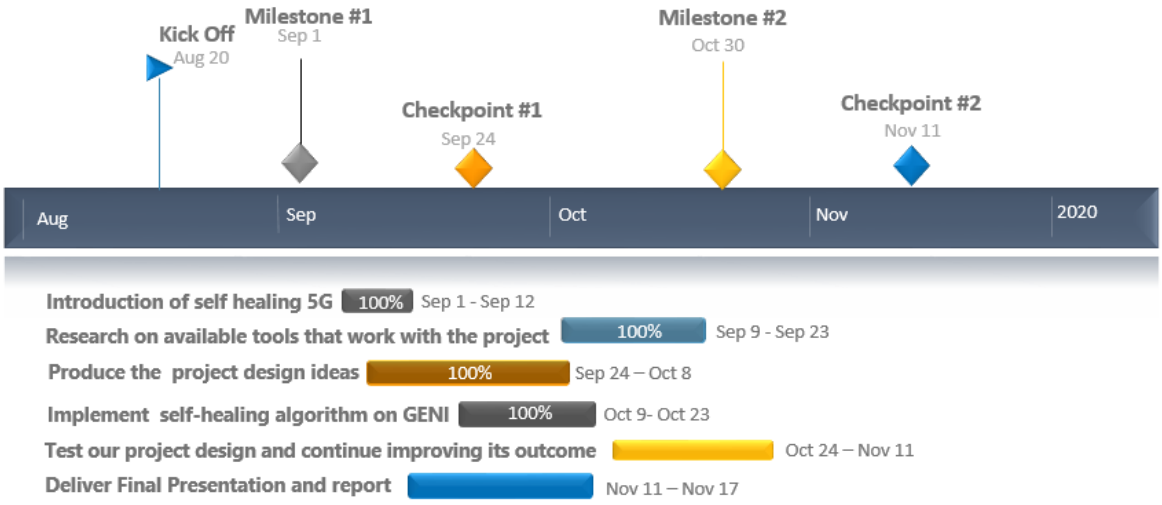

Figure 3

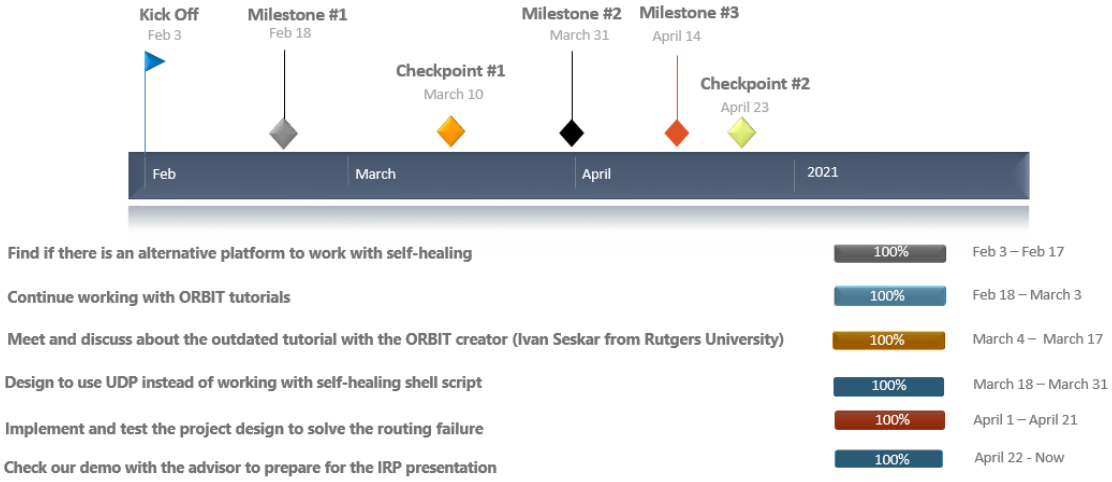

Figure 4

The flowchart portrays the project's timeline for two semesters. First and foremost, we are going to start off with a project introduction where we learn what requirements are needed in terms of skills, knowledge and experience that we can contribute to this project. We also need to register for accounts in the free platforms which are GENI and the ORBIT

testbed. In this stage, project planning is an essential task as it ensures that we are on track in this project.

After this, we will proceed with designing and doing a lot of research on the project. We will have to come up with project design ideas, do research on the available resources to make this project successful and then finalize on the project's plan. The project design ideas are crucial because it has to include the project's functional requirements which are the effects of a network outage on network users should be mitigated and the self-healing of the network should be time efficient where the failure mitigation must be done within a few milliseconds. Other than that, usability, modifiability and compatibility of the algorithm has to be ensured. The algorithm should be usable for most 5G network applications, modifiable to suit specific network needs and compatible with common network node hardwares.

Next is the program development where we start implementing the self-healing algorithm on the ORBIT testbed, identifying failure scenarios through experiments. testing the project's design and continuing the development of it as well as improving it. This stage is fundamental in the process of the project because this is where the most progress can be viewed and certain adjustments or modifications can be made in our project. The testing for this project would include both virtual and real world tests. This stage is also where we can fix mistakes and enhance the algorithm's efficiency and functionality. We have to make sure that the end-user is not affected by the connection or base station failure. The self-healing process has to be time efficient where the mitigation failure must be done within a few milliseconds.

In the second semester, we have experienced that ORBIT documentation is not updated as the ORBIT version has been updated to a new version. Due to this problem, it caused our team to not be able to do any useful experiments about self-healing algorithm. Therefore, our team found a new online platform, WITestlab which can be used for network experiments. Unfortunately, account registration is not allowed during that time because of COVID protocol.

Next, we decided to go back to ORBIT platform to start working on some simple tutorials. At the same time, we contacted an ORBIT expert, Ivan Seskar from Rutgers University to seek for the old version of ORBIT documentation. During this period, Ivan not only provided us with some useful documents about ORBIT but also guided our team on how to configure host-apd and acp on nodes.

After that, we stop using shell scripts on self-healing algorithm and instead use UDP sockets with executables that run on each node to do our self-healing algorithm. The reason is because the complexity of self-healing algorithm increases, this caused data manipulation and calculations are unmanageable. Besides, we have expanded our network topology, where we currently have a central controller, three clients and servers. From

here, we implemented the self healing algorithm where we have failure cases such as the downstream routing failure and the direct node failure. Routing starts here.

2.5 Project Tracking Procedures

We're using the GENI portal created by the professor to keep track of our Self-healing-5G project. In the GENI portal, we can decide who will be the project leader, leading other members in presenting the project status to the professor each week. All members can create slices for project explanation and also be able to check the update of the project's process. In ORBIT we have a shared code base that all members can access, testbeds are shared between the entire group so all members can access and see current test progress and code progression.

2.6 Personnel Effort Requirements

Individuals need to have a basic knowledge of coding on Linux commands and C code to implement Self-Healing algorithms on testbed and operate testbed servers. During the first semester, all members need to learn how to use GENI experiment network deployment because it serves as a proof of concept experiment platform for wireless aspects. In the second semester all members are expected to contribute to the testing of our algorithm as well as the configuration of the ORBIT network topology.

2.7 Other Resource Requirements

ORBIT testbed 9 is required for testing as all of the setup scripts and code are written with a specific topology in mind, testbed 9 also contains enough nodes to simulate the constructed network.

2.8 Financial Requirements

There are no financial requirements for the project because we will be using free platforms such as GENI and ORBIT. Experiments will be conducted with those platforms and coding will be done mostly in C and Linux.

3 Design

3.1 Related Products and Literature

1. Self-healing methods and base station for obtaining the basic configuration parameters in self-configuration process of base station

   A kind of base station self-configuring process is obtained by the self-healing method and the base station of basic configuration parameters. The present invention relates to the wireless communication field, particularly self-healing method and the base station that a kind of base station self-configuring process is obtained the basic configuration parameter in LTE (Long Term Evolution, Long Term Evolution) system's self-organizing network.

2. Realizing method and system of self healing of base station cells in long-term evolution system.

   The implementation method of self healing of the base station cell and system in the long evolving system. The present invention relates to base station cell foundation technology in mobile communication Long Term Evolution (LTE, the Long Term Evolution) system, relating in particular to the implementation method and the system of self healing of base station cells in the LTE system.

   This implementation method comprises:

   > The base station detects baseband board and occurs judging the vacant base-band resource that self whether has the minimum bandwidth requirement that satisfies sub-district on the unusual baseband board when unusual, if exist, then initiates the sub-district and sets up process and rebuild sub-district on the unusual baseband board; Otherwise the redundancy bandwidth of obtaining the sub-district from normal baseband board is as available base-band resource, and when available base-band resource satisfies the minimum bandwidth requirement of sub-district on the unusual baseband board, rebuilds the sub-district on the unusual baseband board.

3. Self-healing network which wireless networks will fix their own broken communication links.

   The challenge of wireless communication is that wireless communications travel through unpredictable environments, which cost degraded connections or broken communication links between a server, a broader network, and a client. The present invention relates to automated network analysis through the provider, connection stability from the internet, and bandwidth evaluation as the essential features of the self-healing network process. Usually, networks establish two or

more routes between the server and the client (receiver). Through evaluation, networks detect route failures, trigger diagnosis and compensation of the network rerouting.

3.2 Design Thinking

The majority of the broad design decisions for this project have mostly been laid out before the beginning of this project. The general idea as to how to complete this project falls on the basics of self healing networks which includes outage detection, diagnosis, and compensation.

The design choices left are all about how we choose to implement the three fundamental steps of self healing. For outage detection a series of check in exchanges between client and the internet can be arranged so that if a check in fails there can be further investigation. This leads into diagnosis, once a failure has been reported the algorithm needs to begin testing for the source of the outage (client to server, server to internet etc). From there a survey of available nodes needs to be made. For compensation the central controller will first search out neighboring servers that have enough excess bandwidth to give out to the client. If no servers have enough excess bandwidth to compensate for an extra client the server with the highest excess capacity will connect to the client and the bandwidth of the original client and the new client will be throttled so that both clients may still connect to the broader network.

3.3 Proposed Design

The current design of the self healing project is centered around using the wired ethernet interfaces of the ORBIT testbed. Specifically our design is built around using sandbox 9 in the ORBIT testbed. This specific sandbox was chosen as it has many nodes to accommodate a full small scale network of 7 nodes. We are using shell scripts that utilize various OMF commands to image and route the nodes correctly. The server, client, and internet nodes will all communicate with each other using network sockets. Any computational tasks will be taken care of using C files that are unique to each node and can be completed in real time. The proposed design meets many of our design requirements as it is a free to use resource and can be heavily modified to meet the needs of any test that is needed.

3.4 Technology Considerations

The strengths of the free platforms is that the connection is very fast and it works efficiently. In terms of reserving nodes in ORBIT testbed, the steps are rather simple and easily understandable. However, we are currently facing issues with wireless interfaces of the ORBIT nodes. From what we can gather the wireless interfaces were present but due to the lack of proper wireless drivers we were not able to use the wireless components. The amount of work that would be required to create our own drivers would be much more than what we have time for so the decision was made to use the existing ethernet

interfaces. The issue stems from the mismatching versions of OMF and the Linux version that is installed upon imaging the nodes. There isn't much that can be done given the timeframe of the class so we would have to use wired connections for simulation purposes.

3.5 Design Analysis

Our proposed design is a very strong proof of concept for our overall plans of a general self-healing algorithm. It is very feasible with our current knowledge of networking and ORBIT. We already have demonstrated our knowledge of routing protocols for our proof of concept, also we have created several scripts to run on our nodes that can detect and compensate for a network outage. The next step for our design is to add more metrics and protocols for reconnecting downed nodes in such a way that the network can be maximized. Once we get to a point that we are happy with our test network we can begin the process of scaling up our algorithm to better simulate a real world scenario.

3.6 Development Process

We are following the development of the waterfall, which starts with the requirements (verify that all network stations can be self-healing when one is down, then another station has to support it), then gathers the software we need to implement such as Linux and C (mostly on Linux). We also have to design how fast the station will react when noticing a problem and test if the failure station will automatically transfer clients to other working stations, which prevents clients internet's connection lost.

3.7 Design Plan

This project will need to work with an ORBIT sandbox and across other open-access networks. We need to create ORBIT topologies design, test, and improve the network's automated repair. There will be at least three stations and three clients for this project. In the case of the user's internet connection, there are UDP packets connecting the internet node to the clients. If the message fails or there is an outage meaning that a base station is not functioning, then the network will attempt to heal itself by rerouting the traffic from the failing base station, to the working base stations and finally to the broader network. Once this is done, traffic between the Internet and Client will continue.

3.8 Evolution from CPR E 491

There are a few key differences between our project in the first semester of senior design to the current project. The first difference is the addition of two network nodes: one server and one client. These nodes were added to increase the complexity of our network to be closer to a real world environment, and so that we could model the failure case of two servers failing at the same time. The other main change between the two semesters was the transfer from shell scripting language to C. As the complexity of our algorithm increased, it was no longer possible to model the network using shell scripts. We then transitioned from using ICMP packets to UDP packets for intra-network communication. This allowed us to exchange network state information between nodes within our network.

Finally, our second semester design now allows us to model direct server failure rather than just routing failures.

3.9 Security concern

For our project there are not many security threats as we simply reroute traffic in a network. In our system there is no consideration for critical connections that cannot afford to be throttled if new clients are connected to the same server. To fix this situation there would have to be special steps put in place on the server side to stop the throttling of any important connections in the event that new clients attempt to connect to a shared server.

4 Testing

4.1 Unit Testing

To test the functionality of our project there are two main scenarios that we use. The first scenario represents a failure of the connection between a server and the internet. In this case we will remove the specified route to create an error in our system using a linux command from the central controller. The response of the system is to route traffic from server 2 to server 3 and to the internet. This represents a downstream connection failure of the server to the broader network.

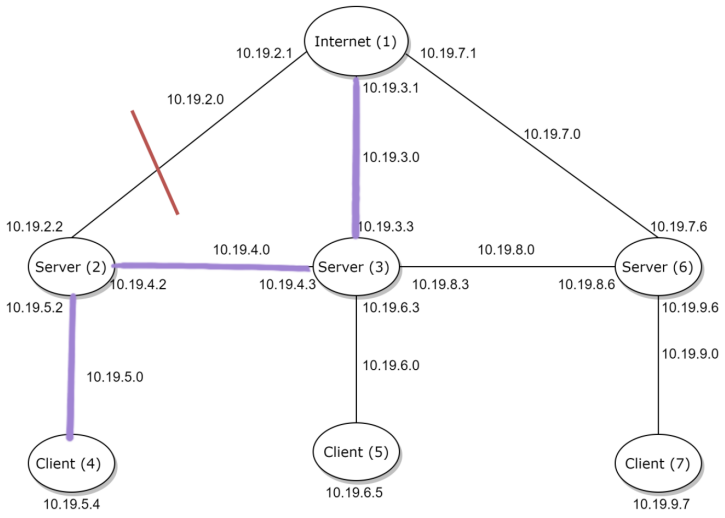

Figure 5

A second scenario (figure 6) that we use is to remove the functionality of an entire node or nodes. This is done by either removing all connections to the node or simply by turning the node off. Like in the previous test the central controller will reroute any traffic through available nodes. In the below picture it can be seen that server nodes 2 and 6 are non functioning and traffic from all client nodes are routed through server 3. The scenario that this test is representing is a complete failure of a server or base station. This can be a result of power loss, antenna damage etc.

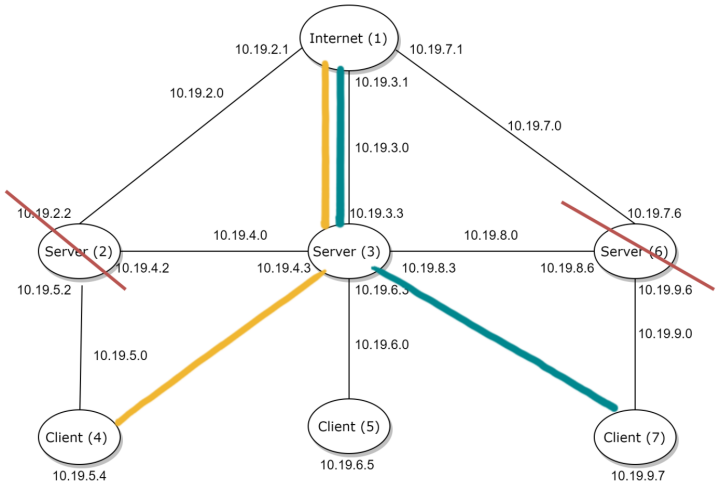

Figure 6

4.2 Interface Testing

There are many interfaces that need to be utilized and tested throughout our project. Some of these interfaces include the client to server, server to internet, and central controller to all nodes. Not only do we need to ensure that the nodes are communicating correctly with one another, we also need to ensure that the communication between nodes is happening through the correct interfaces. This is because we have created a network through one interface, however there still exists other interfaces in which nodes are still connected. One tool that has proven extremely useful is the traceroute command. This command can be used from any node to ping another node and the path will be displayed. To test if our connections are behaving correctly we analyze the traceroute information and confirm that the correct route is being used.

4.3 Acceptance Testing

To prove the capability and success of this project, it is necessary to show progress toward the main pillars of self healing networks which are outage detection, outage diagnosis, and outage compensation. Proving the competency of the project, our system must be able to handle an unexpected loss in signal by first detecting the severity and location of the loss, diagnosing the cause of the loss, and drawing on the resources of surrounding cells in a quick and efficient manner.

For our project this means that our algorithm is able to handle any amount of unexpected loss whether that is a node failure or a routing failure. Our algorithm should be able to handle multiple points of failure and route clients successfully. This means that the bandwidth is not exceeded for any connection and that the clients receive the maximum amount of bandwidth available by the network.

4.4 Results

During the course of this project we were able to achieve several successes on the ORBIT platform. We were able to create a seven node network that is able to achieve the three steps of the self healing process. Our network that we created was able to detect a network outage (node failure, connection failure,etc.) and diagnose the source of that outage. The algorithm would then go on to search out a solution for reconnecting the client to the network. Several variables were considered such as available bandwidth and connected clients. After the completion of this project we were able to realize all the major components of self healing and implement them in a small scale environment.

5 Implementation

Our implementation of this project has already been partially described in the rest of this design document. There are two main components of our project. The network topology and the self-healing algorithm.

Network Topology:

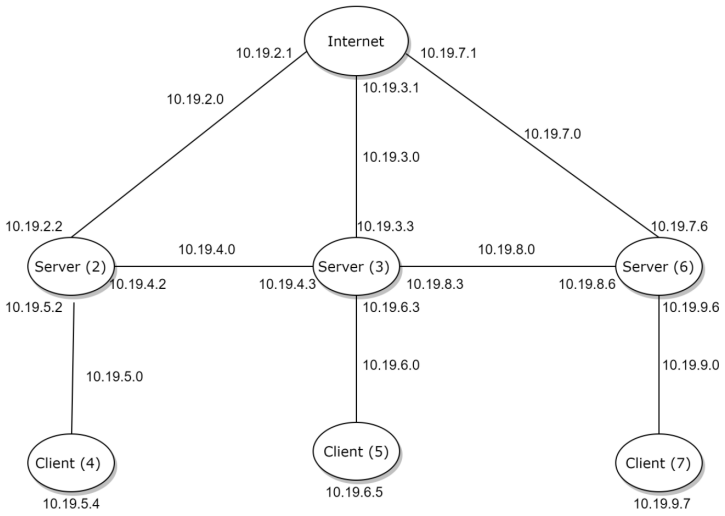

Figure 7

Our designed network consists of 3 clients, 3 servers, and one central controller. The clients represent everyday network users, the servers represent network base stations, and the central controller represents a monitoring service that can assess network state. We assume that the central controller is somewhere away from the local nodes, and therefore can be conflated with the broader network. While this topology doesn't exactly match the topology of a real-world wireless network, we believe that it is a model that can be used to easily test our self-healing algorithms. This topology represents the following real-world network:

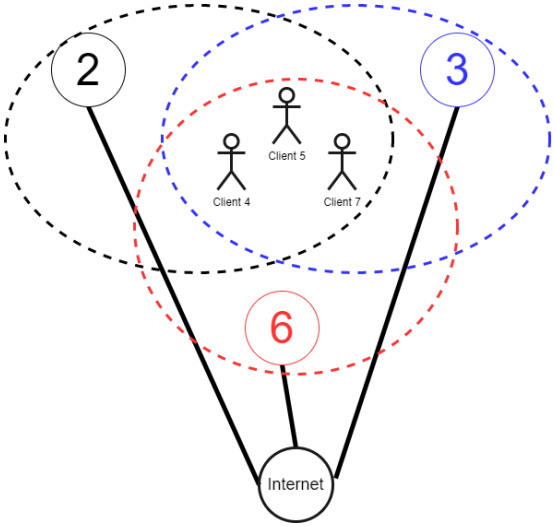

Figure 8

In this network, there are users with three nearby base stations. Our algorithm attempts to connect the users back to the internet in case of failure in an efficient manner. Although our topology is wired, we believe that it represents the wireless model above.

Self-Healing:

To simulate network traffic we use UDP sockets to send traffic from the clients to the broader network and vice-versa. These UDP sockets are configured on each port of the server so we know which client is connected to the broader network. Our algorithm has three parts: Detection, Diagnosis, and Compensation. The UDP connection is constantly being monitored by the central controller so the network state can be assessed. Once a client or server misses a check in, we begin the self-healing process. First, the servers are polled and the current bandwidth capabilities of the available servers is calculated. If there is available bandwidth, the clients are redistributed to the nearest network node, if not the clients are assigned to the server with the largest available bandwidth and the clients are

throttled. If there are no available servers, then we have a total network failure and there is no self-healing possible.

6 Closing Material

6.1 Conclusion

We built a simple self-healing algorithm that is run on the ORBIT testbed. This algorithm can test several different connections and reroute traffic depending on connection health, node health, and available bandwidth. This is a very close step behind moving to a large scale system and being able to test our project in a real world setting.

- The internet node represents the broader network.
- the server nodes represent base stations.
- the client nodes represent users.

If an outage is detected the network will attempt to heal itself by rerouting the traffic from the failing base station to a working base station and then to the broader network. Once it goes through all processes traffic between the internet and Client will be back to normal in which users can connect to the internet again.

6.2 References

Citations:

郑, 纪伟. (2011, December 28). Self-healing method for acquiring basic configuration parameters in the self-configuration process of base station and base station. Retrieved October 25, 2020, from https://patents.google.com/patent/CN102300207A/en

张, 现周, 易, 鸿锋, & 张, 文英. (2012, July 25). Realizing method and system of self healing of base station cell in long-term evolution system. Retrieved October 25, 2020, from https://patents.google.com/patent/CN101772059B/en

Robert Poor, Cliff Bowman, Charlotte Burgess Auburn, Ember Corporation. Self-healing Network. Retrieved April 24, 2021, from https://queue.acm.org/detail.cfm?id=864027

Appendix

Appendix I - Operational manual

There are many steps to getting the testbed configured correctly. In-order to proceed with the following guide, you must first procure an ORBIT account . This process normally takes a few weeks to complete.

Once you have this account, familiarize yourself with the platform and the basics of how to reserve testbeds, this information can be found on the ORBIT Wiki:

https://www.orbit-lab.org/wiki/Documentation/bAccountManagement/DSSHConf

Follow these steps until you have your ssh keys properly generated. Then, reserve sandbox 9 for the amount of time you would like to run your experiment. Ssh into: username@console.sb9.orbit-lab.org

There are now a series of configuration scripts that must be run in order to configure the sandbox to the correct network topology. These scripts will take approximately one hour to complete and once they have been run, the network will be ready to experiment on. These scripts will be briefly summarized and more information on the code behind them will be provided in Appendix IV. To fully bring up all nodes easily, run the all-in-one configuration script : ./Bringup.sh 1 1 1 1

Setting up the baselines:

1. First you must set up all network nodes with the correct baseline images. This is done through a series of 3 commands:
    a. omf tell -a offh -t node1-x (turns off x node)
    b. omf load -i baseline.ndz -t node1-x (loads baseline image onto node x)
    c. omf tell -a on -t node1-x (turns on x node)
2. This process must be repeated for each network node in the experiment.

Installing the tools:

1. The self-healing algorithm requires two tools to be downloaded onto each node: net-tools and traceroute.
2. To download these tools:
    a. ssh root@node1-x (ssh into network node x)
    b. sudo apt install net-tools
    c. sudo apt install traceroute
    d. exit
3. Perform this operation for each of the 7 network nodes

Configure Routes:

1. This is the most important step of the bring up process and it involves setting up the desired network topology for the experiment. Each node must be configured so that its connections resemble the desired topology. Because each node is technically connected to a common network switch, we must artificially create our network using routing tables and a secondary ethernet interface “enp94s0f0”.
2. There are many steps to configuring routes, but here are some of main parts:
    a. ifconfig enp94s0f0 x.x.x.x netmask 255.255.255.0 up (set up network interface)
    b. sudo ip addr add 10.19.4.3/24 dev enp94s0f0 (add additional ip addresses to interface)
    c. sudo route add -net x.x.x.x netmask 255.255.255.0 gw x.x.x.x enp94s0f0 (add routing for wanted network topology)
    d. echo 1 > ip_forward (allow ip forwarding on
3. For an example of a node being fully configured for our topology please refer to Appendix IV.1
4. Please note: As we have been configuring all of our routing through enp94s0f0, we must now specify that that interface should be used when performing tests of network connectivity. For example: ping -I enp94s0f0 x.x.x.x
5. The network topology should now be configured.

Load Experiment Executables:

1. scp UDPServer root@node1-1:/root/UDPServer
2. scp UDPClient4 root@node1-4:/root/UDPClient4
3. Repeat step 2 for all clients.

Run Experiment:

1. ssh root@node1-1
2. ./UDPServer
3. exit
4. ssh root@node1-4
5. ./UDPClient
6. exit
7. Perform 4-6 for each client
8. To see traffic and algorithm output, ssh into central controller(node1-1)
9. To kick off self-healing either manually disconnect a server (omf tell -t node1-x -a offh)
10. Or, delete an essential route in any node’s routing table

Appendix II - initial version of the design

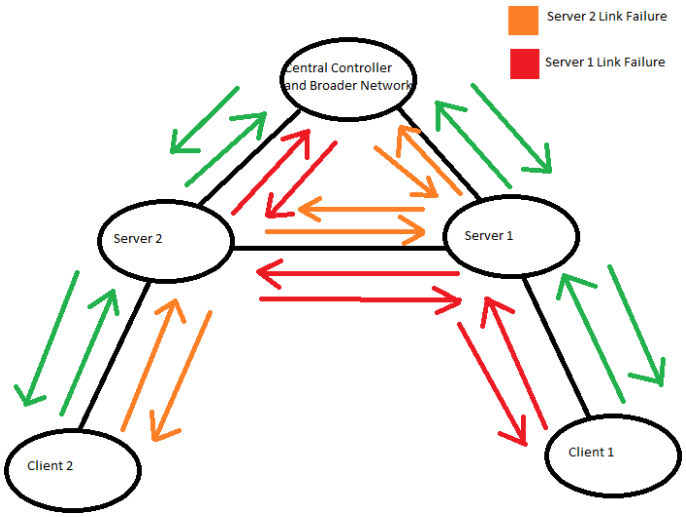

Figure 9

An early version of our self healing algorithm was built using the Geni platform. This platform was a method of simulating networks using virtual machines that could have their interfaces configured through simple commands in the network generation step. Using Geni was a good way to introduce ourselves with self healing, however being that we were using virtual machines there was a limited amount of simulation possibilities.

The network that we had created on Geni was a simple two server, two client network that would serve as a trivial test for simple rerouting scripts. This simple network design would later be created on the physical ORBIT testbed and would be built upon.

The self healing algorithm for this version of our design had two significant flaws when compared to our current version. The first flaw was that this version used a shell script to perform the self-healing algorithm, rather than the current version that uses executable C programs. The second flaw was that this version was only able to represent downstream routing failures. Our current design is capable of simulating direct server failures as well as

Appendix III - Other considerations

We considered using WITEST (alternative platform which can work with GENI) to work with a self-healing process instead of ORBIT because of its outdated tutorials. Unfortunately, we couldn't sign up to use the WITEST platform because of the Covid-19, which we would need to register an account in person with one of the WITEST developers in New York. We decided not to go and continue using ORBIT to accomplish all variables and requirements needed to complete the project.

Appendix IV

```
echo "---------------CONFIGURING NODE 3---------------"
ssh -o "StrictHostKeyChecking no" -tt root@node1-3 <<- EOT
ifconfig enp94s0f0 10.19.3.3 netmask 255.255.255.0 up;
sudo ip addr add 10.19.4.3/24 dev enp94s0f0;
sudo ip addr add 10.19.6.3/24 dev enp94s0f0;
sudo ip addr add 10.19.8.3/24 dev enp94s0f0;
sudo route del -net 10.19.3.0 netmask 255.255.255.0 gw 0.0.0.0 enp94s0f0;
sudo route del -net 10.19.4.0 netmask 255.255.255.0 gw 0.0.0.0 enp94s0f0;
sudo route del -net 10.19.6.0 netmask 255.255.255.0 gw 0.0.0.0 enp94s0f0;
sudo route del -net 10.19.8.0 netmask 255.255.255.0 gw 0.0.0.0 enp94s0f0;
sudo route add -net 10.19.3.1 netmask 255.255.255.255 gw 10.19.3.3 enp94s0f0;
sudo route add -net 10.19.4.2 netmask 255.255.255.255 gw 10.19.4.3 enp94s0f0;
sudo route add -net 10.19.6.5 netmask 255.255.255.255 gw 10.19.6.3 enp94s0f0;
sudo route add -net 10.19.8.6 netmask 255.255.255.255 gw 10.19.8.3 enp94s0f0;
sudo route add -net 10.19.2.0 netmask 255.255.255.0 gw 10.19.4.3 enp94s0f0;
sudo route add -net 10.19.5.0 netmask 255.255.255.0 gw 10.19.4.2 enp94s0f0;
sudo route add -net 10.19.9.0 netmask 255.255.255.0 gw 10.19.8.6 enp94s0f0;
cd /proc/sys/net/ipv4;
echo 1 > ip_forward;
exit;
EOT
```

Figure 10