Self-healing Networks Using the ORBIT Testbed

sdmay21-36

Team Members

Advisor

Mohamed Y Selim: myoussef@iastate.edu

Student Members - sdmay21-36@iastate.edu

Matthew Johnson

Adam Tiedeman

Parnumart Hanthiradej (Nick)

Yuin-Choon Ng (Austin)

Amber Chin

Self Healing - Problem Statement

Purpose

- Manage network outages autonomously
- Human experts cannot sift through the vast amount of network traffic in a reasonable amount of time

Why do we need self healing

- As the number of node in a network increases the probability of network outages increases
- As node density increases so will the complexity of the system

Components of self healing

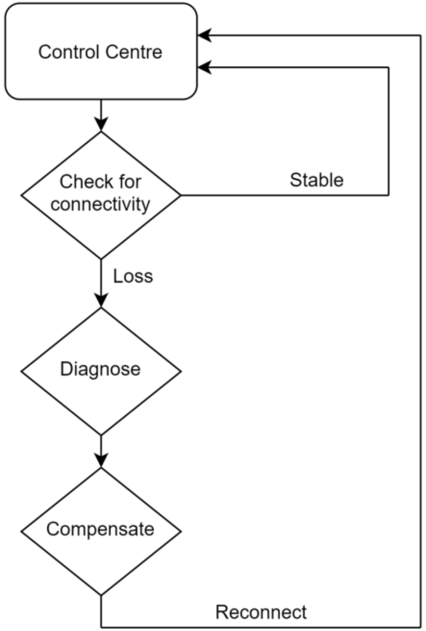
- Outage Detection
- Outage Diagnosis
- Network Compensation

Conceptual Sketch

Failure detection
- checks for network connectivity
- initiates failure diagnosis

Failure diagnosis
- Reroute traffic or attempt to maintain client connection with available servers
- Execute failure compensation

Failure compensation
- Routing error: central controller reroute based on diagnosis
- Total failure: assess clients' connection loss, command servers to reach out to client
- After failure is compensated, resume monitoring



Engineering Requirements

Functional

- Detect the loss of communication with a base station and mitigate the effects of the outage so users of the network do not experience an extended outage
- The central controller should be able to assess the network state with limited network congestion
- The self-healing process should make sure network nodes are not overloaded beyond their bandwidth

Non-functional

- Economical
  - We are using the free platform ORBIT
- Environmental
  - This project should be functional across open-access networks.

Engineering Constraints

- Our project must be able to run on an ORBIT testbed consisting of at least 7 network nodes
- Our central controller must be able to quickly assess network state without flooding the network
- Our wired connections must correspond to our topology to simulate the wireless network
- The setup of the orbit sandbox limits us.

Engineering Standards

Networking 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 Standard

ORBIT Testbed

- Wireless Network Protocol
- Radio grid testbed
- NSF funded project
- Sandbox networks
- Reserve time slots on ORBIT

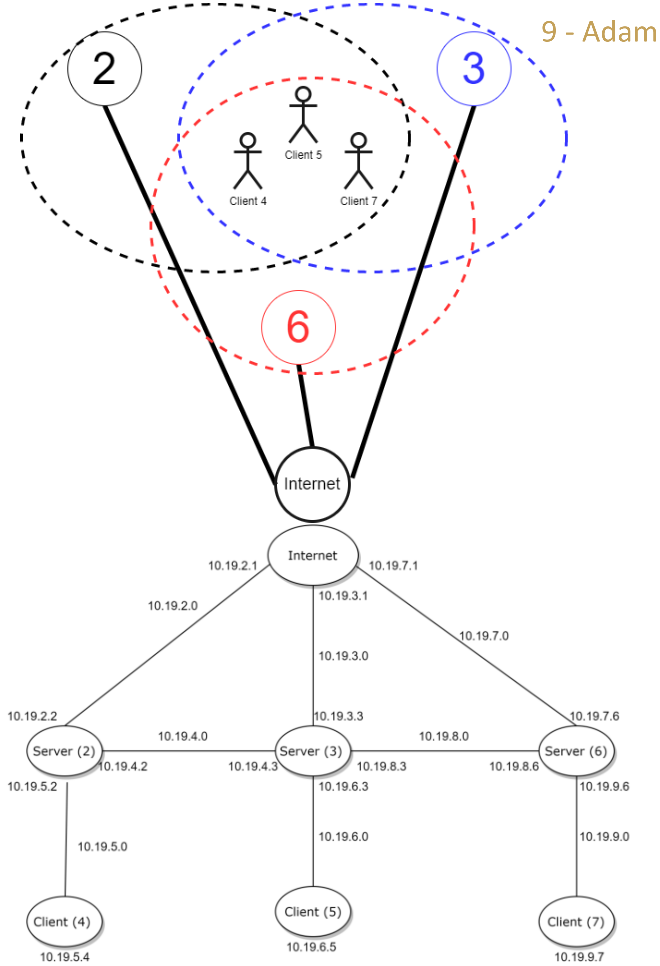
https://www.eurekalert.org/multimedia/pub/119905.php

Network Topology

- Our network simulates a topology consisting of three clients and three servers within wireless communication range of one-another

- This is a basic representation of a physical network, but the concepts demonstrated can be scaled up to a larger topology with many servers and many clients

- We use linux routing tables to model physical connections



Configuration Scripts

Load baseline

- Turns nodes off, loads image onto node, turns node on

Install tools

- Ssh into nodes, installs net-tools and traceroute

Configure routes

- Artificially create network on secondary ethernet interface
- Add ip address and routes/ ip forwarding

Prepare executables

- Compile and transfer all files to respective nodes

Description of algorithm

- UDP sockets are installed on each individual node
- Sockets are configured with executables on each node to accomplish self-healing
- Allows us to transfer network state “on the fly”
- Clients generate traffic to central controller as a “check in”
- If “check in” is missed (timed out) the central controller can give specific commands on how to reroute the network
- To find a suitable route central controller searches for server that can accept entire client bandwidth, if no server can accept the server with highest available bandwidth is chosen and bandwidth is throttled

UDP Client

```
system("echo -n \"01.5\" | nc -u 10.19.2.1 8082 -s 10.19.5.4 | echo");
printf("Hello message sent.\n");
```

Roles:

- Each client runs a program that generates traffic to send to the central controller using netcat
- This traffic tells the central controller the client is in a working state and its associated server is correctly routing traffic

Upon Failure:

- The client will look for information received from the central controller to indicate how its routing table should be updated upon failure
- The client will then reroute based on the central controller's instructions

UDP Server

```
servaddr2.sin_family    = AF_INET; // IPv4
servaddr2.sin_addr.s_addr = inet_addr("10.19.2.1");
servaddr2.sin_port = htons(8082);

servaddr3.sin_family    = AF_INET; // IPv4
servaddr3.sin_addr.s_addr = inet_addr("10.19.3.1");
servaddr3.sin_port = htons(8083);

servaddr6.sin_family    = AF_INET; // IPv4
servaddr6.sin_addr.s_addr = inet_addr("10.19.7.1");
servaddr6.sin_port = htons(8086);
```

Roles:

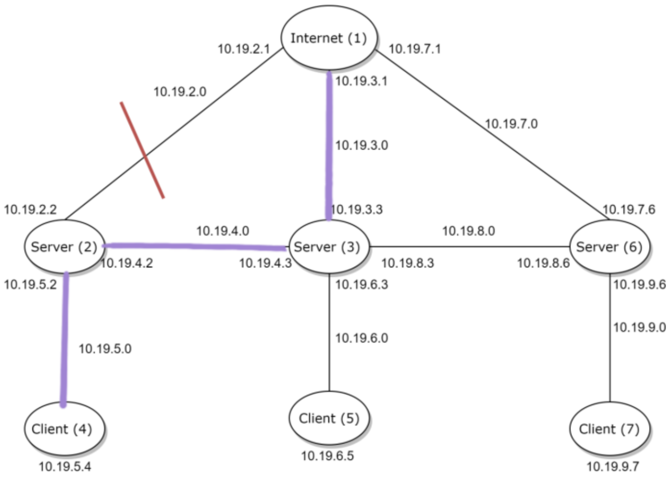
- Set up server sockets on each of its three outgoing interfaces
- Monitor incoming signals from clients and update table with which clients are connected to which servers, and their respective bandwidth requirements

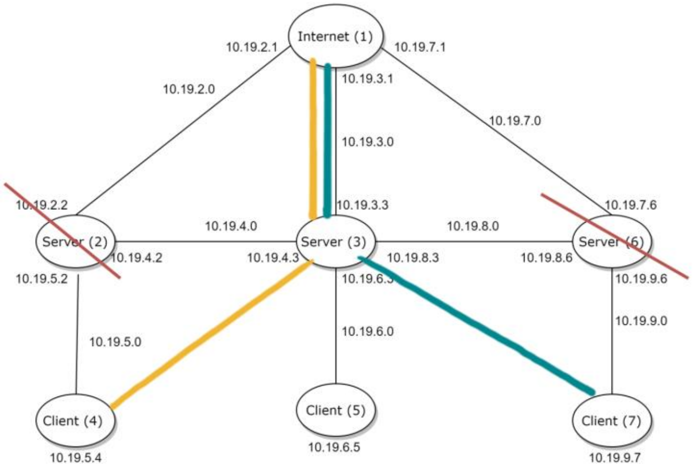
On the Event of Failure:

- First assess how the missing server's traffic can be routed to adjacent nodes
  - First try to assign broken server's clients to the nearest adjacent server(if it has excess bandwidth to serve the client)
  - If this server did not have excess bandwidth, check the other server's available bandwidth
- If neither server has excess bandwidth, we throttle connections to allow all clients to maintain a lower-speed connection
  - The server with the most excess bandwidth is chosen to service the clients that now need connection
  - Clients are throttled based on a ratio of their current connection rather than a flat amount
- Once the path to recovery has been calculated, the updated routing information is sent to the client over UDP and the client is expected to update its routing table

Failure Cases

Downstream routing failure Direct node failure





Technical Challenges

- Many ORBIT documentations are either outdated or nonexistent
- Getting our artificial network to behave properly was hard
- Outdated ORBIT wireless card drivers- say we could have written our own but too much time
- Unreliable nodes and reservations

Future of the project

- Get project working on outdoor test grid
- Convert to wireless network using updated drivers
- Increase nodes and node density

Conclusion

- We built a simple self-healing algorithm that is run on the ORBIT testbed.
    - 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.

Question?