|  |
| --- |
| CPE 400 |
| Social Networking Algorithm |
| A Visual Representation & Simulation |

|  |
| --- |
| Saharath Kleips  11-23-2015 |

**Introduction / Contribution**

This project seeks to be used as a visual aid and simulation on the implementation of a “Social Network” routing protocol. A social network routing protocol establishes the concepts of “friends” when transmitting data between large distances. Suppose you are a client, “A,” and you wish to send data to client “B.” If there is a direct connection between A and B, then there is little worry about your data being intercepted or sniffed; your privacy is maintained. Now assume two more clients, C and D, both equidistant away from A and B. C has been known to read any data it receives, however, D is very trusted. In this scenario, you would prefer to transmit data through D to B (A to D to B) over C (A to C to B). This concepts spawns the idea of a Social Network, in which routers / clients / etc. maintain a friends list (a list of trusted peers). A central server would be able to maintain a list of all clients and respond with the most optimal path determined by each client’s friends list. The motivation here is privacy, rather than efficiency, such that if D in the last example was further from A and B than C, A would still choose to transmit through D.

The simulation I have developed maintains a high-level view of how a central server could determine paths between friends (trusted peers). Rather than maintaining a list of each

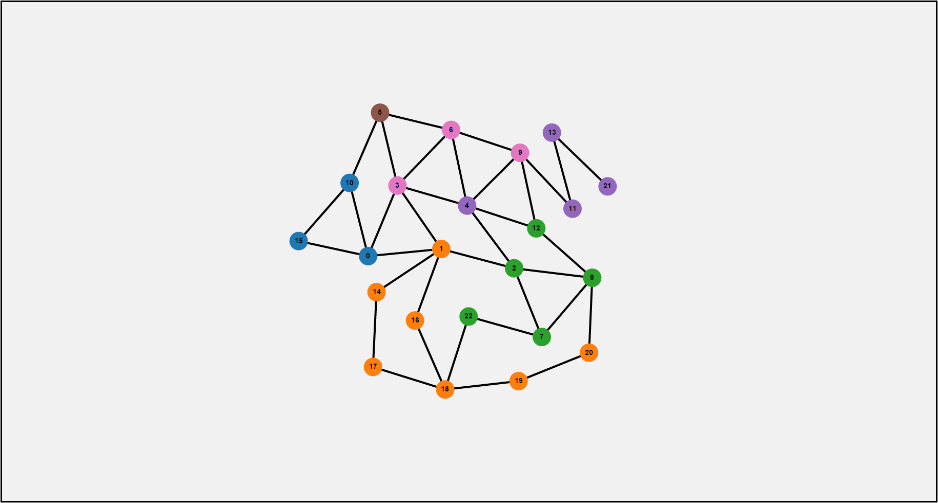


Figure 1: A graph of connected peers whose colors determine their relationship to each other.

Client’s friends, clients are organized into “friend groups” which are represented by color. Figure 1 shows 6 groups (Blue, Brown, Green, Orange, Pink, and Purple) all variously connected to each other. In this simulation, clients only trust paths whose endpoints are within their own circle of friends (the same color). When a client is selected, the entire group’s valid paths light up green. Endpoints whose color does not match the group’s color is highlighted red, which

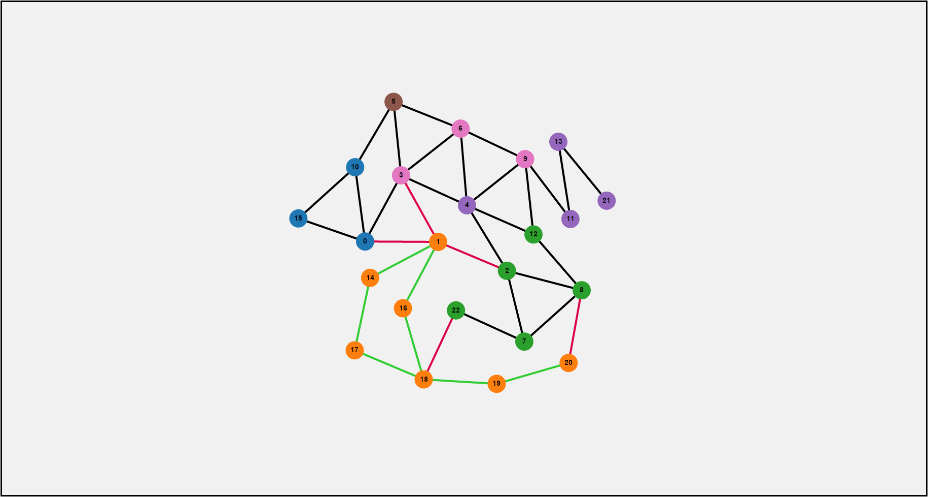


Figure 2: All of Orange's valid paths.

signifies that while the path may be connected, it is invalid. Figure 2 displays that all edges that connect two Orange clients are valid. In the case of Client 20 connected to Client 8, the edge is red due to Client 8 not belonging to the correct friend group. The same case goes for Client 18 and Client 22, and Client 1 and all of his connections. After experimenting with the current

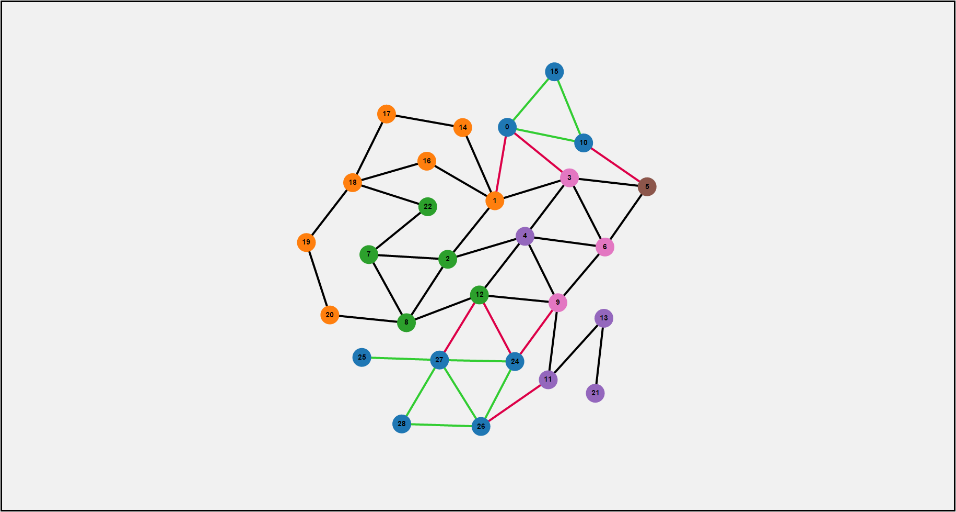


Figure : Two separate groups of friends belonging to the same friend group.

rules and constraints of the simulation, a notable problem appears represented in Figure 3. Figure 3 displays the same graph, however a group of Blues have been added on the complete opposite side of the original 3 Blues. Sending data from Client 24 to Client 0 would be deemed impossible due to the current constraints, as there are no Blue clients which connect the two. They are however, connected… just by incorrectly colored nodes.

**Results**

The Social Network Algorithm has a couple ways to be improved upon. Firstly, rather than creating friend groups, each Client should be able to maintain its own list of friends. This way, groups of friends may actually overlap. However, should the algorithm allow transmission of data through clients who are not friends at all? One solution is to determine if the path being traversed is going through an untrusted source, and heavily encrypting it. This is simple in our algorithm as if the target client is not within the list, heavily encrypt our data. Another issue may also arise, for instance, the path from Client 1 to Client 20 in Figure 2 moves through 16, 18, and 19 before finally reaching its target 20. There is obviously a much shorter path: Client 1 to 2, to 8, and lastly, to 20. While the difference is hops is only 1 (4 versus 3 respectively), it is possible that the difference in paths may get very large. This proposes a scenario in which the increased levels of security might not be worth the decrease in efficiency. A solution to this would simply be to add edge weights based on “the level of friendship” two clients are to each other, and using a path finding algorithm such as Dijkstra’s. For instance edges could first be weighted based on hop length. The edges would then be weighted based on level of friendship, “50” for a first degree friend, “100” for a second degree friend, “150” for a third degree friend, and so on. These numbers may be fine-tuned to adjust the privacy-efficiency trade-off.

**Conclusion**

Overall, the Social Network Algorithm would consist of a centralized server and peers. When a peer wants to transmit data, the server would respond with the appropriate path to for the peer to send to. The server would calculate this path by maintaining a list of all peers, a list of friends for every peer, build a graph where the edge weights are based on distance and degree of friendship, and simply run Dijkstra’s or A\* algorithm through the graph. If the target peer is not within the list of friends, increase the level of security. This revised algorithm accounts for ensuring that every peer within the network may be accessed by every other peer and that it is easily modifiable when optimizing for speed or security. My simulation however, provides a very high-level view of the first iteration of the Social Network Algorithm, how an algorithm based on social networks might work, and the problems that arise with it.

Code Explanation

The simulation was created using Javascript and has been embedded into an HTML page. Simply open “index.html” on a web browser to run the simulation. (The simulation was tested on Google Chrome and may not support IE, although it should support Mozilla Firefox.) If there are any issues, a live demo of the project is being hosted on <https://zarol.github.io/SocialNetworkAlgorithm/>. A block diagram doesn’t exactly best represent this project. I feel a simple description would be more appropriate. The project uses the D3js framework (<https://d3js.org>) in order to build the visual representation of the network. D3js is a graph library that allows me to build responsive graphs. So, the only thing it’s doing in this case is drawing all the nodes, edges, and the physics-based movement you see moving the nodes around. There are two (loosely defined) modules: nodes and edges. A node simply contains a unique identifier and a color which represents what group of friends it is part of. An edge holds two nodes, a source and a target, as well as a state which determines what color to make it. The algorithm for determining the paths is as follows: iterate through all of the edges, if the source and target colors both match the selected node’s color, make the edge green. If only one of the colors match the selected node’s color, make the edge red. All of the user-interaction can be explained in the in-code documentation.