SWAN Summer Research Summary

Student: Samuel Nelson, Class of 2006
Faculty Mentor: Professor L. Felipe Perrone, Department of Computer Science

Computing Connectivity Metrics

The purpose of our research this summer (2004) was to advance the SWAN project by augmenting the simulator to study the effects of denial of service attacks on a wireless ad-hoc network. Primarily, we wanted see how three types of denial of services attacks (range, reboot, and jamming) would effect network metrics. Before we started working, SWAN could output data that, when mined properly, gave specific information about the network layers of each node. This allowed metrics such as packet delivery ratio, total application messages sent and received, total control messages sent and received, and control to application message ratio to be computed. While these were helpful, it would not allow us to see how the network connectivity changed over time. Therefore, our first step in the research was to search for metrics that could accurately represent how connected a graph was.

This search led us to choose seven metrics that we would implement into SWAN. Because the connectivity graph in SWAN is stored as an adjacency matrix, we added an array such that every element corresponded to a node and held the node's vertex count. This allowed linear-time vertex lookup, which is critical for all of our metrics. The first three metrics were the minimum node degree, maximum node degree, and average node degree of the graph. These were easily computed and implemented. The next metric added was the Randic connectivity index, created by Randic for use in studying molecular bindings. To compute this metric, we summed up the weights of all of the edges, where the weigh of each edge was equal to the product of the two nodes' degrees (that corresponded to the edge) raised to the ½ power. The final three metrics were implemented using the QuickUnion graph algorithm. The first was the number of components, which returned how many connected components the network had. So, a completely connected network would have one component. The second was the connectivity efficiency which was computed by summing up all of the connections in one or more hops, and then dividing by all of the possible connections in one or more hops if the network were completely connected. The final one, which we called the connectivity index, was computed by summing up all of the connections in one or more hops, and then dividing by the total number of nodes squared.

Next, we implemented these metrics in a isolated C++ project and tested them on small networks. We then compared the results to those done by hand, verifying that the metrics were being computed correctly. Finally, we added the metric implementation to SWAN. Now it was time to allow the user to interact with the metrics.

Recording and Reporting Connectivity Metrics

With the connectivity metrics in place, the next step was to decide how they should interact with the simulation. In order to give control to the user, we created an entire new

section of the DML input dedicated to reporting these metrics. The new section, starting with the keyword *report*, would be able to have a section for each network in the simulation. Inside each network block, we decided that the user should be able to control the following parameters: how often to recompute the network graph (which I'll talk about in the next paragraph), which metrics to be computed, the time interval for recomputation of the metrics (each metric can specify its own interval), and the output filenames for each of the metrics. In order to accomplish this, we created and registered a report class, which would serve as a base class for any other future metrics. All of our metric computation went in a subclass of the report class, and contains the DML configuration tools necessary to process the report section of the DML.

Previously, SWAN had no need to recompute connectivity after the simulation started. Therefore, we had to change SWAN to allow for the recomputation of the network adjacency matrix at intervals specified in the report section of the new DML. Due to all of these changes, we could finally receive output data that gave us a time series for each of the network connectivity metrics. In other words, we could now see how the network connectivity changed over time.

Built into SWAN is a recording method that allows one to store data in memory and write it all to disk after the simulation finishes. This increases efficiency a lot due to the physical limitations of quickly writing to a disk. To speed up SWAN, we made use of this recorder class to store our metrics and then wrote them to disk at the end. Currently, SWAN only uses text based records which are not as ideal as binary records. In the future, SWAN will be able to write records in binary mode, saving memory space. This will also force the addition of an external player whose job will be to convert the cryptic binary text into a human readable form.

Host Ranges

Now that SWAN had the ability to compute and record connectivity metrics, we started thinking about what would be needed to perform attacks. One of the main types of attacks is called the *range* attack, where selected hosts are chosen to switch ranges during certain times in the simulation. A major prerequisite for this attack is the hosts' ability to change ranges during the simulation. SWAN did not yet have this ability, so we started by creating a variable range radio network class that had all of the functionality of a fixed range radio network, except that when the network graph's adjacency matrix was recomputed it took into account each host's individual cutoff range. This also required that each host be allow to specify its own cutoff range in the DML, along with a global maximum cutoff range that each host had to abide by.

With each host now having its own cutoff range, we had to rethink the way the adjacency matrix was being computed. Because the connectivity graph was bi-directional (there was no need for a uni-directional graph because for transmission to occur both parties much be able to send to each other) we decided that both nodes being looked at could only have an edge if they both were in each others ranges. When calculating the

adjacency matrix, first the global cutoff is used. If the host is outside of the global cutoff, then it is immediately known that no edge exists. If the global cutoff test passes, then a check is made to see if both hosts are within each other's individual cutoff range. If so, then an edge is created. If not, then no edge is created.

It is easy to see the necessity of a variable ranged network when dealing with a range attack scenario. Another use for this addition, which can be implemented at a latter time, deals with non-malicious nodes. For instance, if a node detects the presents of a large number of neighbors, it can diminish its transmission range and still remain connected. The advantage of this is that it will save a substantial amount of battery life, as a large percentage is used in transmitting data. This also could allow the opposite to happen; a critical node in the network could suddenly fall out of the range of any other nodes and increase its transmission range in order to prevent a drop in connectivity. These types of intelligent nodes would be a useful contribution to a wireless network and having the ability to simulate node range changes will allow SWAN to simulation this in the future.

Multipurpose Application Layer

In order to create a more realistic attack scenario, we have developed a single application layer that has the ability to simulate normal, rebootable, and range-changeable nodes. Previously, SWAN nodes had the ability to become attack nodes, but with limitations. One of the biggest limitations was that they couldn't send data, only relay it. To be a host, or send data, the node had to use the old normal application layer. In the new application layer, a host can be an attacker or be normal and still send data.

Combining attack application layers with the normal application layer has inherent advantages other then allowing attacked nodes to send data. For one, it allows many parameters to be shared, so the user can use the same familiar parameters for all attacks. Some of these shared parameters include: at, at_window, attack_on_time, and attack_off_time. The attack on and off times were made to be either deterministic or uniformly random inbetween some specified interval. Another parameter that was added is the probability of a node being an attacker. This allows attackers to be randomly chosen during the scenario, which is a more realistic situation.

Another advantage is simplicity for the programmer and the user. Because the application sessions for all of the attacks would share over half of the same code, it is the logical choice to integrate them into one. The user would find it easy as well, because he/she would have to write a dictionary entry for only one application layer and change the attack type to whatever was desired. Currently, only reboot and range attacks are integrated in the application layer, but its design will allow for easy future integration of a jamming attack.

Perl Scripts

At this point, SWAN was ready to start running the simulations. We discussed how to vary different parameters, and ended up with hundreds of scenarios that we wanted to

run. To make this manageable, a series of perl scripts was created to easy the process of running simulations.

The first file that was created was a template DML file, where dummy characters were inserted in place of parameters that we wanted to vary. Two scripts, executeRange.pl and executeReboot.pl, were created along with there data components, executeRange.dat and executeReboot.dat. The perl scripts would read in data from the data files and replace the temporary DML file dummy variables with the real parameters found in the data files. Then the scripts would execute the run and save the results in correspondence with a naming scheme that is found in the lookup tables. Next, two data mining scripts, averagerConnectivity.pl and averagerExtra.pl, would look through the output data and create files that would represent the average for each scenario. For example, if five scenarios were run, there would be fifty total runs (ten runs per scenario) and therefore fifty total output files for each metric. The averager scripts would average the ten runs for each of the scenarios, and leave the user with five total average files for each metric. Next, plotConnectivity.pl and plotExtra.pl would read in these average files and plot graphs for them in accordance with the parameters set for the plot scripts. These scripts would output postscript and pdf files with the naming schemes that are used in the plot lookup tables. With this information now available, we could study and analyze the data obtained.

Analyzing Data

In order to study the results, we asked ourselves two questions that would be interesting for an attacker to know, and then tried to answer them with the graphs. The first was, do synchronized attacks have the same effect as unsynchronized attacks? The answer was a definite no. The unsynchronized attacks with long attack periods had a higher peak damage done then synchronized attacks. We also discovered that to cause the highest peak damage one would prefer to increase the attack on time, and to prevent recovery one would prefer to decrease the attack off time.

The next question we asked was whether or not the network was effected differently by equivalent on off ratios. In other words, would an on/off time of 10/10 have a different effect then an on/off time of 20/20? We found that this was the case for unsynchronized attacks. When the on/off intervals are small, less damage is done but less recovery is made. When the on/off intervals are large, more damage is done but more recovery is made. This is intuitive, but necessary to confirm. Next we attempted to analyze the packet delivery ratio metrics, with unexpected results. After much searching, we decided that our errors were coming from the AODV layer not being rebooted properly, which still has to be scrutinized.

Future

Right now, we have information about how the network connectivity changes over time for reboot and range attacks. In the future, we would like to fix the AODV layer, or find a new way to computer packet delivery ratio. In the AODV, it appears that it shuts down

properly, but then will not bootup when it's expected to. There seems to be two sections to the boot up process. The first is when the attack is over and the application layer, as well as the other layers, all come up. This correctly works for the AODV layer. The next section is when the AODV layer is allowed to relay packets. There is a delay period specified in draft 10 of the AODV protocol, labeled DELETE_PERIOD. This is currently set to 50 seconds, and is a protection against routing loopbacks. We believe that the reason this is not being called properly is because the queue that the sets off the timer is somehow not working as we anticipated. This queue, called QueueTimer, will have to be looked at closely in the future.

Another helping addition in the future would be to integrate the jamming application layer into the multipurpose application layer. This would not be very difficult, as one can just follow the lead of the other two integrated attacks. The difference is that jamming also has a physical layer that may or may not be integrated into a multipurpose physical layer. Because jamming nodes emit signals constantly, the parameters of the application layer will be different then that of the other two attacks. These problems will have to be discussed and worked out in the future.

Finally, a nice future addition to SWAN would be to enable intelligent hosts that could detect when they have enough neighbors and can change their transmission range, via the variable range radio network that we created, therefore saving battery power. This ability could be incorporated into each individual host, and experiments could be run comparing the battery life of a normal network to the battery life of a power-saving network.

I personally feel that I have learned a tremendous amount of information pertaining to wireless networks (and networks in general) as well as security on these networks. Before starting research I knew very little about network protocols, let alone wireless networks. One of the most interesting topics was learning about the network layers and how they worked together to create a complete network. This is a topic that I knew existed before, but never knew how the layers interacted with one another and what function each served. I also learned a great deal about how attacks effect networks and how networks can recover from them. Working on this project has sparked my interest in networks and network security, both topics which I will continue to explore.