Wireless Link Reversal Algorithm Simulation

Charles Wang
Information Trust Institute
University of Illinois Urbana-Champaign
IL, United States
Charleswang007@gmail.com

***Abstract*— This project is an extension of the Ad-hoc Walkie-Talkie project by UIUC PhD Lewis Tseng and my colleague UIUC undergraduate Pratch Piyawongwisa. Ad-hoc Walkie-Talkie is an application simulating walkie-talkie on top of an ad-hoc wireless network. A wireless sensor network comprises a number of sensor nodes, each of which has limited energy supply as well as intrinsic sensing and communicating capabilities. This paper focuses on link reversal algorithms, specifically the full reversal algorithm, which provides a simple mechanism for routing in ad hoc wireless distributed networks. This project involves four phases: design, implementation, simulation and analysis. Our main goal is to build a simple routing simulation tool for future studies. Hopefully our proposed simulation program will benefit the UIUC Wireless Networking Group, which conducts research on various aspects pertaining to wireless networking.**

Keywords-link reversal, event-driven simulation

I. INTRODUCTION

Ad-hoc Wireless Walkie-Talkie employs communications on top of single-hop IP broadcast, sending UDP packets to the broadcast address, *.*.*.255, with Ghazale's mutual exclusion algorithm as its core application. The person who requires token becomes speaker and can broadcast messages to nearby walkie-talkies via exchanging heart-beat message with each other periodically. In the initialization stage, a phone chooses the phone with largest ID to be its parent. After one phone acquires the token, all phones use Ghazale's algorithm to exchange tokens.

Ad-hoc and sensor networks are emerging area of research that has brought up many interests and attention. "A mobile ad hoc network is a temporary interconnection network of mobile wireless nodes without a fixed infrastructure. In the Ad-hoc Walkie-Talkie project describing above, the actual implementation of ad hoc network is already completed. This paper therefore aims to understand the fundamentals of ad hoc and sensor networks based on computational algorithms and theories. We focus on link reversal algorithms for Ad hoc Wireless Distributed Networks.

The first link reversal routing algorithms were introduced by E.M. Gafni and D.P.Bertsekas in 1981[10]. The full reversal algorithm is obvious and straightforward if an ad hoc network were converted to a destination oriented graph. The full reversal algorithm works by one simple rule: Whenever a node becomes a sink, it reverses its entire incoming links [7]. One of the advantages of the full reversal algorithm is that it does not need to update at every link failure. "If a link gets lost for some reason, there might be still another directed path to the destination from every node and so the graph could still be destination oriented."[7]

We implement the famous event-driven simulation to simulate unexpected hazards occurring during the normal operation of link reversal algorithms. Event-driven simulation is a modeling technique in which flow of control within the system is driven by events rather than sequence. Furthermore, events are managed dynamically by an event scheduler, typically represented by an event queue, which triggers the simulation of hazards in our ad hoc wireless network.

We simulate the full link reversal algorithm and display the result on screen in the form of network topology. "Network topology is a representation of the interconnection between directly connected peers in a network [2]." In this paper exclusively, network topology refers to the network connection between different walkie-talkies. The actual topology of the fast-evolving Internet network is difficult to be graphed. Network topology constantly changes as nodes and links join a network, personnel move offices, and network capacity is increased to deal with added traffic. Keeping track of network topology manually is a frustrating and often impossible job. Yet, accurate topology information is necessary [2]. However difficult it may sound, graphical information is essential and invaluable for network simulation and management. In the analysis phase, we make as few assumptions about the network as possible and quantitatively evaluate the performance of the full link reversal algorithm. By doing so, we aim to have a better understanding of distributed ad-hoc network and its various routing algorithms.

II. DESIGN

The design process is critical for our actual implementation of the link reversal simulation tool. Six main issues need to be considered during the design phase: Graphical Layout, Mobility Control, Network Properties, Link Reversal Algorithms. Destination Oriented Graph and Event-Driven Simulation

A. Graphical Layout

We would like to model the deployment of our ad hoc walkie-talkie mobile phones utilizing a simple mathematical topology program. More precisely, we imagine that the network nodes are placed on a graphical Euclidean plane, with two-dimensional Euclidean space denoted $\mathbb{R}^2$. In the Euclidean plane, the vertices are the wireless nodes. Java applets are popular and frequently used in computer graphics. A Java applet is an applet delivered to users in the form of Java bytecode and can run in a Web browser using a Java Virtual Machine (JVM). To display and print 2D graphics on our Java program, we need also classes from the Java Abstract Window Toolkit (AWT) for producing actual output. The AWT contains all of the classes for creating user interfaces and for painting graphics and images.

Under the AWT package, we construct graphical objects from the class Graphics, the abstract base class for all graphics contexts that allow an application to draw onto components that are realized on various devices. Coordinates are infinitely thin and lie between the pixels of the output device. The default origin of user space is the upper-left corner of the component's drawing area. The x coordinate increases to the right and the y coordinate increases downward. Hence the top-left corner of a window is (0, 0). All coordinates are specified using integers.

B. Mobility Control

Recall that in ad hoc wireless network, nodes are moving and as a result, the underlying communication graph is changing over time, and the nodes have to adapt quickly to such changes and reestablish their routes [8]. Say node A moves gradually from (x1, y1) at time t1 to (x2, y2) at time t2. Since our measure the network time is in unit of seconds, we simply calculate the displacement each node moves in one second and therefore figure the node's position at every second. From our example above, if time t is between t1 and t2, then at time t node A is at position

$$(\mathrm{x1} + \frac{x2 - x1}{t2 - t1} \times t, \mathrm{y1} + \frac{y2 - y1}{t2 - t1} \times t)$$

C. Network Properties

We use a mathematical model to quantitatively evaluate the performance of the Ad-hoc network, at the meantime struggling to make as few assumptions about the network as possible. One of the most important properties of communicating nodes is their broadcast capability. "Two nodes can communicate if they are within their mutual transmission range, which in an unobstructed and homogenous environment translates into whether their Euclidean distance is at most the maximum transmission range R. Thus model is widely known as unit disk graph [5]." We thus set the maximum transmission range as the simulator input.

In a typical sensor network application, sensor deployment and coverage are both significant. For a given configuration and placement of sensors, it is important for the deployed collection of sensors be able to communicate with one another. It follows that a connected sensor network requires good enough coverage property and connected property. Sahni and Xu in [2] mention that "for the coverage property, we need to know the sensing range of individual sensors (we assume that a sensor can sense events that occur within a distance *r*, where *r* is the sensor's sensing range) and for the connected property, we need to know the communication range, *c*, of a sensor. To simplify the calculation, we follow Kar and Banergee's [32] assumption that the sensing range equals the communication range (i.e., $r = c$)

Let's imagine a scenario where five phones together form an ad hoc network. The communication and sensing range for each phone is the same. We model this scenario using five unit-disk graphs, as shown in Figure 2. Now we assume phone A takes the token and begins broadcasting message to its neighbors, as shown in Figure 3. With $r = c$, three different broadcast result can be generated: not connected, somehow connected, and well-connected. The three scenarios are shown in Figure 4, 5, and 6, respectively. In Figure 4, each unit disk barely touches its neighbor and does not overlap; thus those nodes are not connected. In figure 5, each unit disk overlaps with its adjacent neighbors. As a result, phone A can broadcast to B and C, but not D and E. We describe this situation as somehow connected. Finally, Figure 6 displays a situation where one unit disk overlaps with the remaining disks. This is a well-connected network.

D. Link Reversal Algorithms

Busch, Surapaneni and Tirthapura give a general description of link reversal algorithms. "Link reversal algorithms provide a simple mechanism for routing in communication networks whose topology is frequently changing, such as in mobile ad hoc networks. A link reversal algorithm routes by imposing a direction on each network link such that the resulting graph is a destination oriented DAG. Whenever a node loses routes to the destination, it reacts by reversing some (or all) of its incident links. [8]" In this paper, the full reversal algorithm is our primary focus.

In a mathematical modeling graph, a path is a sequence of vertices such that each adjacent pair of vertices is connected by an edge. If the graph is directed, the edges that form the path must all be aligned with the direction of the path. The length of a path is the number of edges it traverses. In addition, a graph is strongly connected if there is a path from every vertex to every other vertex. Moreover, degree of a vertex is the number of edges incident on that vertex.

In our graphical representation of the network, each node has a link with each other node within its broadcast radius. In the beginning, this underlying graph is undirected, i.e. the communication links are all bidirectional. We then transform the undirected graph to a destination-oriented graph using our so-called ranking algorithm, which will be introduced shortly.

For full reversal algorithm, "when a node finds that it has become a sink (has lost all of its outgoing links), then the node reacts by reversing the directions all of its incoming links. The link reversals due to one node may cause adjacent nodes to perform reversals, and in this way, the reversals propagate in the network until the routes to the destination are reestablished. [8]"

To implement full reversal, we decide to assign an "arrow-in" data structure to each node. Each node keeps its own record of "arrow-in" list, where information regarding its neighbor links' destination is stored in "arrow-in" objects, each of which is a tuple in the form of (Neighbor, In). Neighbor is the ID number of the connected node, and In tells the direction of their link. 1 means the link is incoming to the link; 0 means the link is outgoing from the link. If the number of incoming links equals the degree of convex of a node, we say that the node becomes a sink, and its connected links are reversed.

E. Destination Oriented Graph

To produce a graph that is destination oriented, we assign each node a rank number and use the number to determine the direction of each link. We set the rank number of the destination node to be 0. Then neighbors of the destination node are assigned a rank number 1. Then neighbors of neighbors of the destination node are assigned a rank number 2. A node cannot not be assigned a rank number more than once, and the process continues until all the nodes in graph have their own rank numbers. Each node generates a tuple (Rank, ID), with Rank the rank number and ID their identification number. We assign the direction of each link based on information gathered from every tuples. If n is an integer ranging from 1 to the maximum rank number assigned, we draw an arrow from nodes with rank number n, to nodes with rank number n-1. If two nodes have the same rank number, we draw an arrow from the node with higher ID number to the node with lower ID number. For instance, if node A has tuple (Rank_A, ID_A) and node B has tuple (Rank_B, ID_B), a directed link from node A to node B is generated if Rank_A > Rank_B, or Rank_A = Rank_B but ID_A > ID_B.

The resulting destination-oriented graph is said to be acyclic because every directed path in the graph leads to one single destination. In addition, with our proposed ranking algorithm, a link cannot have two different orientations. With aid of the destination-oriented graph, routing simulation becomes fairly easy: a node forwards the receiving packet on any outgoing link, and the packet will eventually reach the destination.

F. Event-Driven Simulation

We mentioned above that event-driven simulation is a modeling paradigm in which flow of control within the system is driven by events rather than sequence. Despite the concern about performance, event-driven simulation has several advantages. First of all, events are flexible and managed dynamically by an event scheduler. Furthermore, an event scheduler handles both synchronous and asynchronous models with arbitrary timing delays. Just as most literature on the subject of event-driven simulation, we represent our event scheduler using an event queue. The event queue performs three separate tasks at every time step: 1. Removing items from the event queue due for processing. 2. Performing appropriate action/updates/etc. 3. Future events being added to the event queue if they are determined and those events will be processes at the appropriate time-step.

Events are placed in the event queue as objects, with fields *Time*, *Action* and *Data*. *Time* tells the event scheduler when the event should be triggered. Action tells the event scheduler what the event does. Finally, data is the information needed for execution of the event. As the event scheduler assign events at every time step, executing an event triggers a process which may generate more signal assignments to be placed on the event queue. We believe that the simplicity of event-scheduler data structure helps us understand fundamental issues in real-world ad hoc wireless network scenario, where hazards happen irregularly.

III. IMPLEMENTATION

We use Java for actual implementation. The architecture of our full link reversal simulator is as follows. A main program and two auxiliary classes comprise the simulator. The main program triggers the thread and does graph painting, while two auxiliary classes, Event and EventPriorityQueue, are the core data structures for our implementation of the event-driven simulation. Since we built the link reversal simulator from scratch, our software development process is step by step, as shown in the hierarchical order of several main programs: Move, Reversal, Reversal_1, ReversalEventDriven and ReversalMultipleNodes. Illustration of simulator architecture, auxiliary classes and main program are shown in Figure 7, 8 and 9. Next we give brief description of these classes.

IV. SIMULATION

After successfully implementing our link reversal simulator in Java, we are excited to test out its functionality. We simulate at least one scenario for each of the main programs: Move, Reversal, Reversal_1, ReversalEventDriven and finally ReversalMultipleNodes. We herein attach screenshots of the running program for each of simulation examples.

We put together previous efforts and create our final version of link reversal simulator, which can be functional on a web browser. The Link Reversal Simulator is the final product of the 6-week ITI (Information Trust Institute) summer research program at University of Illinois, Urbana-Champaign. Our simulator analyzes and predicts what happens in an ideal wireless sensor network. With

features such as mobility and topology control, we believe our simulation tool will benefit the UIUC Wireless Networking Group, which conducts research on various aspects relating to wireless networks.

V. Analysis

After building our simulator, we would like to generate some tests and analyze its performance. In [8]. Busch and others present the first formal performance analysis of link reversal algorithms. Specifically, they study these algorithms in terms of *work* (number of node reversals) and the *time* needed until the network stabilizes to a state in which all the routes are reestablished." They reach a conclusion that the full reversal algorithm requires $O(n^2)$ work and time, where n is the number of nodes which have lost the routes to the destination.

VI. Conclusion

So far we have shown detail progress for our Link Reversal Simulator during four phases: Design, Implement, Simulation and Analysis. In reality, although routing algorithms maintain routes to any particular destination in the network, the lack of a fixed infrastructure makes routing between nodes a hard problem. Using the simulator, we have observed how reversals propagate in the network until the routes to the destination are reestablished. Finally, we sincerely hope our simulator will further future research and course work.

VII. Acknowledgment

Special thanks to the guidance from Prof. Nitin Vaidya.

VIII. References

[1] S. Sahni and X. Xu, "Algorithms for Wireless Sensor Networks," University of Florida, Gainesville, FL. September 7, 2004.

[2] R. Siamwalla, R. Sharma and S. Keshav, "Discovering Internet Topology," Cornell Network Research Group, IEEE Infocom, 1999.

[3] Z. Shen, Y. Chang, C. Can and X. Zhang, "A Topology Maintenance Algorithm Based on Shortest Path Tree for Wireless Ad hoc Networks," National Key Lab of Integrated Service Network, Xidian Univeristy, Xi'an, China

[4] P. De and S. K. Das, "Epidemic Models, Algorithms and Protocols in Wireless Sensor and Ad-hoc Networks," University of Texas at Arlington, TX

[5] R. Wattenhofer, "Algorithms for ad hoc and sensor networks," ETH Zurich, Zurich, Switzerland. April 2005.

[6] K. Kar and S. Banerjee, "Node placement for connected coverage in sensor networks," Proc WiOpt 2003: Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks, 2003.

[7] N. Born, "Analysis of Link Reversal Routing Algorithms for Mobile Ad Hoc Networks," Seminar of Distributed Systems WS. April 2005.

[8] C. Busch, S. Surapaneni, S. Tirthapura, "Analysis of Link Reversal Routing Algorithms for Mobile Ad Hoc Networks," July 2003.

[9] T. Lammle, "CCNA(Cisco Certified Network Associate) Study Guide," 1999.

[10] E.M. Gafni and D.P. Bertsekas, "Distributed algorithms for generating loop-free routes in networks with frequently changing topology," IEEE trans. On commun, COMM-29:

[11] D. Miorandi, H.P. Tan and M.Zorzi, "Ad Hoc Networks with Topology Transparent Scheduling Schemes," Dep. of Information Engineering, v. Grandenigo 6/B, 35131 – Padova (Italy)

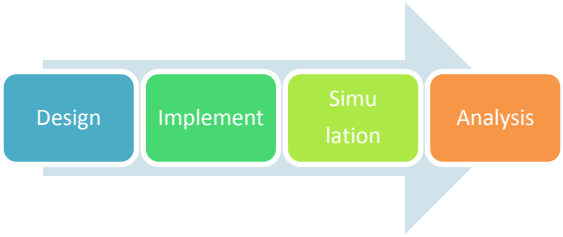

Figure 1: Project Phase Flow Chart

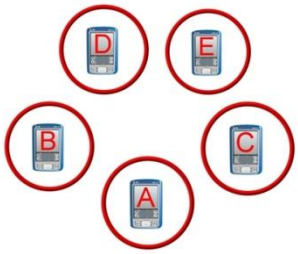

Figure 2: Five wireless phones, each of which has communication range c, modeled as radius of the unit disk

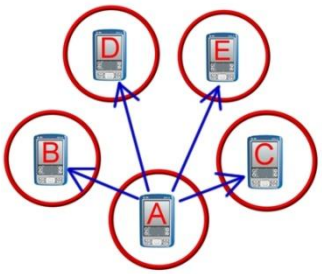

Figure 3: Phone A attempts to broadcast messages to B, C, D, and E.

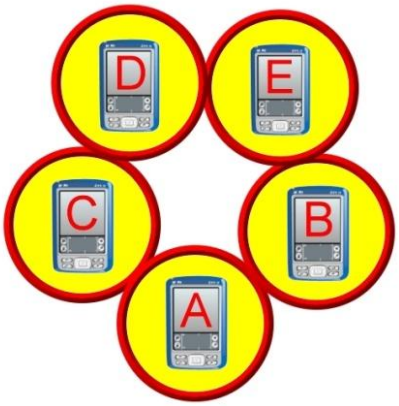

Figure 4: Not connected

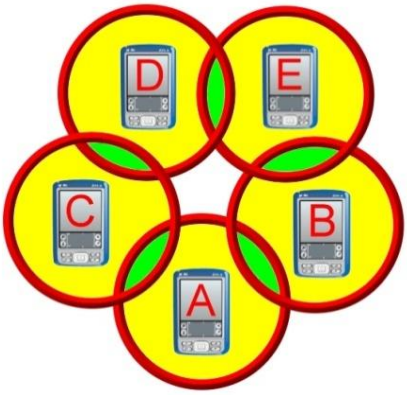

Figure 5: Somehow connected

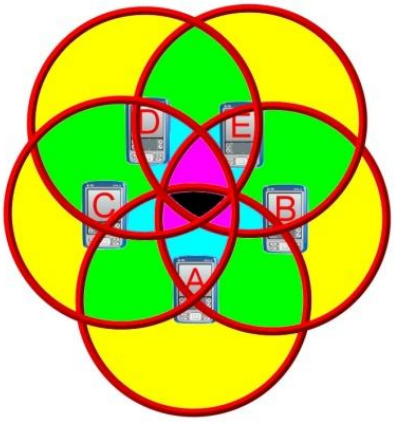

Figure 6: Well-connected

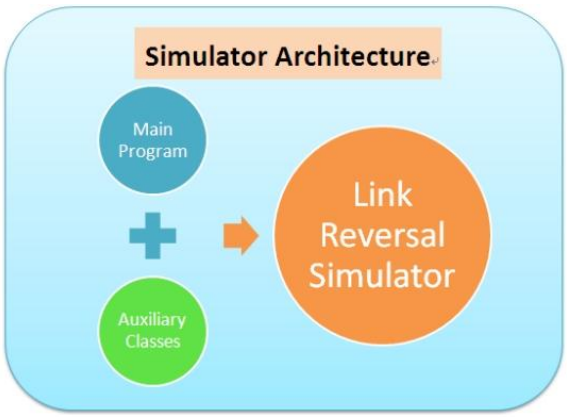

Figure 7: Link Reversal Simulator Architecture

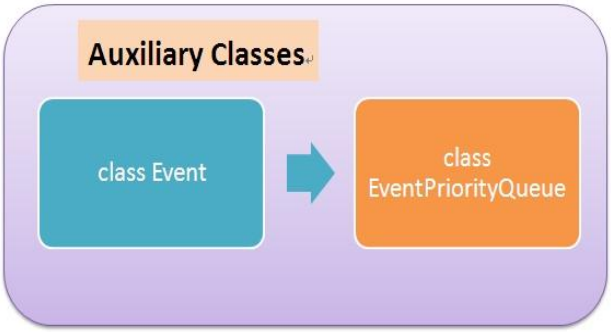

Figure 8: Illustration of Auxiliary Classes

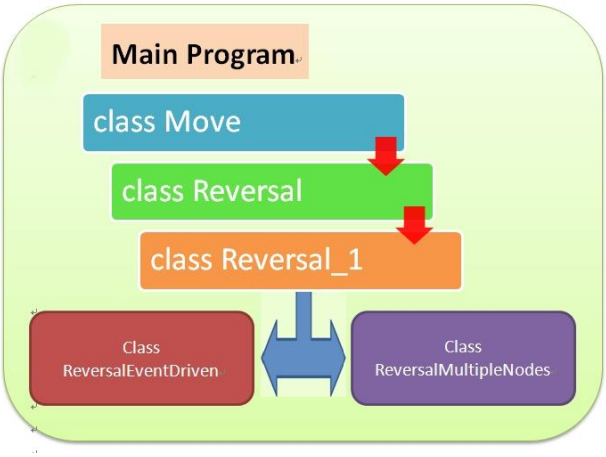

Figure 9: Illustration of Main Program