ECE 6110 Lab Assignment 3: Mobility Models

In this lab, you will compare and contrast the random walk and random waypoint mobility models and do a very preliminary investigation of wireless performance vs. communication distance. This lab builds on the discussions in class of the paper on steady state spatial distributions of random walk and random waypoint mobility models, which can be found at:

https://blough.ece.gatech.edu/6110/mobility_steady_state_distributions.pdf

Turn-in Instructions: Create a separate folder for each part of the assignment inside a main folder named "Lab3_<your_last_name>_<your_first_name>". Then create a single tarball of the main folder and all of its subfolders and submit it in Canvas.

Important programming instructions: All code must be hand typed in your editor of choice. No auto-formatting is allowed to change the text that you type in manually. Also, make sure to use the exact program name and command-line parameter names specified in each part. You will get points deducted if you do not follow these instructions even if your code is functionally correct!!

Part 1: Comparison of Random Walk and Random Waypoint Mobility

The core of your program should distribute nodes uniformly at random in an 80m x 80m square area and move them according to the random walk or random waypoint mobility models. Command-line parameters of your program should be the number of nodes, type of mobility model, duration of simulation, minimum and maximum node speeds (for both models), and pause time (for the random waypoint model). Name these parameters "numNodes", "mobility", "duration", "minSpeed", "maxSpeed", and "pause". The "mobility" parameter should be of type std::string and you should input "walk" to specify random walk and "waypoint" to specify random waypoint. The random walk model should be set to use the "Time" mode for course changes with 2 seconds in between each change for all simulations.

Your program should output the positions of all nodes (x, y coordinates) at the end of the simulation run and their distances from the center point of the region. You can use the GetPosition() method to find the nodes' positions. An example usage of this method is:

```
Ptr<MobilityModel> mmp = nodes.Get(0)->GetObject<MobilityModel>();
Vector apV;
apV = mmp->GetPosition();
```

Notes:

1. To distribute nodes uniformly at random in a square, you can use the RandomRectanglePositionAllocator. For an example usage of this position allocator, refer to $NS3/src/dsdv/examples/dsdv-manet.cc, where $NS3 is your main ns-3 directory.
2. For debugging, it's recommended that you output the positions of the nodes prior to starting simulation and trace course changes for a few nodes during the simulation to verify that the initial uniform random allocation and mobility model are both working correctly. Checking initial positions can be done with GetPosition() as described above and course change tracing can be done as shown at the end of Sec. 7.3 in the ns-3 tutorial. Once you've verified that these pieces are working correctly, you can remove those outputs to speed up your simulations.
3. For random waypoint mobility, you will need to use the SetPositionAllocator() method to initialize the node locations and the "PositionAllocator" argument in the SetMobilityModel method to choose the random waypoints during simulation. Both of these can be set to the same random rectangle position allocator.

Part1a: For 100 nodes, speeds uniformly distributed between 4.0 and 6.0 m/s, a pause time of 2.0 seconds for random waypoint mobility, and a simulation duration of 400 seconds, plot the locations of the 100 nodes at the end of the simulation time for both mobility models and calculate the average distance from the center point of the region averaged over the 100 nodes to have a simple single metric to compare the results. Explain the results based on class discussions of the assigned paper that experimentally evaluates the steady-state spatial distributions of the two mobility models.

Part1b: Redo the experiment from Part 1a for the random waypoint mobility model with pause = 40, minSpeed = 20, and maxSpeed = 22. For this setting, again draw the node location plots and calculate the average distance from the center point. Comment on these results and how they compare to the results from Part 1a, again based on class discussions of the random waypoint model.

Turn-in instructions: In your Part 1 subfolder, include the source codes of your programs for random walk and random waypoint as described above. Also include the three location plots and average distances as specified and your discussion about each set of results as called for above. Finally, include screen shots and text files of two sample outputs, one for random walk and one for random waypoint.

Part 2: Evaluation of Wireless Link Performance vs. Distance

In this part, you will investigate the achievable data rate on a wireless link for 802.11ac technology versus the distance between sender and receiver. To do this, use the ConstantPositionMobilityModel and the SetPosition() method to set up a Wifi access point at the left boundary of your 80m x 80m square region. Your program should then similarly set up a Wifi station node at a position exactly 5 meters to the right of the access point. Use the OnOff application on the access point and the PacketSink application on the station node and calculate and output the goodput for the PacketSink application over a 2 second simulation with the OnOff application set to a rate of 400 Mbps and a packet size of 1024 bytes. You should then move the station node to the right in steps of 5 meters up to a maximum distance of 70 meters from the access point measuring the goodput in the same way at each location. (With appropriate use of the SetPosition() method, you can collect all this data in a single run of your program. However, it's acceptable if you compute one data point per run and run the program multiple times, once for each Wifi station location.) If you compute all goodputs in one program, you do not need any command-line parameters. If you compute one goodput value per program execution, you should have a command-line parameter named "stationDist" to input the distance between the Wifi station and the fixed access point. Plot the goodput vs. Wifi station distance over the distances measured.

For this part, you should use the 802.11ac standard and the Minstrel Wifi manager, which can be set up with the following lines of code:

```
WifiHelper wifi;
wifi.SetRemoteStationManager ("ns3::MinstrelHtWifiManager");
wifi.SetStandard (WIFI_STANDARD_80211ac);
```

Turn-in instructions: In your Part 2 subfolder, include the source code of your program, the goodput vs. distance plot, and a screen shot showing a command line and output for your program.