Design Decisions:

Client and Server Refactoring

Major refactoring occurred to our system. We switched our system so as to use a more streamlined approach. This includes changing from device, network, and partition creation on both the server to being done only on the server. This causes less chance of risk, and allows for more flexibility for the server. This includes beginning a complete overhaul of our CSS and HTML in order to use keyframes to give a much more visually appealing page. Although, this has not been implemented in this iteration, the CSS for this overhaul may be seen in “test.css”, and will be implemented completely next iteration.

Event Queue Integration and Event Logging

In order to better handle server client creation we created an **Event Queue.** The events are stored in the event queue on the client side. The event queue is then propagated to the server, where the events are executed and the changes are sent back to the client. This approach was chosen, as it allows us to seamlessly integrate **event logging**. Each event in the event queue is associated with a unique time stamp. The events in the event queue are then saved by their time stamp. Along with this, a snapshot of the event simulation in which this event occurred is saved, and can be used to restore the topology at that event. When a user would like to view the event logs, he simply retrieves all events from the server which have occurred up to a certain time.

Topology GUI Design Decisions

After discussing the look and feel of our Topology GUI with Dr. Fiech, we proceeded with the following design for our Topology GUI, which we had began during the previous iteration. Our goal is to design the most simple and intuitive GUI for this feature. To do this, we unify the entire process of manipulating the Topology to a single button. This allows every modification to our entire simulation topology to be one with the right mouse button. We have termed a 'transative closure' of communication between networks as a 'partition'.Therefore, we have implemented it as follows

- To view a device name, simply hover over the representation of that device in the topology
- To view a network name, hover your mouse over the representation of the network in the topology
- To move a device within the topology, right-click and drag the device to the desired location.
- To move a network within the topology, right-click and drag the network to the desired location
- To move a device from one network to another, simply drag that device out of the current network and into the new network, or place it outside of any network, in which case, that device will not belong to any network.
- To create a partition between two networks, or to join two partitions together, simply drag one network on top of another. A line, or edge, will appear between these two networks, indicating that these networks are within the same partition. A partition is all networks which can be reached by following any contiguous sequence of edges.
- To remove a partition line between two devices, drag one of the devices on top of the other, just as you did to create the partition line.

We chose this design for our Topology GUI, as it provides the following benefits:

1. Single button operation allows for intuitive and easy to use interface. Users do not have to

scramble to figure out what buttons do what.

2. By choosing this form of creating partitions, rather than the drag one line to another network approach, which we were considering previously, we are allows to use a single button for both partition creation and device/network movement, rather than two buttons which would be required for the other approach.
3. Network and Device movement: We allow devices and networks to be moved around the topology and rearranged. This allows the topology to theoretically handle a variable number of devices, by allowing the user to have the option to move devices and networks around to obtain a view of the topology which is the most clear to them. As well, this allows our topology to be a lot more user-friendly and personal, as the users may create a topology which they understand.
4. Avoid accidental deletion of partitions: We considered two approaches to partition-line deletion. The first, to click on a partition-line to delete it, and the second, to repeat the drag one network on top of the other process which we used to create partitions. We decided on the second choice for the following reasons. First, repeating the process of the creation of a partition-line seemed to us the most intuitive process, that way, this could be seen as a sort of "do-undo" process. Secondly, this way avoids accidental deletion which could be caused by the click a partition to delete way. This is because a user, who may not know how the topology functions, could just click randomly and end up deleting partitions. This avoids the system accidentally doing something which you don't want it to do, and gives more control to the user.

Token Management

Tokens are generated at the start of the simulation. Virtual devices are deployed in the browser. They a re required to provide a valid token before they can be allowed to be registered in the simulation environment (join and communicate within networks). The token entered is then be marked as "used" within the database. Any time someone tries to use a token, that the the token is unused is checked after the token is deemed valid.

Event flow for sequence diagram:

- The token is sent to the server
- The simulation manager routes the token to the token manager for authentication
- The token manager then checks with the database to ensure that the token is a valid token
- Token is checked to make sure it hasn't been used
- The validity of the token is then returned to the client side to handle

Token Distribution: We hypothesize three ways to distribute the tokens. The administrator (the person who creates the simulation) would select the type of token distribution that they would like to use at the start of the simulation. For iteration 1 we chose to only implement email token propagation as we believe this to be the most sought after method of token propagation. We believe the other token propagations are not risky, as they do not satisfy any of the three Q's of architecture, therefore we will not implement these until a later iteration.

- Email: Tokens may be propagated to devices via an email link. These emails will be generated by a “TokenPropagator” object and sent to a list of email addresses supplied to the simulation. These emails include a unique link to the server, encoded by the token, which when accessed for the first time registers the token to the device. If the link is pressed after the first time, an error message will be displayed saying that the token is already registered.
- SMS text message: Tokens may be propagated to tdevices via a link in an SMS text message. These text messages will be generated by a “TokenPropagator” object and sent to a list of addresses (phone numbers) supplied to the simulation. These text messages include a unique link to the server, encoded by the token, which when accessed for the first time registers the token to the device. If the link is pressed after the first time, an error message will be displayed saying that the token is already registered.
- Token distribution upon accessing the simulation website: Tokens may be propagated to devices which access the simulation website. This will work as follows: A device accessing the website of the simulation will be given a unique unused token to the simulation if the number of tokens to be distributed has not been exceeded. The token will then be registered to that device and will allow the device to access the simulation on all future visits.

Token Saving: A token, once registered to a device, will be stored as a unique key in the local storage of the device and that browser to which it is registered. A copy of that unique key is also stored on the server side in our database. This allows the keys to be compared in order to verify that this is in fact a valid key. As the key encodes the device and browser it is registered to, only that device on that browser may use it. It is browser specific as the key is stored in the local storage of that browser on that device.

Device and Network Iterators

In order to handle the replicated data type we required an iterator to iterate through the devices and networks. In order to have our code as reusable as possible, we created a ubiquitous “iterator” class, which upon creation takes a collection, which would be either the list of devices or list of networks provided by their respective manager classes and allows iteration through these collection. Using only a single iterator class mirrors an interface in object oriented programming languages and allows us to have more refactorable and cohesive code, providing structure to our project.