Nathan Pocta, Chandler Coffman, Jacob Goff, Ryan Barker, and Tyler Johnston
ECE 4400

Final Project Report

Motivation

In online gaming, if a host were to have an unsatisfactory connection or left an online match altogether, this would cause an early termination of the match. This termination is a problem for many clients enjoying the match as it progressed. To address the problem of early match termination, we propose host migration as a solution which would allow the progression of the game to continue. Continued progression of the game would grant a transfer of ownership of the game from one host to another in separate threads, as many times as necessary. So long as there are enough clients and viable hosts to accept ownership of the game, the match will always progress until terminating normally.

Project Description and Background

The purpose of this project is to create a fault-tolerant network that can support multiple clients exchanging information with a host. The goal is to create a network that is robust enough to survive the failure of a link and the failure of a host. Link failures will be handled using fast-failover groups in the OpenFlow protocol. The network will be designed with multiple paths between switches for redundancy. When one path fails, the OpenFlow groups for the affected switches will identify and switch to the correct new paths in the network that will ensure continuing functionality. Host failure will not be handled with a controller. When a host fails, or a user quits the application, the clients must identify that the host is no longer active and elect a new host from the pool of clients based on the lowest active IP address. After the new host is selected, the clients send pieces of their game state to the host to allow it to correctly manage and run the application.

Software Architecture

Plan

All vital game information will be stored in a gameMaster structure. This structure will contain things such as the positions of all player's ships, the number of ships each player has, and IP addresses of all players connected to the game. The only process in the entire network that will have access to this data structure is the host of the game. In order to implement host migration after a new game host has been selected, the new host process will request information from all players to form its own gameMaster structure. The host of the game is responsible for processing all messages to and from the players of the game. These messages could include things such as firing orders and telling the players if their firing request was successful. Each player process will have its own unique structure called gameInfo. The gameInfo structure contains all current ship information, an array for storing the results of firing on enemy ships, and an array for storing the host and other player's IP addresses. In the event of a player process being chosen as the host, the player process will fork a new child process that will serve as the host of the game. Communication between all processes in the network will be accomplished using the sockets API in c.

Implementation

Each game begins with an election process. The four IP addresses are hard-coded into the program. Each separate computer starts the program and begins to search for the other players. To do this, each separate computer runs a server for 60 seconds in a forked process. Then, each computer takes turns attempting to connect to the other computers' servers. It marks which ones it can connect to and which ones it cannot connect to in a boolean array. It then finds the lowest-indexed, active system, and designates it as the host. The host forks a process, runs the host program, and then runs a client program. The clients simply run the client program.

The host program starts by allocating a new structure, called a gameMaster, which contains members to keep track of each player's game state. Each member is then initialized, which includes opening a socket from the host process to each player. Once the sockets are opened, the host asks each player for its ship positions, which allows the host to compute the current health of each player and the current state of the game. Having each player send their ship positions to each new host is integral in making host migration function correctly. From here, the host enters a loop where it sequentially allows the players to take turns by transmitting them a 'T'. After a player sends a fire command, the host analyzes the coordinates it receives and transmits hit or miss back to the player. If a player loses, the host informs them and closes its socket with them. This process continues until only one player is left alive, at which point the host informs them they are the winner, cleans up its memory, and ends its process. If a player quits at any point, the host disconnects from them without transmitting.

The client program starts with each client placing their ships on the Battleship board. After this process is completed, they will attempt to connect to the host. Once each client is connected to the host, they wait for the host to request their ship positions. One challenge here was to send all positions in one single send command. To do this, one large, contiguous block of data had to be allocated and made into a two-dimensional array. After the clients each send their ship positions to the host, the game begins. The client will receive a token indicating it is its turn. The player then enters his or her command into the command prompt. The client sends this command to the host, where it is processed. The result of this move is sent back to the client after it has been processed. The host keeps track of the results on its own game board.

If, for any reason, a host machine ceases the running of the host program, a new host should be selected. This process entails running much of the same code as the original election process. This re-election process is almost exactly like the original election process. The difference is that the clients must send their established ship position arrays and their designated player numbers to the new host instead of re-entering them for the new host.

Network Architecture

The players will be connected by a series of switches. A centralized controller will be set up to handle the switches and the fault tolerant network. It will make use of OpenFlow groups and be used to automatically change link states in the network. The controller can also simulate taking down hosts via a manual command. The topology is setup to have one switch per client plus two. We will make the use of the FAST-FAILOVER group due to its ability to detect and overcome port failures. A sample topology of the network can be found in Figure 3 of the Appendix.

Designed Experiments/Demonstration

The functions that operate each part of the software were tested as modularly as possible. This allowed us to more easily debug the final program once all of the pieces had been put together. After successfully integrating the basic game, host operation, player operation, and host migration functions, the best way to experiment with the program was to simply play the game until something did not work correctly.

To demonstrate our project we will have four of our group members ssh into the CES apollo machines and run the precompiled game software. Each of these four people will become players in the game, with one of them also being selected as the game host. The machine that is selected as the host will print out a message to the console. This will allow us to demonstrate the host migration portion of the software, without having to guess as which person is the host. We will then play the game until one player has lost all of their health to show that the game can successfully determine a winner and a loser.

Outcomes

After completing as much testing as we had time for, the most important functions of the game all work correctly. The game can successfully be played with 2-4 people, as well as complete up to two host migrations. However, host migration can only take place after the initial setup of the game has been completed. More specifically, all players must have placed their ships onto the board before migration will be successful. The most difficult piece of the software to successfully implement was the synchronization of game logic after a host migration. It took almost a full night of testing just to get that part working correctly. This was due to the fact that the new selected game host had difficulty in keeping the player numbers assigned to the same person after a migration.

The GENI topology also presented a very large complication to the overall project completion. Currently, hosts 1 and 2 in the network cannot successfully connect to host3 and host4. The GENI topology used for the project was an extension of the second project in the course demonstrating the design of a fault tolerant network. However, in this topology, there would be a total of 12 switches: 4 switches connecting directly to the hosts, and 8 redundant switches between all the main switches, as seen in Figure 3 of the Appendix. After configuring the switches, hosts, and controller, groups and flows were inserted for the main switches and flows were inserted at the redundant switches using the Floodlight API as demonstrated in the second project. Like the second project, one of the FAST-FAILOVER groups for the topology was functioning (namely the group for the switches connecting host3 to host). After downing one of the redundant links between host3 and host4 (s34a in Figure 3) the FAST-FAILOVER group in the other redundant link (s34b) would respond to this event and correctly route the packets between host3 and host4, demonstrating a fault tolerant network capable of two players. We were unable to figure out why this connection could not be established, which is why the demonstration will take place on the CES apollo machines.

Relation to Networking

This project uses SDN (Software-Defined Networking) in order to provide a robust network over which a collection of clients can communicate. It also uses a distributed protocol along with socket-based communication to recover a game state after a host failure. The result is a very robust gaming environment that can be built upon to provide higher levels of service in the gaming industry.

Responsibilities

Each team member's primary responsibility was:

- Chandler: Designed battleship game logic and initial local 2 player Battleship game.
- Tyler: Created and implemented the GENI Fault Tolerant Network.
- Ryan: Created game host and host sockets logic for Networked Battleship game.
- Nate: Created host election, host re-election, and main Networked Battleship scripts.
- Jacob: Created player sockets logic for Networked Battleship game.

Scheduling

The Gantt chart followed through the life cycle of this project is included in the appendix of this report as Figures 1 and 2. It includes all of the deadlines and responsibilities each group member held themselves to. The chart is included in both graphical and tabular format to assist in reading. Through the duration of the project, we managed to follow the Gantt chart sufficiently well and met each deadline we set for ourselves. A major component of our success were the weekly scheduled check-in meetings that allowed us to hold ourselves accountable for tasks and helped us reach resolutions on any tasks we were having problems with. Following our Gantt chart, we were able to finish the entire project two nights before the deadline. This just goes to show that communication and scheduling are the two most important parts of a group assignment, and we excelled at both of them.

Conclusion

In summary, for this project we designed a networked battleship game. The game contained two primary components: A fault tolerant network configured in GENI and networked battleship game code written in C. Each host had a copy of the Networked Battleship game installed. The GENI network was configured to handle potential downed links between hosts using FAST-FAILOVER groups with the use of the Floodlight API. However, when the SDN code for the network was compiled and after the network was configured, only one FAST-FAILOVER group was working, and only two hosts were able to successfully communicate with each other. The main difficulty in designing the Networked Battleship game was testing the C sockets programming itself. This was resolved by having everyone meet in an apartment and debugging the code across multiple machines on the same LAN. The project as a whole was contained many layers of complexity from configuring a fault tolerant network of twelve switches with fast failover groups to designing a multi-process sockets program that hosts a game over a LAN. This project was enjoyable to tackle, and we all learned a great deal from working on it.

Appendix

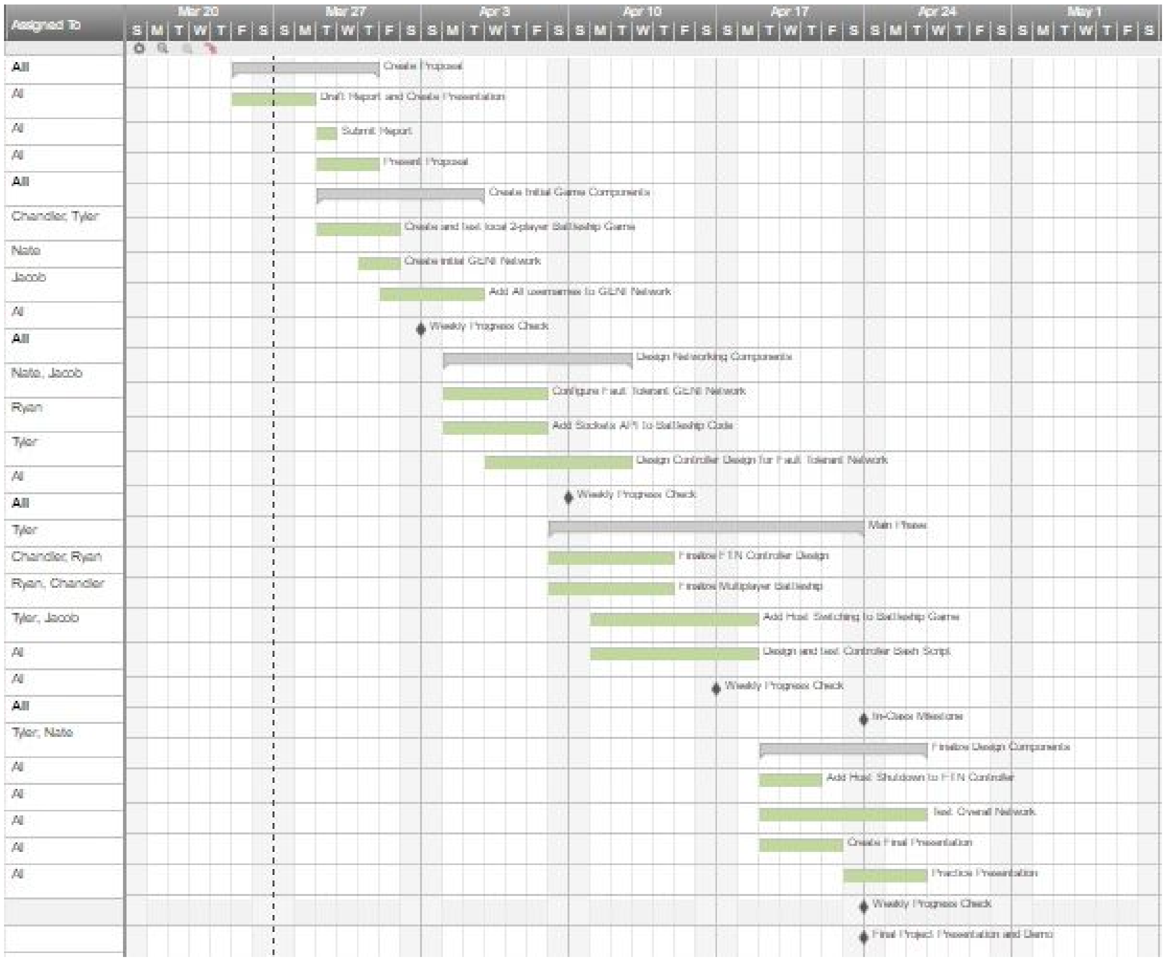
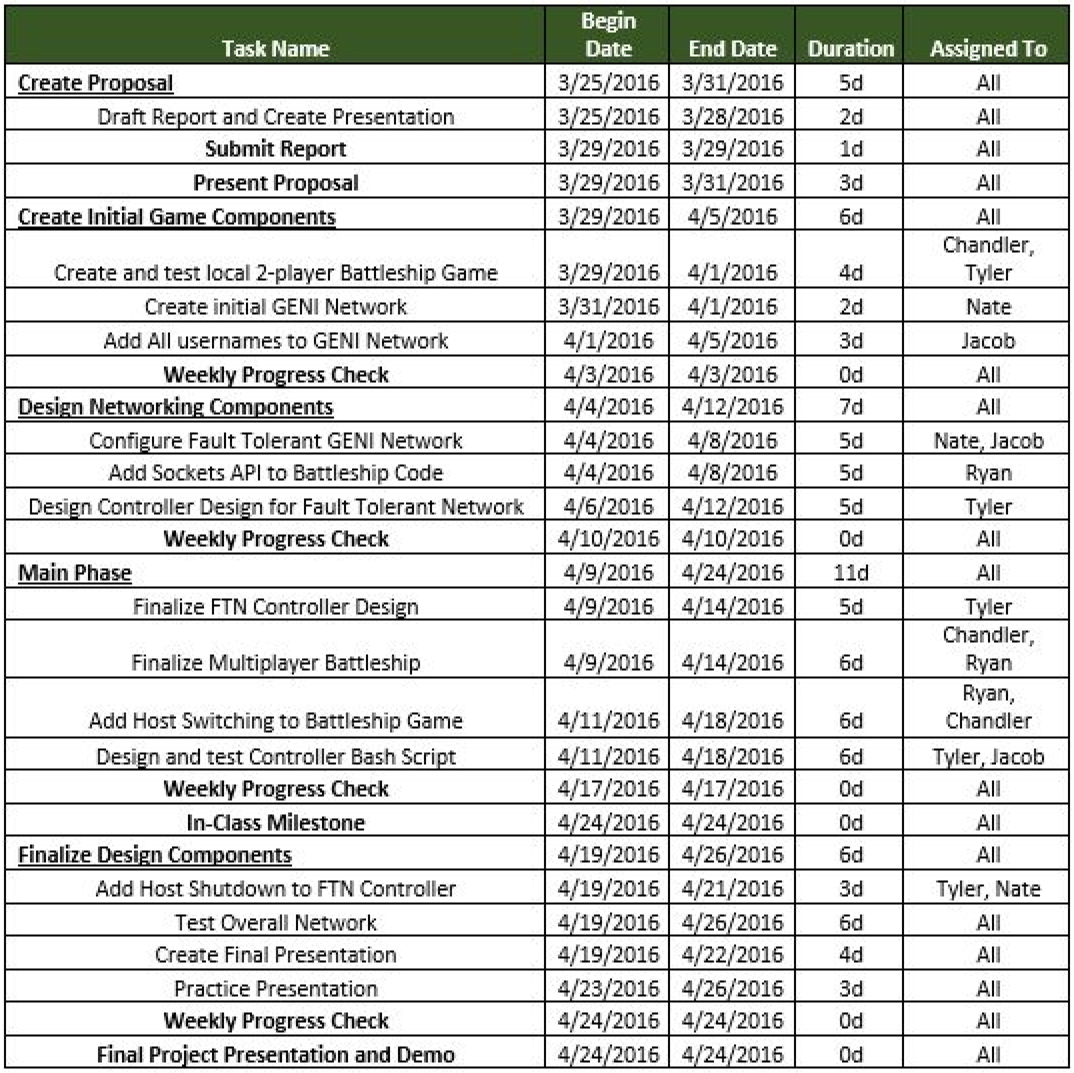


Figure 1. Graphical Gantt Chart.



Task Name	Begin Date	End Date	Duration	Assigned To
Create Proposal	3/25/2016	3/31/2016	5d	All
Draft Report and Create Presentation	3/25/2016	3/28/2016	2d	All
Submit Report	3/29/2016	3/29/2016	1d	All
Present Proposal	3/29/2016	3/31/2016	3d	All
Create Initial Game Components	3/29/2016	4/5/2016	6d	All
Create and test local 2-player Battleship Game	3/29/2016	4/1/2016	4d	Chandler, Tyler
Create initial GENI Network	3/31/2016	4/1/2016	2d	Nate
Add All usernames to GENI Network	4/1/2016	4/5/2016	3d	Jacob
Weekly Progress Check	4/3/2016	4/3/2016	0d	All
Design Networking Components	4/4/2016	4/12/2016	7d	All
Configure Fault Tolerant GENI Network	4/4/2016	4/8/2016	5d	Nate, Jacob
Add Sockets API to Battleship Code	4/4/2016	4/8/2016	5d	Ryan
Design Controller Design for Fault Tolerant Network	4/6/2016	4/12/2016	5d	Tyler
Weekly Progress Check	4/10/2016	4/10/2016	0d	All
Main Phase	4/9/2016	4/24/2016	11d	All
Finalize FTN Controller Design	4/9/2016	4/14/2016	5d	Tyler
Finalize Multiplayer Battleship	4/9/2016	4/14/2016	6d	Chandler, Ryan
Add Host Switching to Battleship Game	4/11/2016	4/18/2016	6d	Ryan, Chandler
Design and test Controller Bash Script	4/11/2016	4/18/2016	6d	Tyler, Jacob
Weekly Progress Check	4/17/2016	4/17/2016	0d	All
In-Class Milestone	4/24/2016	4/24/2016	0d	All
Finalize Design Components	4/19/2016	4/26/2016	6d	All
Add Host Shutdown to FTN Controller	4/19/2016	4/21/2016	3d	Tyler, Nate
Test Overall Network	4/19/2016	4/26/2016	6d	All
Create Final Presentation	4/19/2016	4/22/2016	4d	All
Practice Presentation	4/23/2016	4/26/2016	3d	All
Weekly Progress Check	4/24/2016	4/24/2016	0d	All
Final Project Presentation and Demo	4/24/2016	4/24/2016	0d	All

Figure 2. Gantt Chart Tabular View.

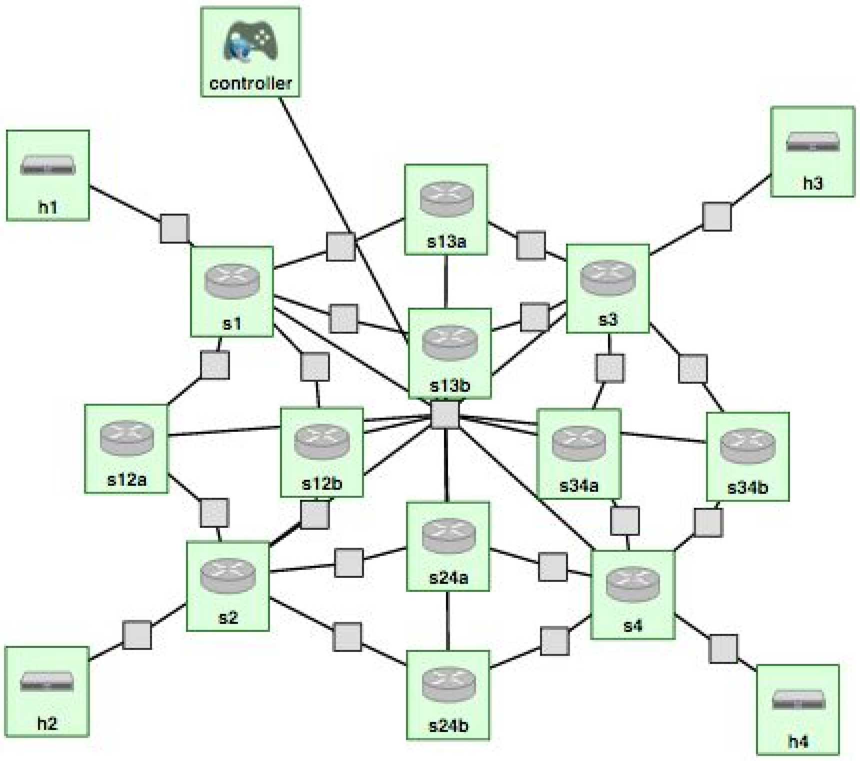


Figure 3. GENI Network Topology.