Rubrics for the project grading CPE 400: 20% of the total grade

Project Submission due by Wednesday, Dec 5th

Deliverables:

1) Code:

a) [5] Turning in code that compiles and runs properly

i) Code compiling properly and providing results as explained in the report: 5;

ii) Code compiling but not providing results as explained in the report: 2;

iii) Code not compiling: 0

b) [2] Documenting the code

i) Clear explanation of the entire code: 2;

ii) Partial explanation of the code: 1;

iii) No explanation of the code: 0

2) Technical Report

a) [5] A report explaining the functionality of the protocol

i) Protocol/functionality addressed extensively in details with as many realistic aspects as possible + error handling scenarios: 5;

ii) Protocol/functionality addressed with some details and some error handling but not extensive: 3;

iii) Unclear explanation of the protocol: 1;

iv) No explanation: 0

b) [3] Out of the box thinking: novel contribution

i) Important Novel Protocol/functionality contributed with details: 3;

ii) Important Novel Protocol/functionality contributed but explanation not clear: 2;

iii) Novel Protocol/functionality mentioned but no explanation provided: 1;

iv) No Novel contribution provided: 0

c) [5] Results and analysis of the results

i) Results and analysis contributed with details: 5;

ii) Results and analysis contributed but explanation not clear: 3;

iii) Results mentioned but no analysis provided: 1;

iv) No results provided: 0

Functionality of the Protocol

What protocols will be utilized in this simulation?

* DSR Optimization: Routing Caching

Novel Contribution

Results and Analysis of the Results

CPE 400

Nicholas Mason, Nicholas Thom, Sybille Horcholle

November 26, 2018

Dynamic Routing Mechanism Design in Faulty Networks

Introduction

The topic addressed in this project finds a creative solution for dynamic routing in faulty, mesh networks. In any given network, whether it is a personal area network (PAN), wide area network (WAN), or local area network (LAN), nodes are connected by links to create a mesh network. Different nodes in a mesh network can communicate with each other by establishing links between nodes. By communicating with a neighbor node, a particular node can find the most efficient path to get data to and from a desired node. In many cases, nodes and links fail intermittently. When simulating a mesh network, it is important to implement early detection of link and node failure in order to avoid a networking failure. This code will label “dead-end” nodes to improve the efficiency of finding the quickest path from one node to another. Dijkstra’s Algorithm will be utilized to ensure efficient and reliable results in finding the shortest path between nodes. In addition, the code will take the probability of a node failing into consideration when applying Dijkstra’s Algorithm to find the shortest path possible. If a node is prone to failing, that particular path will not be utilized.

Functionality of the Protocol

This code has been written in the Python scripting language to ensure easy readability of the source code and show the functionality of the code by implementing a simulation of the mesh network. Dynamic Source Routing (DSR) is the primary platform used in the creation of this project. The code implemented in this project takes a DSR platform, and incorporates Dijkstra’s Algorithm to find the shortest path possible between a start node and an end node. A critical concept to handle in this project is failure between nodes in the shortest path between a start and end node. Each node is given a random probability to fail at the beginning of the simulation, which will be stored for the program to consider when creating the shortest path. The program utilizes a random number generator to give a probability of failing for each node. A second number is randomly generated for each node, resulting in a node failing if it is less than or equal to the first random number generated for the given node.

In addition, a failure flag has been initialized to let the program know immediately upon starting which nodes have failed. An active flag allows the program to know which nodes are safe to use in the process of transporting data. Additional code functionality for the node class includes checking to see if a given node has a neighbor, and what information the node holds. Checking to see if a neighbor has been visited has also been implemented to ensure a node can visit all its neighbors efficiently. This gives the program a better chance of avoiding the faulty nodes entirely, choosing a more efficient and safe path to transport information.

Novel Contribution

When a node fails, the first principle a network uses to resolve the problem is to find a new route through a different link to the needed node. However, if the faulty node proves to be the sole node in the necessary link to get information from a host to an end node, the network will be unsuccessful in transporting the data. The novel contribution added to this program is the ability to create a new link between nodes if a faulty node proves to be pivotal in the path from node A to node B. In addition, creating a flag for dead-end nodes will allow the program to only venture to that given node on a given path if it is indeed the host node or the end node. The implementation of Dijkstra's Algorithm will allow the simulator to choose the best routing path to achieve the optimal efficiency. Incorporating various flags for faulty nodes and links, along with dead-ends, will allow the program to find the shortest path between node A and node B in a given data packet transfer.

Results and Analysis of Results

Through the process of simulating a mesh network, along with incorporating the functionality presented in this project, faulty nodes and links were able to be detected and avoided in the data transfer process.

Fig. 1: Figure 1 shows the mesh network simulated to give accurate results of the efficiency of the program.

Fig. 2: Figure 2 shows the results of being able to detect the probability of failure of a given node.

Conclusion