A Hybrid Wireless Network Protocol

Overview and Analysis

Computer Communications

EEL 5718

Spring 2006

26 April 2006

Sean Donovan

Casey Morrison

Michael Sandford

Wenxing Ye

Introduction

In this day and age, the use of wireless networks and mobile network interfaces are becoming widespread and expected. As the use of these mobile nodes increases, the demand for higher quality of service and reliability will also increase. Wireless local area networks (WLAN) currently employ two basic types of structures. The first structure, infrastructure WLAN, is a single-hop network that relies on base stations with a fixed connection to the wired network backbone. BS-oriented networks, as they are known, use a cellular networking scheme to connect a large number of nodes to a much smaller number of base stations.

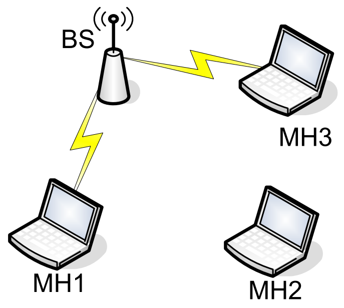


Fig. 1 - Infrastructure WLAN (BS-Oriented)

Because BS-oriented networks are connected to a wired backbone, they are very reliable and offer a high bandwidth. Another advantage of this system is the relative simplicity of the mobile nodes. The main disadvantages of this structure are the inflexibility caused by the need for an infrastructure. Infrastructure WLAN also requires complicated hand-off and human interaction to maintain the network.

The second structure is noninfrastructure WLAN, or ad hoc WLAN. In an ad hoc network, the entire network consists of mobile nodes. Ad hoc networks are not centralized; instead, quality of service is carried from node to node in a single-hop or multi-hop network. In this structure, mobile nodes depend on one another for communication.

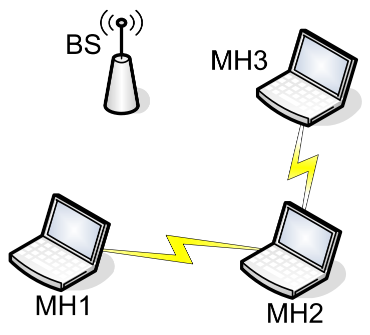
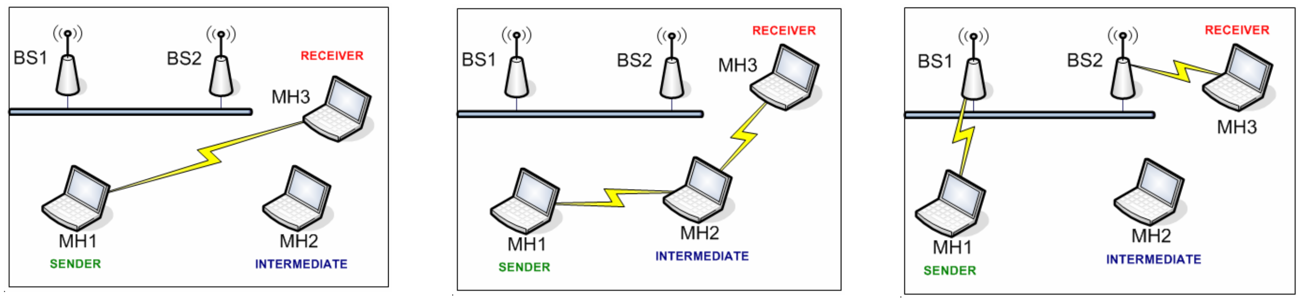


Fig. 2 - Non-Infrastructure WLAN (ad-hoc network)

Without expensive base stations and infrastructure management, ad hoc networks have a much lower cost than BS-oriented WLANs. These WLANs are much more flexible because the network capacity increases with the addition of new nodes. Unfortunately, because of this flexibility, ad hoc networks are generally unreliable.

In order to enhance the quality and reliability of networks in the future, there is a great need to create a new network structure that uses the advantages and minimizes the disadvantages of the two systems. To accomplish this dual purpose, the authors devised a “Hybrid Wireless Network Protocol,” which uses both base station networking and node-to-node networking. Depending on network reliability, this new protocol enables a node to either transmit directly to a neighbor node (one-hop), transmit through a neighbor node that is within rand of the destination node (two-hop), or use a base station to convey the message to the destination node (BS-oriented).



(a) (b) (c)

Fig. 3 – Hybrid Wireless Network Protocol for (a) One-Hop Direct Transmission, (b) Two-Hop Direct Transmission, and (c) BS-Oriented Transmission

Published Analysis

In [1], a hybrid wireless network protocol is proposed and discussed. In the authors' model, a connection can be built in BS-oriented mode, one-hop direct transmission mode, or two-hop direct transmission mode. It is important to understand when the transmission mode changes. The following figure shows the state transition diagram.

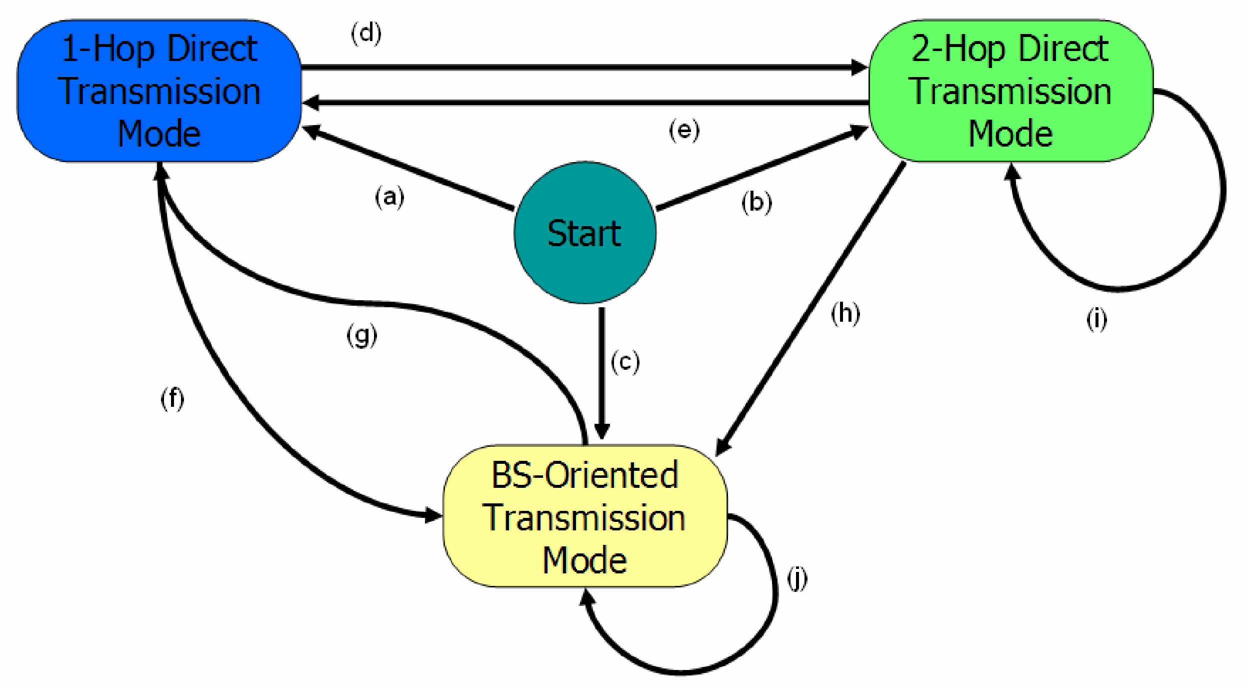

1-Hop Direct Transmission Mode
2-Hop Direct Transmission Mode
Start
BS-Oriented Transmission Mode
(a)
(b)
(c)
(d)
(e)
(f)
(g)
(h)
(i)
(j)

Fig. 4 Transition diagram for transmission mode

The transmission mode switches under the following conditions:

(a) The receiver and the sender can hear each other directly.

(b) The receiver and the sender can both communicate with an intermediate directly.

(c) Neither (a) nor (b).

(d) The receiver moves apart from the sender, so it can no longer hear from the sender. However, the sender finds one of its neighbors can communicate with the receiver directly.

(e) The receiver discovers that it can hear from the sender directly without the help of the intermediate.

(f) The receiver can no longer hear the sender, and none of its neighbors can either.

(g) The receiver discovers that it can hear from the sender directly without the help of the base station.

(h) The intermediate node is gone, and no neighbors of the sender can communicate with the receiver directly.
(i) The intermediate node is gone, but the receiver and the sender succeed in finding another intermediate to build a connection.
(j) The receiver or the sender moves from one BS's coverage to another BS's coverage.

The transition from the BS-oriented mode to two-hop direct-transmission mode is not possible because the communicating party cannot know that a third mobile host exists and is within range. According to this diagram, we can see that one-hop direct mode has the highest priority, two-hop direct mode has the second-highest, and BS-oriented mode has the lowest. In this protocol, a connection has the inclination to achieve a transmission mode with highest priority [1].

Authors' Tools and Methods

The authors of our paper chose to simulate a hybrid network modeled after a cell phone network rather than WLAN or any other wireless system. This assumption is necessary in order to make enough simplifications to build a working system. Once decided, this leads to the following ideas. There is a fixed arrival rate of calls to the system that occur on random intervals (an exponential distribution). This models the reality of the phone system where the users follow no fixed schedule. The call lengths are also functions of a random exponential. There is a fixed average call time but there could be many short calls balances by a few very long ones. This too is necessary to model reality.

The authors chose to use Simulink to simulate the hybrid network model proposed in the paper. They define an 8x8 grid of cells that make up the geometry of the simulation scenario. Due to ambiguity we were not able to ascertain the details of their model, but one image included in the paper leads us to believe that their environment includes a base station at each of the four corners of the grid with mobile nodes moving randomly anywhere within the grid.

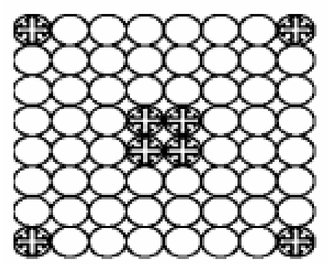


Fig. 5 - The authors' grid model

Some parameters are defined as below:

- New request arrival time: exponential (0.5)
- Service time: exponential (25.0)
- Cell resident time: exponential (12.0) (BS-oriented)
    - exponential (6.0) (one-hop direct transmission)
    - exponential (3.0) (two-hop direct transmission)
    - exponential (1.5) (three-hop direct transmission)
- *N*: number of new calls
- *Nh*: number of handoffs
- *Nb*: number of new calls blocking
- *Nf*: number of forced termination
- *Nc*: number of completion calls

The authors defined several probabilities as controls:

- $P_{one-hop}$ – The source and destination can communicate directly
- $P_{two-hop}$ – The source and destination can communicate through a neighbor
- $P_{two-hop} = 4P_{one-hop}$

These lead to performance metrics like:

- New call blocking – new calls that cannot be handled by the system
    - $Pb = \frac{Nb}{N}$
- Forced termination – calls that are forced to end due to lack of resources
    - $Pf1 = \frac{Nf}{Nh},\ Pf2 = \frac{Nf}{N}$

- Non-complete calls – the sum of blocked calls and forced terminations
  - $Pnc = \frac{Nf + Nb}{Nf + Nb + Nc}$
- Setup cost – the number of links needed to enable a given communication

Authors' Results

With these metrics the authors are then able to evaluate the performance of the hybrid network over the range of one-hop and two-hop probabilities. This enables them to make comparisons between transmission modes and therefore decide the best set of transmission modes to support.

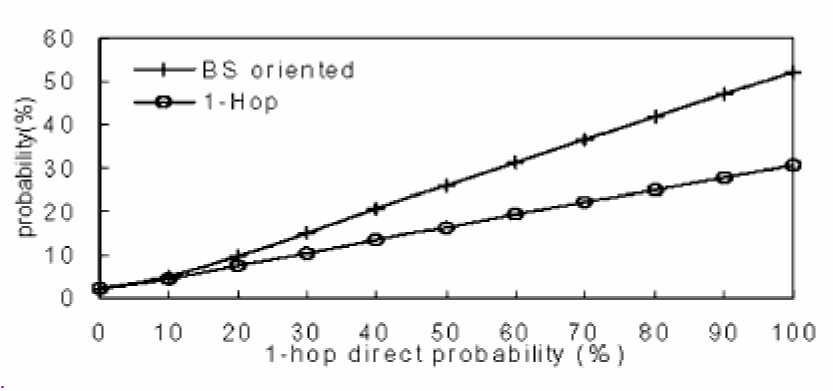


Fig. 6 – Call blocking probability as a function of one-hop probability

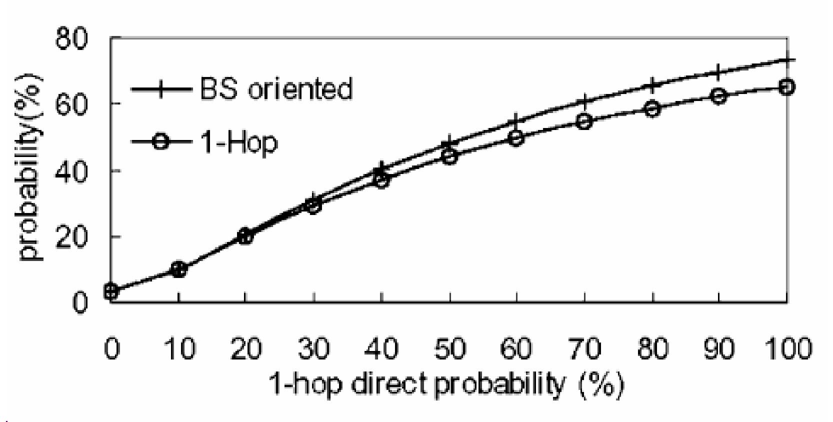


Fig. 7 – Non-completion probability as a function of one-hop probability

In the end, the authors conclude that the proposed hybrid protocol is more reliable, increases bandwidth utility, and is more fault-tolerant to BS failures. However, they discourage introducing three or more hops direct transmission into the protocol, because that will increase the protocol complexity a lot and will lead to much higher handoff cost and forced termination rate.

Performance Analysis Study

Selection of Tools

Given that we were not familiar with SIMSCRIPT and we had several weeks worth of trouble merely trying to obtain a license, we ended up deciding to use MATLAB to verify the authors' results in an independent fashion. By building a similar model and attempting to control the probabilities in some manner, we hoped to be able to come to the same conclusions that the authors did.

Simulation Model

Our model takes an approach based strongly upon reality. Our simulation is turn-based, i.e. we update the model state completely and then allow time to advance by some small amount. We define a continuous, rectangular space with equal sides of length L. Base stations are placed at each corner of the landscape and have a communication radius of length $L/2$. This provides us with a coverage ratio of $\pi/4$ or about 79% at the beginning of the simulation. A certain number of mobile nodes (chosen at run time) are instantiated at random points within the landscape. At every iteration of the simulation, each node changes its position in a quasi-random fashion. Nodes are restricted from changing their direction of motion by more than 45^o at each iteration. This rule was defined in order to avoid unrealistic Brownian motion node behavior. In addition, each node's speed (distance traveled during each iteration) is chosen randomly with some limitations.

As the simulation progresses, the model is designed to gradually restrict the node movement to force an increase in node density. This is accomplished by linearly decreasing the landscape area over time. Thus nodes are forced into a smaller area, and their one-hop probabilities are forced to increase. The motivation behind this design decision was to ensure a wide range of one-hop probabilities over the duration of the simulation. Since the authors plotted their performance metrics as a function of one-hop probability, and our model does not directly control one-hop probability, we had to invent this "shrinking landscape" concept in order to guarantee that performance metrics were gathered for all values of one-hop probability. Figure 8 below illustrates the one-hop probabilities achieved by the "shrinking landscape" over a period of 300 iterations.

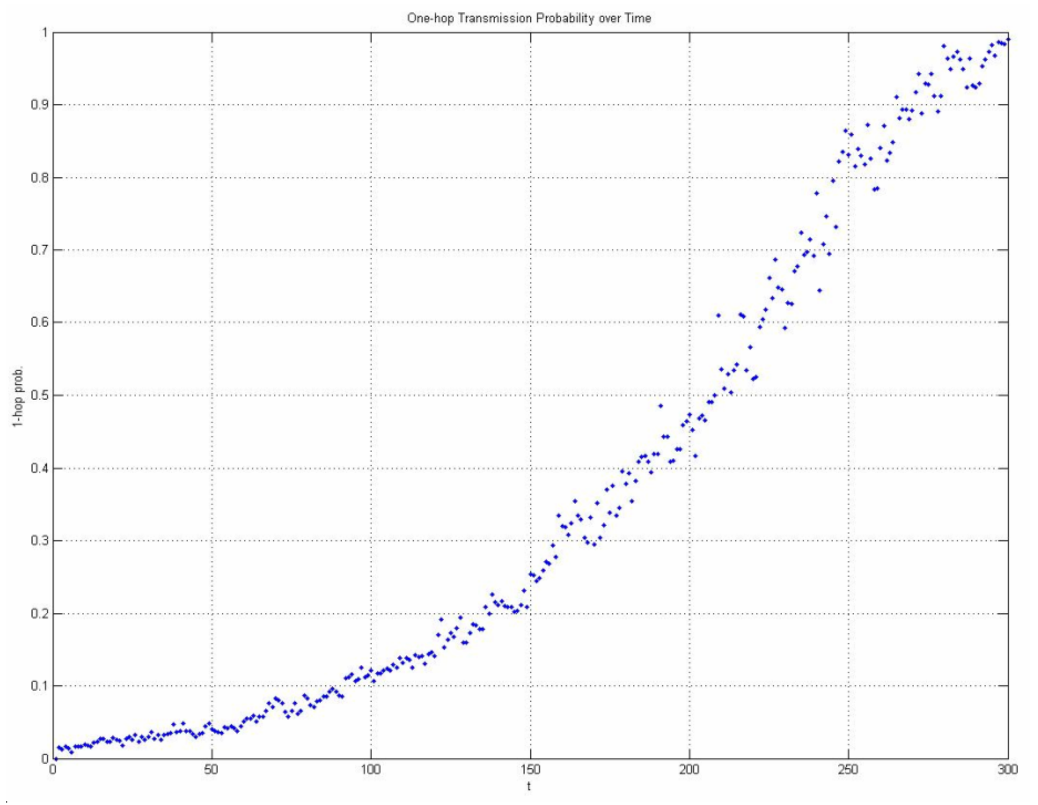


Fig. 8 – One-hop probability as a function of time

Each mobile node, which moves semi-randomly throughout the aforementioned landscape, is given a direct transmission range of $L/5$ that does not change with time. This ensures that the node density, and therefore the one-hop probability, will increase over time. We model the call arrivals and durations exactly as the authors did since it was clear enough to be understood intuitively. An example of the landscape and node movement defined by our model is shown in Figure 9.

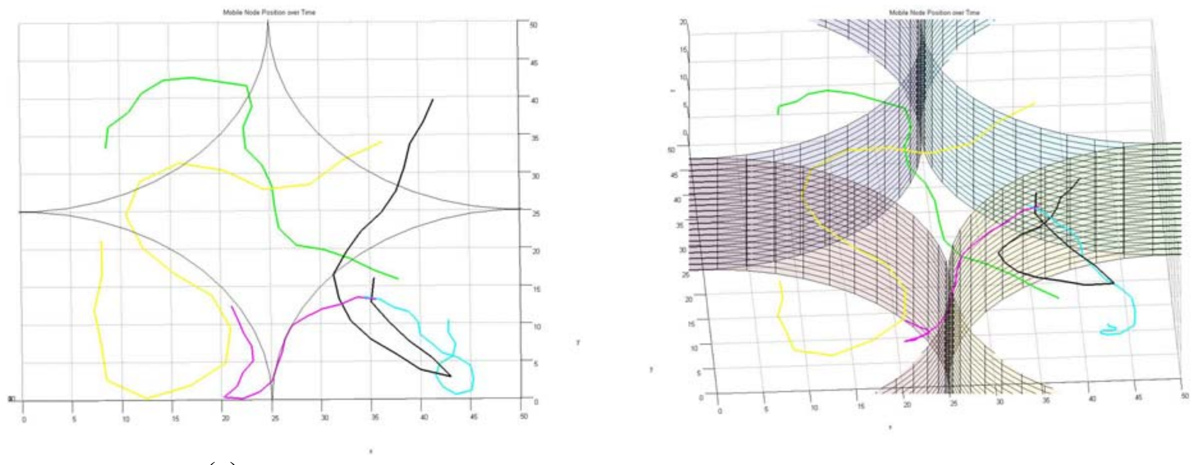

(a) (b)

Fig. 9 – Node movement visualization in (a) two dimensions (x,y) and in (b) three dimensions (x,y,t)

In order to make comparisons between the base-station-only transmission mode and one-hop-direct-transmission-enabled networks we adopted the following general program flow:

1. Generate the node movement data.
2. Generate node call arrivals and call durations.
3. Simulate once with the one-hop range set to zero (i.e. base-station-only mode).
4. Simulate again with the one-hop range set to $L/5$ (i.e. one-hop-enabled mode).
5. Plot the results on the same graph.

By setting the range of the mobile nodes to zero for one of the simulation iterations we force the nodes to make contact via the base stations, as the nodes cannot communicate directly with a range of zero.

At each time step in the simulation (which typically ran for 1000 iterations), metrics such as one-hop probability, call blocking probability, and non-complete probability were gathered. At the conclusion of the simulation, these time-varying metrics where sorted and plotted as functions of one-hop probability. It is left to speculation as to how the authors were able to directly control one-hop probability and gather performance metrics as a function of this probability. One-hop probability is a function of many things, the most obvious of which is node density. Controlling node density directly, if that is in fact what the authors did, is not a realistic way of modeling mobile nodes. While our model does change the landscape size over time, the node movement is always random; therefore, every one-hop probability encountered during the simulation is essentially by chance.

Simulation Results

After writing the simulation we began experimenting with it in order to determine parameters that modeled reality within the constraints of our simulation. Call arrival rates, call durations, and node movement rates all had to be adjusted until they passed "sanity checks." We ensured that the average load on the network was realistic by looking at the call arrival and duration data to ensure that we were not saturated from the start. Real cell phone networks are designed to operate somewhat below capacity so that handoffs can occur without dropping calls.

Once past the initial tuning necessary in order to have a model that lines up with reality, we were able to begin gathering data. We found that we could not match the authors' trends point-for-point, and in fact even some of the trends we observed were different than what they observed. But on the whole we were able to verify that by adding two extra short-range calling modes the performance of a cellular system could be improved.

Figures 10 and 11 illustrate the results that our model produced. It is important to note that while this data does not exactly match the data generated by the authors, the general trends are the same. In particular, these results show that the protocol defined by the authors performs better that the standard base-station-only protocol in terms of call blocking and call non-completion.

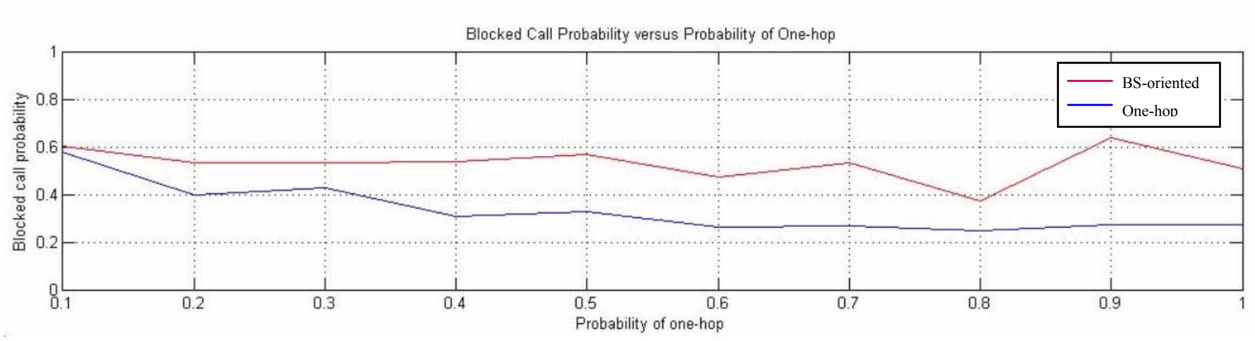


Fig. 10 – Blocked call probability as a function of one-hop probability

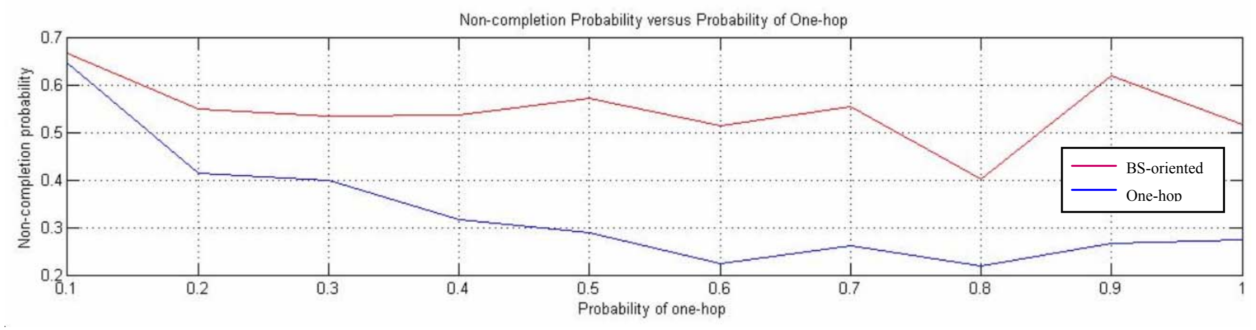


Fig. 11 – Non-completion probability as a function of one-hop probability

One noteworthy difference between our results and those of the authors is the slope of call blocking probability over time for the case of one-hop-enabled networks. Intuition suggests that as one-hop probability increases (i.e. node density increases), the probability of call blocking would decrease for nodes that are capable of one-hop transmission. If a connection is more likely to be made between source and destination, then the probability that a call will be blocked would decrease.

Our model supports this intuitive reasoning. As nodes move closer and closer together, the number (and probability) of blocked calls decreases. However, the model used by the authors produced the inverse trend. In their results, the probability of call blocking increased for both base-station-only mode as well as one-hop-enabled mode as the one-hop probability increased.

Conclusion

Our simulation of the Hybrid Wireless Protocol revealed that the hybrid protocol outperformed the BS-oriented protocol in each simulation, which agrees with the findings of the original research paper. The results of our simulation displayed the same trends as the results in the original research paper even though the actual data generated by the paper was not corroborated by our experiment. For the case of blocked call percentage and dropped call percentage, our simulation revealed that for the full range of one-hop probabilities, the hybrid protocol performed better than the BS-oriented protocol. Although our simulation results were numerically different from the original paper results, we believe it was successful because it led to the same general conclusion: using both ad-hoc and base-station modes of transmission in a wireless network provide more quality and reliability than using only base-stations.

Much of the numerical disagreement between our simulation and the original simulation comes from the simulation tools used and the ambiguity of test parameters. The original simulation employed Simulink to generate a wireless network and simulate it for the entire range of one-hop probabilities. In order to verify the original conclusion independently, we decided to use Matlab to simulate the system. We believe Matlab was a very accurate simulation tool because we were able to simulate node movement, node density, call length, call request frequency, and several other parameters to get our results. Because we had control over so many different experiment parameters, we were able to fine-tune the simulation to become more and more realistic, which was very advantageous. At the same time, access to these parameters also made our simulation deviate from the original simulation results. Because so many parameters had such an impact on the results of the simulation, it was very difficult to hypothesize which parameters were used in the original simulation. If all of the original parameters were known, our simulation would have been much closer to the original simulation.

References

[1] Chang, Ruay-Shiung, et al, "Hybrid Wireless Network Protocols," IEEE Transaction on Vehicular Technology, vol. 52, no. 4, pp. 1099 – 1109, July 2003.

Appendix A: Division of Labor

The following table lists the approximate division of labor for this project:

Table 1 – Division of labor

Name	**Contributions**
Sean Donovan	• Helped gather and interpret simulation data such that our results were as realistic as possible • Helped create the project presentation, specifically introductory background material • Helped to write the introduction and conclusion and to edit the final report
Casey Morrison	• Proposed the simulation model architecture • Helped define, implement, and fine-tune the simulation model • Helped gather and interpret simulation data • Helped create the project presentation, specifically sections dealing with the hybrid protocol description and the Matlab simulation environment • Helped write, edit, and illustrate the final report
Michael Sandford	• Proposed and helped implement simulation algorithms • Assisted in simulation data gathering and model tuning • Wrote the performance analysis section of the final report
Wenxing Ye	• Helped propose ideas and data structures in the simulation model, particularly in regards to the connection tracking capabilities of the model • Proposed alternative mathematical models, although they were not implemented • Helped create the project presentation, specifically sections dealing with the hybrid protocol description • Helped write the "published analysis" section of the final report

Appendix B: Matlab Code

main.m

```
function [BS_nBSerrors,OH_nBSerrors] = main(mobile_nodes, max_landscape_side,
number_iterations, MN_speed)

%---------------------------------------------------------------------------
% Function:    main()
% File name:   main.m
% Description: Main function that handles setup, simulation, and data
%              plotting.
%
% Notes:     - The basic technique employed in this simulation is to vary
%              the one-hop and two-hop tranmission probabilities and to
%              calculate network performance metrics based on that.
%            - Some major assumptions made are the following:
%                  * As probability of one-hop transmission increases, this
%                    necessarily means that the node density increases.
%                  * Quantity of nodes remains fixed throughout the
%                    simulation.
%                  * Multiple base stations (BSs) exist in this
%                    environment.
%---------------------------------------------------------------------------

% Global variables
global numNodes landWidth maxLandWidth minLandWidth;
global BS1 BS2 BS3 BS4 MN1 firstNode lastNode;
global BSRadius oneHopRadius twoHopRadius threeHopRadius;
global xIndex yIndex sIndex maxS numBSChannels maxT;
global posMatrix distMatrix oneHopProbVector twoHopAvailability;
global newArrivalTimes BSChannelAvailability;
global serviceTimeMU;
global callArrivalMU;
global BSOnly;
global BSAndOneHop;
global BS_nBSerrors;
global OH_nBSerrors;

% Global variables for performance tracking
global nBlockCalls;
global nCallAttempts;
global nTermination;
global nComplete;
global nOH2BSHandoffs;
global nBS2OHHandoffs;
global nBS2BSHandoffs;
global callBlockVector;
global callAttemptVector;

% Set global variables
numNodes     = mobile_nodes;
maxLandWidth = max_landscape_side;
minLandWidth = 5;
```

```
landWidth    = max_landscape_side;
maxT         = number_iterations;
maxS         = MN_speed;

% Initialize environment
initializeEnvironment(0);

% Generate simulation environment data
generateEnvironmentData(0);

% Initialize dynamic structures
initializeDynamicStructures(0);

% Simulate for BS-oriented-only environment
simulate(BSOnly);

% Re-Initialize dynamic structures
initializeDynamicStructures(0);

% Simulate for One-hop and BS-oriented environment
simulate(BSAndOneHop);

% Plot data
plotData(0);
```

initializeEnvironment.m

```
function initializeEnvironment(null)

% initializeEnvironment()
% This function will...

% Global variables
global numNodes landWidth maxLandLength maxT;
global BS1 BS2 BS3 BS4 MN1 firstNode lastNode;
global BSRadius oneHopRadius twoHopRadius threeHopRadius;
global xIndex yIndex sIndex aIndex;
global BSOnly BSAndOneHop;
global maxS;
global numBSChannels;
global posMatrix;
global distMatrix;
global oneHopProbVector;
global newArrivalTimes;
global callDurations;
global BSChannelAvailability;
global connTypeMatrix;
global connTimeRemainMatrix;
global rangeMatrix;
global callArrivalMU;
global serviceTimeMU;
global maxOneHopRadius;
global maxLandWidth;

% Initialize global variables
%callArrivalMU = 0.5;
```

```
callArrivalMU = 15;
serviceTimeMU = 10;
xIndex   = 1;
yIndex   = 2;
sIndex   = 3;
aIndex   = 4;
BSRadius = (maxLandWidth/2) - 0.001;
maxOneHopRadius = 15;

numBSChannels  = round(numNodes/3);
%numBSChannels  = numNodes;
%oneHopRadius   = (1/10)*landWidth;
%twoHopRadius   = 2*oneHopRadius;
%threeHopRadius = 3*oneHopRadius;
BSOnly      = 0;
BSAndOneHop = 1;

% Reserved nodes
BS1       = 1;
BS2       = 2;
BS3       = 3;
BS4       = 4;
firstNode = 5;
MN1       = firstNode;
lastNode  = numNodes + 4;

% Initialize node position matrix
% 1. For base stations
%
% +-----------------------+
% |BS4                 BS3|
% |                       |
% |                       |
% |                       |
% |                       |
% |                       |
% |BS1                 BS2|
% +-----------------------+

posMatrix(BS1,xIndex,1) = 0;
posMatrix(BS1,yIndex,1) = 0;
posMatrix(BS1,sIndex,1) = 0;
posMatrix(BS1,aIndex,1) = 0;

posMatrix(BS2,xIndex,1) = landWidth;
posMatrix(BS2,yIndex,1) = 0;
posMatrix(BS2,sIndex,1) = 0;
posMatrix(BS2,aIndex,1) = 0;

posMatrix(BS3,xIndex,1) = landWidth;
posMatrix(BS3,yIndex,1) = landWidth;
posMatrix(BS3,sIndex,1) = 0;
posMatrix(BS3,aIndex,1) = 0;

posMatrix(BS4,xIndex,1) = 0;
posMatrix(BS4,yIndex,1) = landWidth;
posMatrix(BS4,sIndex,1) = 0;
posMatrix(BS4,aIndex,1) = 0;

% 2. For other nodes
for n = firstNode:lastNode
    currentX = rand(1)*(landWidth-1);
    currentY = rand(1)*(landWidth-1);
    posMatrix(n,xIndex,1) = currentX;
    posMatrix(n,yIndex,1) = currentY;
    posMatrix(n,sIndex,1) = rand(1)*maxS;
    posMatrix(n,aIndex,1) = rand(1)*(2*pi);
end;

% Initialize the call duration matrix
callDurations = zeros(numNodes+4,numNodes+4,maxT);

% Initialize range matrix
rangeMatrix = zeros(numNodes+4,numNodes+4,maxT);

% Initialize call arrivals matrix
newArrivalTimes = zeros(numNodes+4,maxT);
```

generateEnvironmentData.m

```
function generateEnvironmentData(null)

% generateEnvironmentData()

% Global variables
global posMatrix;
global distMatrix;
global rangeMatrix;
global oneHopProbVector;
global newArrivalTimes;
global callDurations;
global maxT;

% For t = 1

% Update distance matrix
updateDistMatrix(1);

% Generate simulation environment data
for t=2:maxT
    % Update position matrix
    updatePosMatrix(t);

    % Update distance matrix
    updateDistMatrix(t);
end;

% Generate new call arrival times
setNewArrivalTimes(maxT);

% Generate call durations for each new call arrival
setCallDurations(maxT);
```

updateDistMatrix.m

```
function updateDistMatrix(t)

% updateDistMatrix()
% This function will...

% Identify global variables
global distMatrix firstNode lastNode;
global BS1 BS2 BS3 BS4;

for fromNode = BS1:lastNode
    for toNode = BS1:fromNode
        distance = calcDist(fromNode,toNode,t);
        distMatrix(fromNode,toNode,t) = distance;
        distMatrix(toNode,fromNode,t) = distance;
    end;
end;
```

calcDist.m

```
function dist = calcDist(fromNode, toNode, t)

% calcDist()
% This function will...

% Identify global variables
global posMatrix xIndex yIndex;

% Set variables
fromNodeX = posMatrix(fromNode,xIndex,t);
fromNodeY = posMatrix(fromNode,yIndex,t);
toNodeX   = posMatrix(toNode,xIndex,t);
toNodeY   = posMatrix(toNode,yIndex,t);

% Calculate distance
dist = sqrt((toNodeX - fromNodeX)^2 + (toNodeY - fromNodeY)^2);
```

updatePosMatrix.m

```
function updatePosMatrix(t)

% updatePosMatrix()
% This function will...

% Identify global variables
global posMatrix numNodes defaultS sIndex firstNode lastNode;
global BS1 BS2 BS3 BS4 xIndex yIndex sIndex aIndex maxT;
global landWidth maxLandWidth minLandWidth;
global BSRadius oneHopRadius;
global widthVector;

% Shrink landscape as simulation progresses to increase the
% chances of spanning all p(one-hop) values in [0,1]
%landWidth = maxLandWidth + ((minLandWidth - maxLandWidth)/(maxT - 1))*t;
```

```
%x = (1/maxT)*log(minLandWidth/maxLandWidth);
%landWidth = maxLandWidth*exp(x*t);

% Mike's
decay = 4;
landWidth = minLandWidth + (maxLandWidth - minLandWidth)*exp(-
t/(maxT/decay));

%Sean's: 1/x
%landWidth = 1/((1/minLandWidth - 1/maxLandWidth)*t/maxT + (1/maxLandWidth))
;

%landWidth = sqrt(maxLandWidth^2 + ((minLandWidth^2 -
maxLandWidth^2)/maxT)*t);
%widthVector(t) = sqrt(maxLandWidth^2 + ((minLandWidth^2 -
maxLandWidth^2)/maxT)*t);

% Update node position matrix for BSs
posMatrix(BS1,xIndex,t) = 0;
posMatrix(BS1,yIndex,t) = 0;
posMatrix(BS1,sIndex,t) = 0;
posMatrix(BS1,aIndex,t) = 0;

posMatrix(BS2,xIndex,t) = maxLandWidth;
posMatrix(BS2,yIndex,t) = 0;
posMatrix(BS2,sIndex,t) = 0;
posMatrix(BS2,aIndex,t) = 0;

posMatrix(BS3,xIndex,t) = maxLandWidth;
posMatrix(BS3,yIndex,t) = maxLandWidth;
posMatrix(BS3,sIndex,t) = 0;
posMatrix(BS3,aIndex,t) = 0;

posMatrix(BS4,xIndex,t) = 0;
posMatrix(BS4,yIndex,t) = maxLandWidth;
posMatrix(BS4,sIndex,t) = 0;
posMatrix(BS4,aIndex,t) = 0;

% Update node position matrix for MNs
for n=firstNode:lastNode
    lastS = posMatrix(n,sIndex,t-1);
    updateNodePos(n,lastS,t);

    % Optional mechanism to vary speed over time
    %newS = lastS + rand(1)*(20/maxT);
    %updateNodePos(n,newS,t);
end;
```

updateNodePos.m

```
function updateNodePos(nodeNum, speed, t)

% updateNodePos()
% This function will...

% Identify global variables
```

```
global posMatrix landWidth xIndex yIndex sIndex aIndex;

% Update X position
currentX = posMatrix(nodeNum,xIndex,t-1);
currentY = posMatrix(nodeNum,yIndex,t-1);
currentA = posMatrix(nodeNum,aIndex,t-1);

% Determine random walk in terms of polar coordinates
a = currentA + ((-pi/4) + (pi/2) * rand(1));
r = speed;

% Convert polar coordinates to Cartesian coordinates
nextX = currentX + r*cos(a);
nextY = currentY + r*sin(a);
if (nextX > landWidth || nextX < 0 || nextY > landWidth || nextY < 0)
    a = a + pi/4;
    nextX = currentX + r*cos(a);
    nextY = currentY + r*sin(a);
end;
if (nextX > landWidth || nextX < 0 || nextY > landWidth || nextY < 0)
    a = a + pi/4;
    nextX = currentX + r*cos(a);
    nextY = currentY + r*sin(a);
end;
if (nextX > landWidth || nextX < 0 || nextY > landWidth || nextY < 0)
    a = a + pi/4;
    nextX = currentX + r*cos(a);
    nextY = currentY + r*sin(a);
end;
if (nextX > landWidth || nextX < 0 || nextY > landWidth || nextY < 0)
    a = a + pi/4;
    nextX = currentX + r*cos(a);
    nextY = currentY + r*sin(a);
end;
if (nextX > landWidth || nextX < 0 || nextY > landWidth || nextY < 0)
    nextX = rand(1)*(landWidth-1);
    nextY = rand(1)*(landWidth-1);
    r = rand(1)*(landWidth/5);
end;
if (nextX > landWidth || nextX < 0 || nextY > landWidth || nextY < 0)
    %error('Unable to keep mobile nodes in bounds.');
end;

% Update position matrix
posMatrix(nodeNum,xIndex,t) = nextX;
posMatrix(nodeNum,yIndex,t) = nextY;
posMatrix(nodeNum,sIndex,t) = r;
posMatrix(nodeNum,aIndex,t) = a;
```

setNewArrivalTimes.m

```
function setNewArrivalTimes(maxT)

% setNewArrivalTimes()
% This function will...
% Identify global variables
global numNodes firstNode lastNode newArrivalTimes callArrivalMU;

% For each MN, randomly generate new arrival times according to an
% exponential distribution (with parameter 0.5). At time t, a new arrival
% is indicated by a positive (non-zero) integer representing the
% destination of the arriving call.
for n = firstNode:lastNode
    for t = 2:maxT
        arrivalTime = round(exprnd(callArrivalMU));

        % Determine who MN wants to connect to
        destination = randint(1,1,[firstNode,lastNode]);

        % Make sure destination is not itself
        while (destination == n)
            destination = randint(1,1,[firstNode,lastNode]);
        end;

        if (arrivalTime == 0)
            newArrivalTimes(n,t) = destination;
        else
            % No arrivals for arrivalTime number of slots
            for s = t:(t+arrivalTime-1)
                newArrivalTimes(n,s) = 0;
            end;

            % Now comes the arrival
            t = t + arrivalTime;
            newArrivalTimes(n,t) = destination;
        end;
    end;
end;
```

setCallDurations.m

```
function setCallDurations(maxT)

% setCallDurations()
% This function will...

% Identify global variables
global numNodes firstNode lastNode;
global newArrivalTimes;
global callDurations;
global serviceTimeMU;

% For each new call, randomly generate service time according to an
% exponential distribution (with parameter serviceTimeMU).
for n=firstNode:lastNode
    for t=2:maxT
        % Check if there is a new call arrival at this node
        if (newArrivalTimes(n,t) ~= 0)
            % Generate a random call duration (service time)
            serviceTime = round(exprnd(serviceTimeMU)) + 1;
            callDurations(n,t) = serviceTime;
        end;
    end;
end;
```

initializeDynamicStructures.m

```
function initializeDynamicStructures(null)

% initializeDynamicStructures()
% This function will...

% Global variables
global connTypeMatrix;
global connTimeRemainMatrix;
global BSChannelAvailability;
global maxT;
global numBSChannels;
global BS1 BS2 BS3 BS4;
global numNodes;

global nBlockCalls;
global nCallAttempts;
global nTermination;
global nComplete;
global nOH2BSHandoffs;
global nBS2OHHandoffs;
global nBS2BSHandoffs;
global nBSerrors;

global callBlockVector;
global callAttemptVector;

% Initialize global variables
nBlockCalls    = 0;
nCallAttempts  = 0;
nTermination   = 0;
nComplete      = 0;
nOH2BSHandoffs = 0;
nBS2OHHandoffs = 0;
nBS2BSHandoffs = 0;
nBSerrors      = 0;

callBlockVector   = zeros(maxT);
callAttemptVector = zeros(maxT);

% Initialize BS channel availability matrix
%clear BSChannelAvailability;
BSChannelAvailability = zeros(4,maxT);
BSChannelAvailability(BS1,1) = numBSChannels;
BSChannelAvailability(BS2,1) = numBSChannels;
BSChannelAvailability(BS3,1) = numBSChannels;
BSChannelAvailability(BS4,1) = numBSChannels;

% Initialize the connection matrices
connTypeMatrix = zeros(numNodes+4,numNodes+4,maxT);
connTimeRemainMatrix = zeros(numNodes+4,numNodes+4,maxT);
```

```
% Zero out range matrix
rangeMatrix = zeros(numNodes+4,numNodes+4,maxT);
```

simulate.m

```
function simulate(sim_type)

%---------------------------------------------------------------------------
% Function:    simulate()
% File name:   simulate.m
% Description:
%
% Notes:
%---------------------------------------------------------------------------

% Global variables
global maxT;
global oneHopRadius;
global maxOneHopRadius;
global BSOnly BSAndOneHop;
global rangeMatrix;

% Global variables for performance tracking (in general)
global nBlockCalls;
global nCallAttempts;
global nTermination;
global nComplete;
global nOH2BSHandoffs;
global nBS2OHHandoffs;
global nBS2BSHandoffs;
global callBlockVector;
global callAttemptVector;
global BSChannelAvailability;
global nTerminationVector;
global nCompleteVector;
global nOH2BSHandoffsVector;
global nBS2OHHandoffsVector;
global nBS2BSHandoffsVector;
global nBSerrors;

% Global variables for performance tracking of particular mode
global BS_nBlockCalls;
global BS_nCallAttempts;
global BS_nTermination;
global BS_nComplete;
global BS_nOH2BSHandoffs;
global BS_nBS2OHHandoffs;
global BS_nBS2BSHandoffs;
global BS_callBlockVector;
global BS_callAttemptVector;
global BS_BSChannelAvailability;
global BS_nTerminationVector;
global BS_nCompleteVector;
global BS_nOH2BSHandoffsVector;
global BS_nBS2OHHandoffsVector;
global BS_nBS2BSHandoffsVector;
global BS_nBSerrors;

global OH_nBlockCalls;
global OH_nCallAttempts;
global OH_nTermination;
global OH_nComplete;
global OH_nOH2BSHandoffs;
global OH_nBS2OHHandoffs;
global OH_nBS2BSHandoffs;
global OH_callBlockVector;
global OH_callAttemptVector;
global OH_BSChannelAvailability;
global OH_nTerminationVector;
global OH_nCompleteVector;
global OH_nOH2BSHandoffsVector;
global OH_nBS2OHHandoffsVector;
global OH_nBS2BSHandoffsVector;
global OH_nBSerrors;

% Set global variables
BSOnly = 0;
BSAndOneHop = 1;

% Determine simulation type
if (sim_type == BSOnly)
    % BS-oriented-only simulation
    oneHopRadius = 0;

    % Simulate
    for t=1:maxT
        updateRangeMatrix(t);
        updateConnections(t);
    end;

    % Record data
    BS_nBlockCalls       = nBlockCalls;
    BS_nCallAttempts     = nCallAttempts;
    BS_nTermination      = nTermination;
    BS_nComplete         = nComplete;
    BS_nOH2BSHandoffs    = nOH2BSHandoffs;
    BS_nBS2OHHandoffs    = nBS2OHHandoffs;
    BS_nBS2BSHandoffs    = nBS2BSHandoffs;
    BS_callBlockVector   = callBlockVector;
    BS_callAttemptVector = callAttemptVector;
    BS_BSChannelAvailability = BSChannelAvailability;
    BS_nTerminationVector = nTerminationVector;
    BS_nCompleteVector    = nCompleteVector;
    BS_nOH2BSHandoffsVector = nOH2BSHandoffsVector;
    BS_nBS2OHHandoffsVector = nBS2OHHandoffsVector;
    BS_nBS2BSHandoffsVector = nBS2BSHandoffsVector;
    BS_nBSerrors            = nBSerrors;

elseif (sim_type == BSAndOneHop)
    % BS-oriented and One-hop simulation
    oneHopRadius = maxOneHopRadius;
    % Simulate
    for t=1:maxT
        updateRangeMatrix(t);
        updateOneHopProb(t);
        updateConnections(t);
    end;

    % Record data
    OH_nBlockCalls       = nBlockCalls;
    OH_nCallAttempts     = nCallAttempts;
    OH_nTermination      = nTermination;
    OH_nComplete         = nComplete;
    OH_nOH2BSHandoffs    = nOH2BSHandoffs;
    OH_nBS2OHHandoffs    = nBS2OHHandoffs;
    OH_nBS2BSHandoffs    = nBS2BSHandoffs;
    OH_callBlockVector   = callBlockVector;
    OH_callAttemptVector = callAttemptVector;
    OH_BSChannelAvailability = BSChannelAvailability;
    OH_nTerminationVector = nTerminationVector;
    OH_nCompleteVector    = nCompleteVector;
    OH_nOH2BSHandoffsVector = nOH2BSHandoffsVector;
    OH_nBS2OHHandoffsVector = nBS2OHHandoffsVector;
    OH_nBS2BSHandoffsVector = nBS2BSHandoffsVector;
    OH_nBSerrors            = nBSerrors;

else
    % Nothing here yet
end;
```

updateRangeMatrix.m

```
function updateRangeMatrix(t)

global BSRadius;
global oneHopRadius;
global distMatrix;
global firstNode;
global lastNode;
global BS1 BS2 BS3 BS4;
global rangeMatrix;

for bs = BS1:BS4
    for mn = firstNode:lastNode
        if (distMatrix(bs,mn,t) < BSRadius)
            rangeMatrix(bs,mn,t) = 1;
            rangeMatrix(mn,bs,t) = 1;
        else
            rangeMatrix(bs,mn,t) = 0;
            rangeMatrix(mn,bs,t) = 0;
        end;
    end;
end;

for mn1 = firstNode:lastNode
    for mn2 = firstNode:lastNode
```

```
        if (distMatrix(mn1,mn2,t) < oneHopRadius);
            rangeMatrix(mn1,mn2,t) = 1;
            rangeMatrix(mn2,mn1,t) = 1;
        else
            rangeMatrix(mn1,mn2,t) = 0;
            rangeMatrix(mn2,mn1,t) = 0;
        end;
    end;
end;
```

updateConnections.m

```
function updateConnections(t)

% updateConnections()
% This function will...

% Identify global variables

if (t > 1)
    % Service existing calls
    serviceExistingCalls(t);
end;

% Examine new call arrivals
examineNewCalls(t);
```

serviceExistingCalls.m

```
function serviceExistingCalls(t)

global rangeMatrix;
global newArrivalTimes;
global firstNode;
global lastNode;
global BSChannelAvailability;
global connTypeMatrix;
global connTimeRemainMatrix;
global BSConn oneHopConn noConn;
global numBSChannels;
global nBSerrors;

%variables claiming
global nTermination;
global nTerminationVector;
global nComplete;
global nCompleteVector;
global nOH2BSHandoffs;
global nOH2BSHandoffsVector
global nBS2OHHandoffs;
global nBS2OHHandoffsVector
global nBS2BSHandoffs;
global nBS2BSHandoffsVector;

% Record number of terminations at the beginning of time t
```

```
nTerminationStart = nTermination;

% Record number of completions at the beginning of time t
nCompleteStart = nComplete;

% Record number of 1-hop-to-BS handoffs at the beginning of time t
nOH2BSHandoffsStart = nOH2BSHandoffs;

% Record number of BS-to-1-hop handoffs at the beginning of time t
nBS2OHHandoffsStart = nBS2OHHandoffs;

% Record number of BS-to-BS handoffs at the beginning of time t
nBS2BSHandoffsStart = nBS2BSHandoffs;

% Copy connection type matrix from time t-1 to time t, then change as
% needed.
connTypeMatrix(:,:,t) = connTypeMatrix(:,:,t-1);

% Copy BS channel availability matrix from time t-1 to time t, then change as
% needed.
BSChannelAvailability(:,t) = BSChannelAvailability(:,t-1);

% Remaining time -1 and tear down the connections should be closed
for x = firstNode+1:lastNode
    for y = firstNode:x-1
        if (connTimeRemainMatrix(x,y,t-1) > 1)
            % Connection was alive at last time slice.

            % Decrement connection time
            connTimeRemainMatrix(x,y,t) = connTimeRemainMatrix(x,y,t-1) - 1;
            connTimeRemainMatrix(y,x,t) = connTimeRemainMatrix(y,x,t-1) - 1;

            % Determine if connection is still alive
            if (connTypeMatrix(x,y,t) == 0)
                % No connection between x and y?! This must be wrong.
                error('connTypeMatrix wrong');

            elseif (connTypeMatrix(x,y,t) == 1)
                % This is a BS-oriented connection

                % Check to see if nodes are now in range for one-hop
                if (rangeMatrix(x,y,t) == 1)
                    % x and y can talk with each other through 1-hop, we need
handoff
                    nBS2OHHandoffs = nBS2OHHandoffs + 1;

                    % Set connection type matrix to indicate 1-hop connection
                    connTypeMatrix(x,y,t) = -1;
                    connTypeMatrix(y,x,t) = -1;

                    % Discover which BS was servicing which node
                    xBSlast = getBSinrange(x,t-1);
                    yBSlast = getBSinrange(y,t-1);

                    if (xBSlast == 0 || yBSlast == 0)
                        % Somehow x or y was not in range of a BS
                        error('Error in updating BS channel availability.
Range shows node was not in range.');
                    else
                        % Free the BS channels
                        BSChannelAvailability(xBSlast,t) =
BSChannelAvailability(xBSlast,t) + 1;
                        if (BSChannelAvailability(xBSlast,t) > numBSChannels)
                        %    error('Error in updating BS channel
availability. Number of channels exceeded maximum');
                            nBSerrors = nBSerrors + 1;
                        end;

                        BSChannelAvailability(yBSlast,t) =
BSChannelAvailability(yBSlast,t) + 1;
                        if (BSChannelAvailability(yBSlast,t) > numBSChannels)
                        %    error('Error in updating BS channel
availability. Number of channels exceeded maximum');
                            nBSerrors = nBSerrors + 1;
                        end;

                        % Tear down the connections between BSs
                        connTypeMatrix(1:4,x,t) = [0 0 0 0]';
                        connTypeMatrix(1:4,y,t) = [0 0 0 0]';
                        connTypeMatrix(x,1:4,t) = [0 0 0 0];
                        connTypeMatrix(y,1:4,t) = [0 0 0 0];
                    end;
                else
                    % x and y can not communicate through 1-hop

                    % Time permitting, check for two-hop availability

                    % Make sure nodes are still in range of their BSs
                    xBS = getBSinrange(x,t); %x is covered by xBS at t
                    yBS = getBSinrange(y,t); %x is covered by yBS at t
                    if (xBS == 0 || yBS == 0)
                        % Either x or y has stepped out of the coverage of
BS, termination occurs
                        nTermination = nTermination + 1;
                        connTypeMatrix(x,y,t) = 0;
                        connTypeMatrix(y,x,t) = 0;

                        % Tear down the connections between BSs
                        connTypeMatrix(1:4,y,t)=[0 0 0 0]';
                        connTypeMatrix(1:4,x,t)=[0 0 0 0]';
                        connTypeMatrix(x,1:4,t)=[0 0 0 0];
                        connTypeMatrix(y,1:4,t)=[0 0 0 0];

                        % Set remaining time to 0
                        connTimeRemainMatrix(x,y,t) = 0;
                        connTimeRemainMatrix(y,x,t) = 0;

                        % Free the BS channels
                        xBSlast = getBSinrange(x,t-1); %x is covered by
xBSlast at t-1
                        yBSlast = getBSinrange(y,t-1); %x is covered by
yBSlast at t-1
```

```
                        BSChannelAvailability(xBSlast,t) =
BSChannelAvailability(xBSlast,t) + 1;
                        if (BSChannelAvailability(xBSlast,t) > numBSChannels)
                        %    error('Error in updating BS channel
availability. Number of channels exceeded maximum');
                            nBSerrors = nBSerrors + 1;
                        end;

                        BSChannelAvailability(yBSlast,t) =
BSChannelAvailability(yBSlast,t) + 1;
                        if (BSChannelAvailability(yBSlast,t) > numBSChannels)
                        %    error('Error in updating BS channel
availability. Number of channels exceeded maximum');
                            nBSerrors = nBSerrors + 1;
                        end;

                    else
                        % x and y can both be covered by some BS. The
                        % connection remains unchanged or handoff between
                        % BSs occurs.

                        % Check channel availibility at new BS before handing
off
                        xBSlast = getBSinrange(x,t-1);
                        yBSlast = getBSinrange(y,t-1);

                        if ( (xBSlast ~= xBS && BSChannelAvailability(xBS,t)
== 0) || (yBSlast ~= yBS && BSChannelAvailability(yBS,t) == 0))
                            % x or y has moved to a new BS that does not
                            % have an open channel available

                            % Terminate call
                            nTermination = nTermination + 1;

                            % Tear down the existing connection
                            connTypeMatrix(1:4,x,t)=[0 0 0 0]';
                            connTypeMatrix(x,1:4,t)=[0 0 0 0];
                            connTypeMatrix(x,y,t) = 0;
                            connTypeMatrix(y,x,t) = 0;
                            connTypeMatrix(1:4,y,t)=[0 0 0 0]';
                            connTypeMatrix(y,1:4,t)=[0 0 0 0];

                            % Set remaining time to 0
                            connTimeRemainMatrix(x,y,t) = 0;
                            connTimeRemainMatrix(y,x,t) = 0;

                            % Free the BS channels
                            BSChannelAvailability(xBSlast,t) =
BSChannelAvailability(xBSlast,t) + 1;
                            if (BSChannelAvailability(xBSlast,t) >
numBSChannels)
                            %    error('Error in updating BS channel
availability. Number of channels exceeded maximum');
                                nBSerrors = nBSerrors + 1;
                            end;
                            BSChannelAvailability(yBSlast,t) =
BSChannelAvailability(yBSlast,t) + 1;
                            if (BSChannelAvailability(yBSlast,t) >
numBSChannels)
                            %    error('Error in updating BS channel
availability. Number of channels exceeded maximum');
                                nBSerrors = nBSerrors + 1;
                            end;
                        elseif (xBSlast ~= xBS && yBSlast ~= yBS && xBS ==
yBS && BSChannelAvailability(xBS,t) <= 1)
                            % x and y both left their old BS and moved into
range of a new, common BS.
                            % This new BS has less than 2 channels available,
so a connection cannot be made.
                            % Terminate call
                            nTermination = nTermination + 1;

                            % Tear down the existing connection
                            connTypeMatrix(1:4,x,t)=[0 0 0 0]';
                            connTypeMatrix(x,1:4,t)=[0 0 0 0];
                            connTypeMatrix(x,y,t) = 0;
                            connTypeMatrix(y,x,t) = 0;
                            connTypeMatrix(1:4,y,t)=[0 0 0 0]';
                            connTypeMatrix(y,1:4,t)=[0 0 0 0];

                            % Set remaining time to 0
                            connTimeRemainMatrix(x,y,t) = 0;
                            connTimeRemainMatrix(y,x,t) = 0;

                            % Free the BS channels
                            BSChannelAvailability(xBSlast,t) =
BSChannelAvailability(xBSlast,t) + 1;
                            if (BSChannelAvailability(xBSlast,t) >
numBSChannels)
                            %    error('Error in updating BS channel
availability. Number of channels exceeded maximum');
                                nBSerrors = nBSerrors + 1;
                            end;

                            BSChannelAvailability(yBSlast,t) =
BSChannelAvailability(yBSlast,t) + 1;
                            if (BSChannelAvailability(yBSlast,t) >
numBSChannels)
                            %    error('Error in updating BS channel
availability. Number of channels exceeded maximum');
                                nBSerrors = nBSerrors + 1;
                            end;
                        end;
                        if (xBSlast ~= xBS && BSChannelAvailability(xBS,t) >
0)
                            % x has moved to a new BS with an available
                            % channel. BS-to-BS handoff needed.
                            nBS2BSHandoffs = nBS2BSHandoffs + 1;

                            % Update BS channel availibility
                            BSChannelAvailability(xBSlast,t) =
BSChannelAvailability(xBSlast,t) + 1;

                            % Tear down the existing connection
                            connTypeMatrix(1:4,x,t)=[0 0 0 0]';
                            connTypeMatrix(x,1:4,t)=[0 0 0 0];

                            % Update BS channel availibility
                            BSChannelAvailability(xBS,t) =
BSChannelAvailability(xBS,t) - 1;

                            % Build the new connection between x and xBS
                            connTypeMatrix(x,xBS,t) = 1;
                            connTypeMatrix(xBS,x,t) = 1;
                        end;
                        if (yBSlast ~= yBS && BSChannelAvailability(yBS,t) >
0)
                            % y has moved to a new BS with an available
                            % channel. BS-to-BS handoff needed.
                            nBS2BSHandoffs = nBS2BSHandoffs + 1;

                            % Update BS channel availibility
                            BSChannelAvailability(yBSlast,t) =
BSChannelAvailability(yBSlast,t) + 1;

                            % Tear down the existing connection
                            connTypeMatrix(1:4,y,t)=[0 0 0 0]';
                            connTypeMatrix(y,1:4,t)=[0 0 0 0];

                            % Update BS channel availibility
                            BSChannelAvailability(yBS,t) =
BSChannelAvailability(yBS,t) - 1;

                            % Build the new connection between y and yBS
                            connTypeMatrix(y,yBS,t) = 1;
                            connTypeMatrix(yBS,y,t) = 1;
                        end;
                    end;
                end;
            else
                % x and y were connected through 1-hop at time t-1

                % Check if x and y are still in range
                if (rangeMatrix(x,y,t) == 0)
                    % x and y can no longer talk directly through 1-hop,
                    % handoff or termination needed

                    xBS = getBSinrange(x,t); % x is covered by xBS at t
                    yBS = getBSinrange(y,t); % y is covered by yBS at t
                    if (xBS == 0 || yBS == 0)
                        % Either x or y is out of range of a BS, termination
occurs
                        nTermination = nTermination + 1;
                        connTypeMatrix(x,y,t) = 0;
                        connTypeMatrix(y,x,t) = 0;
                        % Set remaining time to 0
                        connTimeRemainMatrix(x,y,t) = 0;
                        connTimeRemainMatrix(y,x,t) = 0;
                    elseif (BSChannelAvailability(xBS,t) == 0 ||
BSChannelAvailability(yBS,t) == 0)
                        % Either xBS or yBS does not have an available
                        % channel. Call must be terminated.
                        nTermination = nTermination + 1;
                        connTypeMatrix(x,y,t) = 0;
                        connTypeMatrix(y,x,t) = 0;

                        % Set remaining time to 0
                        connTimeRemainMatrix(x,y,t) = 0;
                        connTimeRemainMatrix(y,x,t) = 0;
                    else
                        % 1-hop to BS handoff occurs, we need to build the
                        % connections between x and xBS, y and yBS
                        nOH2BSHandoffs = nOH2BSHandoffs + 1;

                        % Update BS channel availability
                        BSChannelAvailability(xBS,t) =
BSChannelAvailability(xBS,t) - 1;
                        BSChannelAvailability(yBS,t) =
BSChannelAvailability(yBS,t) - 1;

                        % Update connections
                        connTypeMatrix(x,y,t)=1;   % set to be BS connection
type
                        connTypeMatrix(y,x,t)=1;   % set to be BS connection
type
                        connTypeMatrix(x,xBS,t)=1; % build connection between
x and xBS
                        connTypeMatrix(xBS,x,t)=1; % build connection between
xBS and x
                        connTypeMatrix(y,yBS,t)=1; % build connection between
y and yBS
                        connTypeMatrix(yBS,y,t)=1; % build connection between
yBS and y
                    end;
                end;
            end;
        elseif (connTimeRemainMatrix(x,y,t-1) == 1)
            % The connection between x and y ends at time t
            nComplete = nComplete + 1;

            % Update connection type
            connTypeMatrix(x,y,t) = 0;
            connTypeMatrix(y,x,t) = 0;

            % Make sure to zero out time remaining
            connTimeRemainMatrix(x,y,t) = 0;
            connTimeRemainMatrix(y,x,t) = 0;

            % Check if this was a BS-oriented connection
            if (connTypeMatrix(x,y,t-1) == 1)
                % Determine xBS and yBS
                xBSlast = getBSinrange(x,t-1); % x is covered by xBSlast at
t-1
                yBSlast = getBSinrange(y,t-1); % y is covered by yBSlast at
t-1

                % Check for error
                if (xBSlast == 0 || yBSlast == 0)
                    % Somehow, either x or y wandered out of range of its
                    % BS in the last time slice, but this routine did not
                    % process that correctly
                    error('Error processing call completion. Cannot update BS
channel availability because no BS in range.');
                else
                    % Free the BS channels
                    BSChannelAvailability(xBSlast,t) =
BSChannelAvailability(xBSlast,t) + 1;
                    if (BSChannelAvailability(xBSlast,t) > numBSChannels)
                    %    error('Error in updating BS channel availability.
Number of channels exceeded maximum');
                        nBSerrors = nBSerrors + 1;
                    end;

                    BSChannelAvailability(yBSlast,t) =
BSChannelAvailability(yBSlast,t) + 1;
                    if (BSChannelAvailability(yBSlast,t) > numBSChannels)
                    %    error('Error in updating BS channel availability.
Number of channels exceeded maximum');
                        nBSerrors = nBSerrors + 1;
                    end;

                    % Tear down the connections between BSs
                    connTypeMatrix(1:4,y,t) = [0 0 0 0]';
                    connTypeMatrix(x,1:4,t) = [0 0 0 0];
                end;
            end;
        else
            % No connection existed between this pair
        end;
    end;
end;

% Record number of terminations at the end of time t
nTerminationFinish = nTermination;

% Record number of completions at the end of time t
nCompleteFinish = nComplete;

% Record number of 1-hop-to-BS handoffs at the end of time t
nOH2BSHandoffsFinish = nOH2BSHandoffs;

% Record number of BS-to-1-hop handoffs at the end of time t
nBS2OHHandoffsFinish = nBS2OHHandoffs;

% Record number of BS-to-BS handoffs at the end of time t
nBS2BSHandoffsFinish = nBS2BSHandoffs;
```

```
% Record total number of terminations for time t
nTerminationVector(t) = nTerminationFinish - nTerminationStart;

% Record total number of completions for time t
nCompleteVector(t) = nCompleteFinish - nCompleteStart;

% Record total number of 1-hop-to-BS handoffs for time t
nOH2BSHandoffsVector(t) = nOH2BSHandoffsFinish - nOH2BSHandoffsStart;

% Record total number of BS-to-1-hop handoffs for time t
nBS2OHHandoffsVector(t) = nBS2OHHandoffsFinish - nBS2OHHandoffsStart;

% Record total number of BS-to-BS handoffs for time t
nBS2BSHandoffsVector(t) = nBS2BSHandoffsFinish - nBS2BSHandoffsStart;
```

getBSinrange.m

```
function bs = getBSinrange(node,t)

% Get the BS who can cover node.
% Return 0 if no BS convers node

global rangeMatrix;
global BS1 BS2 BS3 BS4;

bs = 0;
for i = BS1:BS4
    if (rangeMatrix(node,i,t) == 1)
        bs = i;
        break;
    end;
end;
```

examineNewCalls.m

```
function examineNewCalls(t)

% examineNewCalls()
% This function will...

% Identify global variables
global BS1 BS2 BS3 BS4 firstNode lastNode
global numNodes;
global rangeMatrix;
global newArrivalTimes;
global callDurations;
global connTypeMatrix;
global connTimeRemainMatrix;
global BSChannelAvailability;
global BSConn oneHopConn noConn;
global nBlockCalls;
global nCallAttempts;
global callBlockVector;
global callAttemptVector;
```

```
% Set global variable
BSConn = 1;
noConn = 0;
oneHopConn = -1;

% Set local variables
numBlockCalls = 0;
numCallAttempts = 0;

% Save the number of call blocks (and attempts) prior to this iteration
nBlockCallsStart   = nBlockCalls;
nCallAttemptsStart = nCallAttempts;

% Examine new call arrivals
for srcNode=firstNode:lastNode
    % Determine if node n has a new call arrival (new desire to transmit)
    destNode = newArrivalTimes(srcNode,t);

    if (destNode > 0)
        % Calculate a service time (call duration) for connection
        %DON'T GENERATE DYNAMICALLY: callDuration =
round(exprnd(serviceTimeMU);
        callDuration = callDurations(srcNode,t);

        % Call duration must be a positive (non-zero) integer
        if (callDuration <= 0)
            error('Error in referencing callDurations matrix. Durations of 0
or less are not allowed.');
        end;

        % New call arrival for MN(destination)
        % Determine if current MN is busy with another call
        if (callInProgress(srcNode,t) == 0)
            % There is no call currently in progress. Attempt to make call.
            nCallAttempts = nCallAttempts + 1;
            numCallAttempts = numCallAttempts + 1;

            % Make sure destination node is not busy
            if (callInProgress(destNode,t) == 1)
                % Destination currently has a call in progess. This new
                % arrival is blocked.
                nBlockCalls = nBlockCalls + 1;
                numBlockCalls = numBlockCalls + 1;
                %nCallAttempts = nCallAttempts - 1;

            % Determine if source and destination are in range for one-hop
            elseif (rangeMatrix(srcNode,destNode,t) == 1)
                % Source and destination are in range, and neither is busy --
> one-hop

                % Make two entries in connection type matrix:
                %   1. Indicate one-hop connection from source to dest.
                %   2. Indicate one-hop connection from dest. to source
                connTypeMatrix(srcNode,destNode,t) = oneHopConn;
                connTypeMatrix(destNode,srcNode,t) = oneHopConn;
                % Set call duration
                connTimeRemainMatrix(srcNode,destNode,t) = callDuration;
                connTimeRemainMatrix(destNode,srcNode,t) = callDuration;
            else
                % Source and destination are not in range

                % Time permitting, determine if source and destination are
                % in two-hop range.

                % Determine if source is in range of any of the BSs
                srcBS = getBSinrange(srcNode,t);

                % Determine if destination is in range of any of the BSs
                destBS = getBSinrange(destNode,t);

                if (srcBS == 0 || destBS == 0)
                    % Connection cannot be made. Increment number of blocked
calls.
                    nBlockCalls = nBlockCalls + 1;
                    numBlockCalls = numBlockCalls + 1;
                elseif (srcBS == destBS && (BSChannelAvailability(srcBS,t) <=
1))
                    % Connection cannot be made. Increment number of blocked
calls.
                    nBlockCalls = nBlockCalls + 1;
                    numBlockCalls = numBlockCalls + 1;
                elseif (BSChannelAvailability(srcBS,t) == 0 ||
BSChannelAvailability(destBS,t) == 0)
                    % Connection cannot be made. Increment number of blocked
calls.
                    nBlockCalls = nBlockCalls + 1;
                    numBlockCalls = numBlockCalls + 1;
                else
                    % Update connection type matrix
                    % NOTE: there are at most six entries in the connection
                    % type matrix associated with this connection

                    % Source node connections
                    connTypeMatrix(srcNode,destNode,t) = BSConn; % end-to-end
                    connTypeMatrix(srcNode,srcBS,t) = BSConn;    % source to
BS
                    connTypeMatrix(srcBS,srcNode,t) = BSConn;    % BS to
source

                    % Source node connections
                    connTypeMatrix(destNode,srcNode,t) = BSConn; % end-to-end
                    connTypeMatrix(destNode,destBS,t) = BSConn;  % dest to BS
                    connTypeMatrix(destBS,destNode,t) = BSConn;  % BS to dest

                    % Set call duration
                    connTimeRemainMatrix(srcNode,destNode,t) = callDuration;
                    connTimeRemainMatrix(destNode,srcNode,t) = callDuration;

                    % Update BS channel availability matrix
                    BSChannelAvailability(srcBS,t)  =
BSChannelAvailability(srcBS,t) - 1;
```

```
                    BSChannelAvailability(destBS,t) =
BSChannelAvailability(destBS,t) - 1;
                end;
            end;
        else
            % There is another call in progress. Drop this call without
            % considering it a blocked call.
        end;
    else
        % No new call arrival for this time slice
        % Existing calls serviced by serviceExistingCalls().
    end;
end;

% Record the final number of blocked calls (and attempts) up through this
iteration
nBlockCallsFinish   = nBlockCalls;
nCallAttemptsFinish = nCallAttempts;

% Calculate the number of call blocks (and attempts) DURING this iteration
%callBlockVector(t)   = nBlockCallsFinish - nBlockCallsStart;
callBlockVector(t) = numBlockCalls;
%callAttemptVector(t) = nCallAttemptsFinish - nCallAttemptsStart;
callAttemptVector(t) = numCallAttempts;
```

callInProgress.m

```
function busy = callInProgress(mn1,t)

% callInProgress()
% This function will...

% Identify global variables
global BS1 BS2 BS3 BS4 firstNode lastNode numNodes;
global connTypeMatrix;

% Initialize busy to false
busy = 0;

for mn2 = firstNode:lastNode
    if (connTypeMatrix(mn1,mn2,t) == 0)
        % No connection between mn1 and current mn2 at time t
    else
        % Connection exists between srcNode and current destNode
        busy = 1;
        break;
    end;
end;
```

plotData.m

```
function plotData(null)

%---------------------------------------------------------------------------
% Function:    plotData()
% File name:   plotData.m
% Description:
%
% Notes:
%---------------------------------------------------------------------------

% Global variables
global numNodes landWidth maxLandWidth BS1 BS2 BS3 BS4 MN1 firstNode
lastNode;
global xIndex yIndex sIndex defaultS numBSChannels maxT;
global posMatrix distMatrix oneHopProbVector twoHopAvailability;
global newArrivalTimes;
global BSChannelAvailability;
global BSRadius;

% Global variables for performance tracking
global nBlockCalls;
global nCallAttempts;
global nTermination;
global nComplete;
global nOH2BSHandoffs;
global nBS2OHHandoffs;
global nBS2BSHandoffs;
global callBlockVector;
global callAttemptVector;
global widthVector;
global maxLandWidth;

% Global variables for performance tracking of particular mode
global BS_nBlockCalls;
global BS_nCallAttempts;
global BS_nTermination;
global BS_nComplete;
global BS_nOH2BSHandoffs;
global BS_nBS2OHHandoffs;
global BS_nBS2BSHandoffs;
global BS_callBlockVector;
global BS_callAttemptVector;
global BS_BSChannelAvailability;
global BS_nTerminationVector;
global BS_nCompleteVector;
global BS_nOH2BSHandoffsVector;
global BS_nBS2OHHandoffsVector;
global BS_nBS2BSHandoffsVector;

global OH_nBlockCalls;
global OH_nCallAttempts;
global OH_nTermination;
global OH_nComplete;
global OH_nOH2BSHandoffs;
global OH_nBS2OHHandoffs;
global OH_nBS2BSHandoffs;
global OH_callBlockVector;
global OH_callAttemptVector;
global OH_BSChannelAvailability;
global OH_nTerminationVector;
global OH_nCompleteVector;
global OH_nOH2BSHandoffsVector;
global OH_nBS2OHHandoffsVector;
global OH_nBS2BSHandoffsVector;

% Process some data
for t = 1:maxT
    % Create call block, forced termination, and non-completion probabilities
    if (BS_callAttemptVector(t) == 0)
        BS_callBlockProb(t)   = 0;
        BS_forcedTermProb(t)  = 0;
    else
        BS_callBlockProb(t)   =
BS_callBlockVector(t)/BS_callAttemptVector(t);
        BS_forcedTermProb(t)  =
BS_nTerminationVector(t)/BS_callAttemptVector(t);
    end;

    if ((BS_nTerminationVector(t) + BS_callBlockVector(t) +
BS_nCompleteVector(t)) == 0)
        BS_nonCompleteProb(t) = 0;
    else
        BS_nonCompleteProb(t) = (BS_nTerminationVector(t) +
BS_callBlockVector(t))/(BS_nTerminationVector(t) + BS_callBlockVector(t) +
BS_nCompleteVector(t));
    end;

    if (OH_callAttemptVector(t) == 0)
        OH_callBlockProb(t)  = 0;
        OH_forcedTermProb(t) = 0;
    else
        OH_callBlockProb(t)  = OH_callBlockVector(t)/OH_callAttemptVector(t);
        OH_forcedTermProb(t) =
OH_nTerminationVector(t)/OH_callAttemptVector(t);
    end;

    if ((OH_nTerminationVector(t) + OH_callBlockVector(t) +
OH_nCompleteVector(t)) == 0)
        OH_nonCompleteProb(t) = 0;
    else
        OH_nonCompleteProb(t) = (OH_nTerminationVector(t) +
OH_callBlockVector(t))/(OH_nTerminationVector(t) + OH_callBlockVector(t) +
OH_nCompleteVector(t));
    end;

    % Create total BS channel data
    BS_totalBSChannels(t) = BS_BSChannelAvailability(BS1,t) +
BS_BSChannelAvailability(BS2,t) + BS_BSChannelAvailability(BS3,t) +
BS_BSChannelAvailability(BS4,t);
    OH_totalBSChannels(t) = OH_BSChannelAvailability(BS1,t) +
OH_BSChannelAvailability(BS2,t) + OH_BSChannelAvailability(BS3,t) +
OH_BSChannelAvailability(BS4,t);

    % Create total handoffs data
    BS_totalHandoffs(t) = BS_nOH2BSHandoffsVector(t) +
BS_nBS2OHHandoffsVector(t) + BS_nBS2BSHandoffsVector(t);
```

```
    OH_totalHandoffs(t) = OH_nOH2BSHandoffsVector(t) +
OH_nBS2OHHandoffsVector(t) + OH_nBS2BSHandoffsVector(t);
end;

% Data plots

t = 1:maxT;

%{
tmax = 50
t=1:tmax;
cyl_radius = BSRadius*ones(1,tmax);
[cx cy cz] = cylinder(cyl_radius,50);
C1(:,:,1) = 0.5*ones(size(cz));
C1(:,:,1) = 0.5*ones(size(cz));
C1(:,:,1) = 0.5*ones(size(cz));
C2(:,:,1) = 0.4*ones(size(cz));
C2(:,:,1) = 0.4*ones(size(cz));
C2(:,:,1) = 0.4*ones(size(cz));
C3(:,:,1) = 0.3*ones(size(cz));
C3(:,:,1) = 0.3*ones(size(cz));
C3(:,:,1) = 0.3*ones(size(cz));
C4(:,:,1) = 0.2*ones(size(cz));
C4(:,:,1) = 0.2*ones(size(cz));
C4(:,:,1) = 0.2*ones(size(cz));

subplot(1,1,1);
plot3(squeeze(posMatrix(5,xIndex,t)),squeeze(posMatrix(5,yIndex,t)),t,'m-
','LineWidth',2.0); grid on; hold on;
plot3(squeeze(posMatrix(6,xIndex,t)),squeeze(posMatrix(6,yIndex,t)),t,'y-
','LineWidth',2.0);
plot3(squeeze(posMatrix(7,xIndex,t)),squeeze(posMatrix(7,yIndex,t)),t,'c-
','LineWidth',2.0);
plot3(squeeze(posMatrix(8,xIndex,t)),squeeze(posMatrix(8,yIndex,t)),t,'g-
','LineWidth',2.0);
plot3(squeeze(posMatrix(9,xIndex,t)),squeeze(posMatrix(9,yIndex,t)),t,'k-
','LineWidth',2.0);

%{
surf(cx,cy,tmax*cz,C1,'FaceAlpha',0.1);
surf(cx+maxLandWidth,cy,tmax*cz,C2,'FaceAlpha',0.1);
surf(cx+maxLandWidth,cy+maxLandWidth,tmax*cz,C3,'FaceAlpha',0.1);
surf(cx,cy+maxLandWidth,tmax*cz,C4,'FaceAlpha',0.1);hold off;
%}
%{
surfl(cx,cy,tmax*cz);shading interp; colormap(copper);
surfl(cx+maxLandWidth,cy,tmax*cz);shading interp; colormap(summer);
surfl(cx+maxLandWidth,cy+maxLandWidth,tmax*cz); shading interp;
colormap(bone);
surfl(cx,cy+maxLandWidth,tmax*cz); shading interp; colormap(pink);hold off;
%}
xlabel('x');
ylabel('y');
zlabel('t');
title('Mobile Node Position over Time');
axis([0 maxLandWidth 0 maxLandWidth 0 tmax]);
%}

% One-hop probability versus time
%{
subplot(3,1,3);
plot(t,oneHopProbVector(t),'b.'); grid on;
xlabel('t');
ylabel('1-hop prob.');
title('One-hop Transmission Probability over Time');
%}

%{
% Blocked call probability versus one-hop probability
subplot(2,1,1);
scatter(oneHopProbVector,BS_callBlockProb,6,'r.'); grid on;
xlabel('Probability of one-hop');
ylabel('Blocked call probability');
title('Blocked Call Probability versus Probability of One-hop');

subplot(2,1,2);
scatter(oneHopProbVector,OH_callBlockProb,6,'b.'); grid on;
xlabel('Probability of one-hop');
ylabel('Blocked call probability');
title('Blocked Call Probability versus Probability of One-hop');
%}


% Blocked call probability versus one-hop probability
[OH_blockingProb, oneHopProb] = sortData(oneHopProbVector,OH_callBlockProb);
[BS_blockingProb, temp]       = sortData(oneHopProbVector,BS_callBlockProb);
subplot(3,1,1);
plot(oneHopProb,OH_blockingProb,'b-'); grid on; hold on;
plot(oneHopProb,BS_blockingProb,'r-');
xlabel('Probability of one-hop');
ylabel('Blocked call probability');
axis([.1 1 0 1]);
title('Blocked Call Probability versus Probability of One-hop');

% Forced termination probability versus one-hop probability
[OH_terminationProb, temp] = sortData(oneHopProbVector,OH_forcedTermProb);
[BS_terminationProb, temp] = sortData(oneHopProbVector,BS_forcedTermProb);
subplot(3,1,2);
plot(oneHopProb,OH_terminationProb,'b-'); grid on; hold on;
plot(oneHopProb,BS_terminationProb,'r-');
xlabel('Probability of one-hop');
ylabel('Forced termination probability');
title('Forced Termination Probability versus Probability of One-hop');

% Completion probability versus one-hop probability
[OH_nonCompProb, temp] = sortData(oneHopProbVector,OH_nonCompleteProb);
[BS_nonCompProb, temp] = sortData(oneHopProbVector,BS_nonCompleteProb);
subplot(3,1,3);
plot(oneHopProb,OH_nonCompProb,'b-'); grid on; hold on;
plot(oneHopProb,BS_nonCompProb,'r-');
xlabel('Probability of one-hop');
ylabel('Non-completion probability');
```

```
title('Non-completion Probability versus Probability of One-hop');


%{
% Blocked call probability versus one-hop probability
subplot(3,1,2);
scatter(oneHopProbVector,BS_callBlockProb,5,'r'); grid on; hold on;
scatter(oneHopProbVector,OH_callBlockProb,5,'b'); hold off;
xlabel('Probability of one-hop');
ylabel('Blocked call probability');
title('Blocked Call Probability versus Probability of One-hop');
%}
%{
% Forced termination probability versus one-hop probability
subplot(3,1,2);
scatter(oneHopProbVector,BS_forcedTermProb,5,'r'); grid on; hold on;
scatter(oneHopProbVector,OH_forcedTermProb,5,'b'); hold off;
xlabel('Probability of one-hop');
ylabel('Forced termination probability');
title('Forced Termination Probability versus Probability of One-hop');

% Non-completion probability versus one-hop probability
subplot(3,1,3);
scatter(oneHopProbVector,BS_nonCompleteProb,5,'r'); grid on; hold on;
scatter(oneHopProbVector,OH_nonCompleteProb,5,'b'); hold off;
xlabel('Probability of one-hop');
ylabel('Non-completion termination probability');
title('Non-completion Probability versus Probability of One-hop');
%}

%{
% Total handoffs versus one-hop probability
subplot(4,3,[10 12]);
scatter(oneHopProbVector,BS_totalHandoffs,5,'r'); grid on; hold on;
scatter(oneHopProbVector,OH_totalHandoffs,5,'b'); hold off;
xlabel('Probability of one-hop');
ylabel('Total # of handoffs');
title('Total Handoffs versus Probability of One-hop');
%}

%{
% Channel availability
subplot(3,2,2);
plot(BS_totalBSChannels,'r-'); grid on; hold on;
plot(OH_totalBSChannels,'b-'); hold off;
xlabel('t');
ylabel('BS channel availability');
title('BS Channel Availability');

% Channel availability (BS-oriented)
subplot(3,3,7);
plot(BS_BSChannelAvailability(1,:),'b.'); grid on; hold on;
plot(BS_BSChannelAvailability(2,:),'r.');
plot(BS_BSChannelAvailability(3,:),'g.');
plot(BS_BSChannelAvailability(4,:),'k.');
plot(BS_totalBSChannels,'m-'); hold off;
xlabel('t');
ylabel('BS channel availability');
title('(BS) BS Channel Availability');

% Channel availability (BS-oriented)
subplot(3,3,8);
plot(OH_BSChannelAvailability(1,:),'b.'); grid on; hold on;
plot(OH_BSChannelAvailability(2,:),'r.');
plot(OH_BSChannelAvailability(3,:),'g.');
plot(OH_BSChannelAvailability(4,:),'k.');
plot(OH_totalBSChannels,'m-'); hold off;
xlabel('t');
ylabel('1-hop channel availability');
title('(1-Hop) BS Channel Availability');
%}
```

sortData.m

```
function [metric_p, probBins] = sortData(prob_t,metric_t)

% sortData()
% This function will...

% Identify global variables
global maxT;

% Set local variables
min_prob  = 0;
max_prob  = 1;
prob_step = 0.1;
num_bins = ceil((max_prob-min_prob)/prob_step);
sortingTable = zeros(num_bins,4);
binIndex  = 1;
sumIndex  = 2;
termIndex = 3;
avgIndex  = 4;

% Initialize bin ranges
for n = 1:num_bins
    sortingTable(n,binIndex) = n*prob_step;
end;

% Sort data into bins
for t = 1:maxT
    for n = 1:num_bins
        lower_bound = (min_prob + (n-1)*prob_step);
        upper_bound = (min_prob + n*prob_step);
        if (prob_t(t) >= lower_bound && prob_t(t) < upper_bound)
            sortingTable(n,sumIndex)  = sortingTable(n,sumIndex)  +
metric_t(t);
            sortingTable(n,termIndex) = sortingTable(n,termIndex) + 1;
        end;
    end;
end;

% Average data
```

```
for n = 1:num_bins
    if (sortingTable(n,termIndex) == 0)
        sortingTable(n,avgIndex) = 0;
    else
        sortingTable(n,avgIndex) =
sortingTable(n,sumIndex)/sortingTable(n,termIndex);
    end;
end;

% Return data
metric_p = sortingTable(:,avgIndex);
probBins = sortingTable(:,binIndex);
```

Appendix A: Division of Labor

The following table lists the approximate division of labor for this project:

Name	Contributions
Sean Donovan	
Casey Morrison	
Michael Sandford	
Wenxing Ye	