Application of
Mathematica Based Simulation Template to
Demand Assigned MAC Described in IEEE P802.11-92/39
("The IBM MAC Proposal")

Submitted By:
Larry Van Der Jagt
Knowledge Implementations, Inc.
32 Conklin Road
Warwick, NY 10990

Voice: 914-986-3492  FAX: 914-986-6441  EMail: KIIVLDJ@attmail.com

Main Issues Addressed:
29.1 How Does IEEE 802.11 Do Simulation
9.1 MAC throughput & throughput probability

Commercial Introduction

This notebook contains code that has been developed by Knowledge Implementations, Inc. (KII) to support the IEEE 802.11 effort. The primary business of KII is providing contract engineering services. We have experience in many aspects of networking as well as significant experience in the area of process control. We have been involved with engineering work in the wireless networking field since 1987.

In order to help defer the cost of our involvement in generating this code we are requesting that individuals and organizations that obtain this code and use it pay a shareware fee of $200.00 to KII. As an incentive to do this we have encoded the source for the package RadioSupport1.m that contains the main analysis routines that this notebook requires to execute. KII will provide unencoded source for this package to anyone that pays the shareware fee.

Copyright © 1992 Knowledge Implementations, Inc. (KII)

KII MAKES NO REPRESENTATIONS, EXPRESSED OR IMPLIED, WITH RESPECT TO ELECTRONIC MATERIALS, INCLUDING WITHOUT LIMITATIONS ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE, ALL OF WHICH IS EXPRESSLY DISCLAIMED. KII DOES NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THIS ELECTRONIC MATERIAL WILL MEET YOUR REQUIREMENTS OR THAT THE OPERATION OF THE PROGRAM WILL BE UNINTERRUPTED OR ERROR FREE.

In addition to the foregoing, you should recognize that all software systems and their documentation contain errors and omissions. KII shall not be responsible under any circumstances for providing information on or corrections to errors and omissions discovered at any time in this Electronic Material, whether or not it is aware of the errors or omissions. KII does not recommend the use of this Electronic Material for applications in which errors or omissions could threaten life, injury or significant loss.

IN NO EVENT SHALL KII BE LIABLE FOR ANY LOST PROFITS, LOST BENEFITS, OR ANY CONSEQUENTIAL, INDIRECT, INCIDENTAL, SPECIAL OR PUNITIVE DAMAGES, WHETHER IN CONTRACT OR IN TORT, EVEN IF KII HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

This Electronic Material is copyrighted by KII.

In return for payment of the shareware fee KII agrees to grant the user a non-exclusive license to use this Electronic Material for personal use only. The user may not use this Electronic Material, directly or indirectly, alone or as part of a larger system, in whole or in part, by transferring it, reselling it or licensing it to others for compensation except for the actual cost of making the physical copy and/or transfer. Any copy transferred by the user to others must be complete and unaltered.
Overview

In IEEE P802.11-92/123 a detailed computational framework was established for the execution of performance simulations of Media Access Control State machines operated over Physical Layer Entities. This framework allowed the experimenter to locate stations geographically and to assign attributes to those stations. Some of the attributes that were defined were the station Location, TxPower, State and MessageProbability. A random sample from a statistical model was used to simulate the path loss between stations. The path loss matrix that results from this sampling was used to determine the signal level at each station resulting from transmissions from other stations. Each station maintained a StationTxQueue and a StationRxQueue that tracked messages that it desired to send and that it received. The filling and emptying of these queues was performed by traffic generating functions and by a state machine model of MAC operation. In document 92/123 the state machine model used was a very simple ALOHA type model. This document extends the work of 92/123 by implementing state machines for a demand assigned MAC similar to that described in Documents IEEE P802.11/92-39 and IEEE P802.11/91-74 submitted by K.S. Natarajan et al. At this time the feature described in 92/39 that allows for direct communication between mobiles during the C period has not been implemented. Also, at this time the assumption is made that the station is registered with an access point at all times. The addition of registration and direct C period communications would be possible in future work. This document also makes some improvements to the basic framework of document 92/123 that make it easier to use for a wide variety of situations and state machines.

The state machines illustrated in this document continue the process of demonstrating the usefulness of this framework in testing MAC/PHY choices and evaluating performance. In their current state they implement many of the functions described in the previous submittal, they do not implement many of the functions, such as synchronization that will be necessary for the final MAC/PHY design. It should be clear, however, from the functions that are performed that further enhancements would allow these other functions to be implemented and tested. As is the case of any set of state machines that are in the process of being designed, we fully expect that these machines are not perfect and that testing and optimization will reveal bugs both structural and accidental. One area in which some detail not included in the original submittals that has been added is in the area of the scheduling algorithm and in the choice of when within the C period a station will elect to attempt to gain a bandwidth reservation. Also, as it stands the amount of overhead associated with the various headers is probably larger than it needs to be and some modifications might be done in this area. As was the case with 92/123 we have not made any attempts at improving the code for speed. Our opinion here is that once the scheme that needs to be thoroughly evaluated is implemented that optimization for speed will be in order. In other words, this represents a snapshot of work in progress rather than completed analysis.

Initialization

The process of executing a simulation begins with the loading of the support functions that are contained in the package RadioLAN'RadioSupport1. Each of these functions will be described in the sections of the notebook that follow. The next cell loads the support packages. This package also uses the standard Mathematica packages Statistics'ContinuousDistributions and Statistics'DiscreteDistributions. These standard packages also need the standard packages "Statistics'NormalDistribution", "Statistics'DescriptiveStatistics", "Statistics'Common'DistributionsCommon", and "Statistics'InverseStatisticalFunctions". The loading of all of these packages is handled automatically by the package RadioSupport1. The next cell loads the support packages.

```mathematica
<<RadioLAN'RadioSupport1'
Off[General::spell1];
Off[General::spell];
```
In order to utilize this notebook it is necessary to initialize the fading statistics. This is accomplished using the function `SetFadingParameters`. This function takes three parameters that are used in calculating the path loss as a function of distance. The large scale fading is determined by using the first parameter and the second parameter as the mean and standard deviation of a normal distribution function and taking a random sample of this distribution to be the attenuation value used to calculate the large scale attenuation \(1/d^n\) where \(d\) is the distance at which the attenuation is desired. The value of \(1/d^n\) (expressed in dB) found as the large scale fading value is used as the mean of a second normal distribution with a standard deviation equal to the third parameter. This second distribution models the local fading characteristics. A random sample from this distribution is the value returned for the attenuation at a specific distance.

The cell that follows sets the fading parameters for this specific instance of the simulation. This cell must be executed prior to the start of the simulation. An example of the output of the fading model is provided in the following few cells. The function `GetAttenuation[d]` is the actual function that is used when a sample of the statistical model established with `SetFadingParameters` is required for a given distance.

```
SetFadingParameters[3, .1, 5];
```

This simulation allows the experimenter the flexibility of setting up the station population that will be involved in the simulation. Each station must have a `Location` (in a three dimensional space), `TxPower` (in dBm), `MessageProbability` (the mean of a Poisson Process) and `State` (initial state of the station of the MAC state machine) specified. The cell that follows defines an initialization procedure that can be used to set initial attributes for a list of station or access point names. The names used are arbitrary, however, a list of those must be placed in a list assigned to the symbol `AccessPoints` and `Mobiles` in order for the simulation to operate. Also, a station should not be named `empty` since this is the indicator of an empty transmit or receive queue. The initial location of the station is selected from a uniform distribution. The range of this distribution is set by the parameters `xmin, xmax, ymin, ymax` and `zmin, zmax`.

```
InitializeAPs[stations_]:=(State[#]=Hopping;
SynthTimer[#]=InitSynthTimer;
TAATimer[#]=0;
TAHTimer[#]=0;
TBTimer[#]=0;
TBHTimer[#]=0;
TCTimer[#]=0;
TCBTimer[#]=0;
InitBTimer[#]=0;
InitCTimer[#]=0;
NetworkID[#]=InitNetworkID;
BroadcastFlag[#]=False;
TxPower[#]=InitTxPower;
MessageProbability[#]=InitMessageProb;
StationTxQueue[#]=empty;(*In this simulation the AP never originates so this is always empty*)
StationRxQueue[#]=empty;(*This is the list of stations received from during B period*)
RegisteredMobiles[#]={empty};(*All mobile TXQueues that were successful in Cperiod*)
Location[#]=(Random[Statistics'ContinuousDistributions'UniformDistribution[xmin,xmax]],
Random[Statistics'ContinuousDistributions'UniformDistribution[ymin,ymax]],
Random[Statistics'ContinuousDistributions'UniformDistribution[zmin,zmax]])&,stations);

InitializeMobiles[stations_]:=(State[#]=WaitingForAH;
SynthTimer[#]=InitSynthTimer;
TBTimer[#]=0;
TCTimer[#]=0;
TxPower[#]=InitTxPower;
RegisteredAP[#]=AccessP;
MessageProbability[#]=InitMessageProb;
StationTxQueue[#]=empty;
StationRxQueue[#]=empty;
BHHeader[#]=disable,none;
IncomingFailure[#]=0;
IncomingSuccess[#]=0;
IncomingFlag[#]=0;
```

The `InitializeMobiles` function sets the initial attributes for each mobile station. The `Location` of each station is randomly selected from a uniform distribution. The `TxPower`, `MessageProbability`, `StationTxQueue`, `StationRxQueue`, `BHHeader`, `IncomingFailure`, `IncomingSuccess`, and `IncomingFlag` are all initialized to their default values. The station is then initialized to the `WaitingForAH` state.
The following cells initialize a number of parameters and executes the initialization function defined above and also builds the list of stations that is required by other processing steps. The list TransmitOnStates is a list of the states in which a device has its transmitter turned on and is used by the CheckForReceive routine in order to determine signal to interference ratio. Note, that the basic unit of time for the simulation is the Octettime that is taken to be eight bit times. The hop time in octets is calculated from the number of hops per second and the Octettime parameter. Messages within this simulation are considered to be of fixed length (in octets) with this length set by the symbol messagelength. The parameter MinimumCPeriod is used to assure that some bandwidth is set aside for reservation requests. The parameter reservationlength is used to determine the number of octets needed to be transmitted successfully to obtain a reservation for bandwidth (during the C period). The Header overhead parameters are the fixed number of octets of overhead that are assumed to be in each header. Headers that contain station lists (A Header and B Header) grow in size based on the number of stations that are currently being serviced.

```
Hopsperesecond=500;
Octettime=8 \times 10^{-6};
HopTime=Floor[1/(Octettime Hopspersecond)]
MinimumCPeriod=Floor[.20 Octettime]
messagelength=5;
AHeaderOverhead=2;
BHeaderOverhead=2;
CHeaderOverhead=2;
InitSynthTimer=2;
reservationlength=2; 
250
50
xmin=0;ymin=0;zmin=0;xmax=20;ymax=20;zmax=20;
InitTxPower=10;
InitNetworkID=0;
InitMessageProb=2;
SetAttributes[{SynthTimer, BTimer, TATimer, TATimer, TBTimer, TBTimer, 
TCTimer, TCTimer}, Listable];
SetAttributes[{State, TxPower, MessageProbability, StationTxQueue, 
StationRxQueue, Location}, Listable];
SetAttributes[{BBeader, NetworkID, BroadcastFlag, ReservationQueue, 
RegisteredAP}, Listable];
SetAttributes[{Failure, IncomingSuccess, IncomingFailure, IncomingFlag, 
LostAP}, Listable];
SetAttributes[{ImproperHop, Success, InitBTimer, InitCTimer}, Listable];
AccessPoints={Access1, Access2, Access3};
Mobiles={Station1, Station2, Station3, Station4, Station5};
TransmitOnStates={TransmittingAH, TransmitAPeriod, TransmittingBH, 
TransmitBPeriod, TransmitCPeriod, TransmittingCH};
Stations=Join[AccessPoints, Mobiles];
InitializeAPs[AccessPoints];
InitializeMobiles[Mobiles];
```
Once the station parameters and the fading parameters are set the function BuildAttenuationTable can be called with a concatenated list of stations and access points as a parameter and a table of attenuations between entities will be calculated. In this calculation, an assumption is made that reciprocity applies with respect to attenuation of the channel. That is, the path loss between two entities is the same regardless of which station is transmitting and which is receiving. This assumption leads to a symmetric matrix representation for the attenuation between stations. The row and column numbers represent which stations the attenuation value applies to. For instance, the attenuation between the Access1 and Access2 in this example appears in both (row one, column two) and (row two, column one). The cell that follows illustrates the process of generating an attenuation matrix. It should also be pointed out that the attenuation numbers that are generated do not take into account the reduction in antennae aperture as a function of wavelength and hence, tend to understate the probable attenuation at distance in real applications. The impact of this effect, however, will not be important until specific PHY implementations are added to this framework. The symbol Attenuation is assigned to the calculated Attenuation Table for use by other routines, specifically, by GenerateReceiveLevels, the routine that calculates the receive level at each station for transmissions originating at each other station and the overall level of receive power at a given station.

BuildAttenuationTable[Stations]

(-30.9852, 0, -27.3462, -29.1486, -25.8994, -33.8268, -34.279, -21.1272),
(-45.7394, -27.3462, 0, -30.2465, -16.0334, -13.6978, -34.2907, -35.7278),
(-16.717, -29.1486, -30.2465, 0, -35.2908, -35.9834, -27.5907, -38.1466),
(-31.6867, -25.8994, -16.0334, -35.2908, 0, -4.54351, -35.7892, -35.5751),
(-39.9276, -33.8268, -13.6978, -34.2907, -4.54351, 0, -45.7394, -16.717),
(-40.685, -34.279, -13.6978, -35.7278, -35.7892, -45.7394, 0, -30.9852),
(-29.3515, -21.1272, -35.9834, -38.1466, -35.5751, -16.717, -45.7394, 0)
Having established the parameters associated with the station configuration the next task undertaken is to begin the process of simulating a transmission environment using these stations. The function `GenerateTraffic` uses the `MessageProbability` associated with each station list in the first parameter to determine how many messages a particular station will generate during this time interval. A station name is selected using a uniform discrete distribution from the possible destinations stations (the second parameter list) that a particular station has available to it (Note: all stations other than itself in this model) for each message that is generated. These stations' names are placed in a list and are appended to the end of the list currently in existence as the `StationTxQueue` attribute of each station. This queue consists of a list of destination stations for which these messages are intended. The cells that follow illustrate the generation of the initial set of traffic. Each successive call will append traffic to the end of the `StationTxQueue` for each station. When operating in conjunction with state machines that take messages from the beginning of `StationTxQueue`, the function `GenerateTraffic` implements a FIFO style queue for the desired transmissions.

```
GenerateTraffic[Mobiles,Mobiles];
StationTxQueue[Mobiles] =
{{Station3, empty}, {empty}, {empty}, {empty}, {Station2, empty}}
```

The function that evaluates the instantaneous receive level at a given station is `GenerateReceiveLevels`. This function monitors the state of all stations each time it is executed and calculates the receive power of all stations that are in any state listed in the global list `TransmitOnStates` state at all other stations. The level of receive power at a particular station resulting from the transmission of another station is stored in the matrix `ArrivingTxPowerTable` by `GenerateReceiveLevels`. In order to avoid negative infinity problems a minimum receive power level is established for all stations. This level is set by setting the symbol `MinPower`. Also, in order to facilitate evaluation of how this set of stations will operate in the presence of external noise sources the table `ExternalNoiseTable` of external noise values at individual stations is required. This table is an array with one entry for the external noise level at each station (note: noise in this array is expressed as power in Watts, not dBm). The following cell sets the required parameters. The parameters `CaptureMargin` and `retryLimit` are used later in the notebook and are only initialized here for convenience. They are not used by `GenerateReceiveLevels`.

```
MinPower=-100;
ExternalNoiseTable=10^(({-100,-100,-100,-100,-100})/10) .001;
CaptureMargin=12;
retryLimit=3;
```

The support functions used in the execution of the state machines have been modified to allow for more flexible operation. In particular the function `CheckForReceive` has become a general purpose function for evaluating signal to interference ratio and the job of determining how to adjust transmit and receive queues for acceptable and unacceptable S/I ratios has been relegated to being performed in the individual state machines. This change has allowed for the elimination of the routine `CheckForResponse`. The `CheckForReceive` function now accepts three parameters, source, destination and a complete list of entities. This function returns `OK` if current conditions would result in a signal to interference ratio in excess of the parameter `CaptureMargin` for signals arriving at the destination originating from the source. It begins by calling `GenerateReceiveLevels` to update the `ArrivingTxPowerTable`. It then evaluates the sum of the arriving power from all interferers and determines if it is low enough for successful transmission to take place. If it is it returns `OK` if not it returns `NOK`. The following cell illustrates the use of this function.

```
State[Station1] ^= TransmitPeriod;
CheckForReceive[Station1, Access1, Stations]
```
Evaluating TX Conditions between Station1 and Access1

Evaluating TX Conditions between Station1 and Access1

\[
\begin{array}{cccccccc}
1\times10^{-13} & 1\times10^{-13} & 1\times10^{-13} & 1\times10^{-13} & 1\times10^{-13} & 1\times10^{-13} & 1\times10^{-13} & 1\times10^{-13} \\
1\times10^{-13} & 1\times10^{-13} & 1\times10^{-13} & 1\times10^{-13} & 1\times10^{-13} & 1\times10^{-13} & 1\times10^{-13} & 1\times10^{-13} \\
1\times10^{-13} & 1\times10^{-13} & 1\times10^{-13} & 1\times10^{-13} & 1\times10^{-13} & 1\times10^{-13} & 1\times10^{-13} & 1\times10^{-13} \\
9.8\times10^{-6} & 0.000041 & 4.3\times10^{-6} & 1.10^{-13} & 3.7\times10^{-7} & 3.5\times10^{-7} & 8.9\times10^{-7} & 1.6\times10^{-6} \\
1\times10^{-13} & 1\times10^{-13} & 1\times10^{-13} & 1\times10^{-13} & 1\times10^{-13} & 1\times10^{-13} & 1\times10^{-13} & 1\times10^{-13} \\
1\times10^{-13} & 1\times10^{-13} & 1\times10^{-13} & 1\times10^{-13} & 1\times10^{-13} & 1\times10^{-13} & 1\times10^{-13} & 1\times10^{-13} \\
1\times10^{-13} & 1\times10^{-13} & 1\times10^{-13} & 1\times10^{-13} & 1\times10^{-13} & 1\times10^{-13} & 1\times10^{-13} & 1\times10^{-13} \\
1\times10^{-13} & 1\times10^{-13} & 1\times10^{-13} & 1\times10^{-13} & 1\times10^{-13} & 1\times10^{-13} & 1\times10^{-13} & 1\times10^{-13} \\
\end{array}
\]

The MAC/PHY State Machines Used For This Simulation

Having established the support functions required, the functions that actually process the station state machines can now be presented. For the purposes of this simulation the access point is modeled as a simple state machine in which progress from one state to another is based solely on elapsed time. In general, this machine is very simple with most of the transitions being purely timer based. The complexity of the operation has been offloaded to the scheduler function and to the mobiles since within this simulation the mobiles generate all traffic and are responsible for determining whether message transmission is successful or not.

The state machine for the access point in this simulation assumes that any message received during the preceding B period will be retransmitted during the current A period. Upon exit from the state Bopping, the length of the A period and the AH header is set based on the length of the StationRxQueue resulting from the previous B period activity. The state Schedule executes the scheduler and transfers the StationRxQueue to the StationTxQueue. It also resets the receive queue.

The function `ExecuteAccessPoint` that is defined in the next cell implements this state machine for the list of access points that is provided as a parameter.

```plaintext
ExecuteAccessPoint[accesspoints_] := Do[currentaccess = accesspoints[[n]];
  Switch[State[currentaccess],
    Bopping,
      Print[currentaccess, " is Bopping"];
    Which[SynthTimer[currentaccess] == 0,
      (State[currentaccess] = Schedule;
       TATimer[currentaccess] = Length[StationRxQueue[currentaccess]] + 1 + AHHeaderOverhead;
       TATimer[currentaccess] = (messageLength (Length[StationRxQueue[currentaccess]] - 1)) + 1;
       True,
       SynthTimer[currentaccess] = SynthTimer[currentaccess] - 1;
      ),
    Schedule,
      Print[currentaccess, " is Scheduling"];
      Print["TXQueues", StationTxQueue[Stations]];
      Print["RXQueues", StationRxQueue[Stations]];
      Print["ReservationQueues", ReservationQueue[AccessPoints]];
      Scheduler[currentaccess, Mobiles];
      State[currentaccess] = TransmittingAH;
      StationTxQueue[currentaccess] = StationRxQueue[currentaccess];
      StationRxQueue[currentaccess] = {empty};
      Print["Allocations", BHeader[Mobiles]];
      Print["TXQueues", StationTxQueue[Stations]];
      Print["RXQueues", StationRxQueue[Stations]];]
```
The overall operation of the simulation is based on the concept that all traffic originates from the mobiles. A mobile's StationTXQueue is a list of all the traffic that the mobile needs to transmit. The process of making a bandwidth reservation involves the mobile verifying that, during the portion of the C period in which it elects to transmit, the S/I ratio at the controlling Access Point is sufficient to allow the transfer to be successful. If the S/I ratio is sufficient the StationTXQueue with the station's own name appended as the first element is appended to the ReservationQueue of the Access Point. The Scheduler function is executed by the ExecuteAccessPoint state machine at the start of the A period. The scheduler requires that some parameters be set prior to execution. These are the HopTime and the MinimumCPeriod and the header overhead parameters. These parameters along with the length of the A period (indicated by the length of the StationRXQueue at the start of the cycle) are used to determine how much bandwidth is available for the B period. The Scheduler generates the BHHeader that is used to set the TBTimers of the individual mobiles, the length of an individual message as detailed in the messageLength parameter enters into this calculation. Each mobile has its own BHHeader that consists of a list of 2 items. These items are the time within the B period to start transmitting and the length of time to transmit.
Scheduler also sets the value of the symbol allocated to the total length of the B period allocated. The Scheduler function is defined in the following cell.

```math
Scheduler[accesspoints_, mobiles_] := Do[(Print["Scheduler running"]; 
currentaccess = accesspoints[[n]]; 
currentqueue = ReservationQueue[currentaccess]; 
totalBPeriod = HopTime - (messageLength 
  (Length[StationRxQueue[currentaccess]] - 1)) - AHeaderOverhead - 
BHeaderOverhead - CHeaderOverhead - MinimumCPeriod; 
totalrequested = messageLength 
  (Length[Flatten[currentqueue]] - Length[currentqueue]); 
numberrequested = Length[currentqueue] - 1; 
Map[(BHHeader[#] = {disable, none}) &, mobiles]; 

(* If the total requested bandwidth is less than the total available B period time allocate all of the requests *)
Which[totalrequested <= totalBPeriod, 
(allocated = 0; 
Do[BHHeader[currentqueue[[k, 1]]] = {allocated + 1, messageLength 
  (Length[currentqueue[[k]]] - 1)]; 
allocated = allocated + messageLength 
  (Length[currentqueue[[k]]] - 1), 
{k, 1, numberrequested}]; 
ReservationQueue[currentaccess] = {empty}; 
InitBTimer[currentaccess] = allocated + 1),

(* Allocate all available B period bandwidth among stations when there isn't enough *)
True, 
(allocated = 0; mostper = Floor[totalrequested/(messageLength 
  numberrequested)]; 
Do[Which[(messageLength 
  (Length[currentqueue[[k]]] - 1)) <= mostper, 
(BHHeader[currentqueue[[k, 1]]] = {allocated + 1, messageLength 
  (Length[currentqueue[[k]]] - 1)]; 
allocated = allocated + messageLength 
  (Length[currentqueue[[k]]] - 1), 
ReservationQueue[currentaccess] = ReplacePart[ReservationQueue[currentaccess], {empty}, k]; 
If[k < numberrequested, 
totalrequested = totalrequested - messageLength 
  (Length[currentqueue[[k]]] - 1); 
mostper = Floor[totalrequested/(messageLength 
  (numberrequested - k))], True]], 
{k, 1, numberrequested}]; 
InitBTimer[currentaccess] = allocated + 1)];
```

L. Van Der Jagt
The state machines for the mobile units are considerably more complex owing to the assumption that has been made that all traffic will emanate from the mobiles. This is fundamentally an assumption that we are dealing with interfering Basic Service Areas in this simulation and that there is no Distribution System.

For the purposes of this simulation, it is assumed that a method for achieving synchronization of hopping patterns is available. The implementation of a method for achieving this synchronization within this framework is a subject for future work. In this case a mobile waits in the WaitingForAH state upon becoming active before it starts its hop timing and other functions. When the controllingAP goes into the TransmittingAH state the hop timer is initialized and the hop is begun.

The length of the StationRxQueue at the access point at the end of the B period is the length of the next A period. During the A period messages received during the previous B period are transmitted to the destination mobiles. This is accomplished by mobiles with names in the StationTxQueue of the Access point evaluating the S/I conditions for every time period in the A period and if they are acceptable increasing the IncomingSuccess counter by the number of times its name appears in the StationTxQueue. If the S/I conditions are inadequate the IncomingFailure counter is incremented. In this simulation, the service modeled is connectionless.

A mobile requests enough bandwidth to transmit its entire StationTxQueue, it drops items from this queue when it is allocated bandwidth in the B period. When the StationTxQueue is empty at the start of the C period the traffic generator is run to get new traffic.

During the B period when a station's TBTimer times out it evaluates whether the S/I ratio between the mobile and the access point is OK during the entire time that it is transmitting its allocated messages. If it is it puts its StationTxQueue the portion of its StationTxQueue that it has been authorized to transmit onto the StationRxQueue of the access point and increments the Success counter by the number of messages it was authorized to transmit. If the S/I ratio is NOT at any time during the transmission nothing is place on the StationRxQueue of the access point. In both the success and failure case the items are removed from the transmit queue once transmitted. The Failure counter is incremented if the transmission is not successful.

If the station needs to request bandwidth during the C period it selects a slot in the C period at random (from a uniform distribution of the time remaining in the C period) and during that period it evaluates whether S/I conditions to the access point are acceptable. If they are it adds to the ReservationQueue as detailed in the Scheduler discussion.
The function `ExecuteMobile` that is defined in the next cell implements this state machine for the list of mobiles that is provided as a parameter.

```plaintext
ExecuteMobile[mobiles_] := Do[currentMobile = mobiles[[n]]; 
   controllingAP = RegisteredAP[mobiles[[n]]]; 
   currentNumber = Position[Stations, currentMobile][[1, 1]]; 

   Switch[State[currentMobile], 
   Hopping, 
   (Print[currentMobile, " is Hopping"]; 
   Which[SynthTimer[currentMobile] == 0, 
     State[currentMobile] = WaitingForAH, 
     True, SynthTimer[currentMobile] ^= SynthTimer[currentMobile] - 1]), 

   WaitingForAH, 
   (Print[currentMobile, " is WaitingForAH"]; 
   HTimer[currentMobile] ^= HTimer[currentMobile] - 1; 
   (* If Hop Timer is expired Hop no matter what*) 
   Which[HTimer[currentMobile] == 0, 
     State[currentMobile] = Bopping; 
     ImproperHop[currentMobile] ^= ImproperHop[currentMobile] + 1; 
     SynthTimer[currentMobile] ^= InitSynthTimer;], 

   ReceivingAH, 
   (Print[currentMobile, " is ReceivingAH"]; 
   (*Decrement Hop Timer*) 
   HTimer[currentMobile] ^= HTimer[currentMobile] - 1; 
   (* If its not time to Hop check for AP transmitting AH*) 
   State[controllingAP] ^= TransmittingAH, 
   (* If AP is transmitting AH check S/I and if it is NOK jump to wait for hop and increment the lost access point counter*) 
   Which[CheckForReceive[controllingAP, currentMobile, Stations] == NOK, 
     (State[currentMobile] = WaitingForHop; 
     LostAP[currentMobile] ^= LostAP[currentMobile] + 1)], 

   (*If everything is OK check to see if AP has completed AH period*) 
   State[controllingAP] ^= TransmitAHPeriod, 
   (*If it is done jump to ReceiveAHPeriod State*) 
   (State[currentMobile] ^= ReceiveAHPeriod; 
   (*If we are expecting a message increment IncomingAttempts and set incoming flag to true*) 
   IncomingFlag[currentMobile] ^= Count[ 
     Flatten[StationTxQueue[controllingAP], currentMobile]]), 

   ReceiveAHPeriod, 
   (Print[currentMobile, " is ReceiveAHPeriod"]; 
   (*Decrement Hop Timer*) 
   HTimer[currentMobile] ^= HTimer[currentMobile] - 1; 
```

L. Van Der Jagt
(*If Hop Timer is expired Hop no matter what*)

Which[HTimer[currentmobile]==0,  
(State[currentmobile]="Hopping;  
ImproperHop[currentmobile]="ImproperHop[currentmobile]+1;  
SynthTimer[currentmobile]="InitSynthTimer;),  
(*If its not time to Hop check for AP transmitting A*)

State[controllingAP]==TransmitAPeriod,  
(*If AP is transmitting AH check S/I and if it is NOK with an  
incoming message expected and jump to wait for hop, also increment  
the lost access point counter and incoming failure counter*)

Which[CheckForReceive[controllingAP,currentmobile,Stations]==NOK  
&IncomingFlag[currentmobile]==0,  
(State[currentmobile]="WaitingForHop;  
IncomingFailure[currentmobile]="IncomingFailure[currentmobile]  
+IncomingFlag[currentmobile];  
IncomingFlag[currentmobile]==0;  
LostAP[currentmobile]==LostAP[currentmobile]+1)],

(*If everything is OK check to see if AP has completed A period*)

State[controllingAP]==TransmittingBB,  
(*If it is done jump to ReceiveAPeriod State*)

(State[currentmobile]="ReceivingBB;  
(*If we are expecting a message increment IncomingSuccess and set incoming flag to false*)

Which[IncomingFlag[currentmobile]==0,  
(IncomingSuccess[currentmobile]==IncomingSuccess[currentmobile]  
+IncomingFlag[currentmobile];  
IncomingFlag[currentmobile]==0))},

ReceivingBB,  
(Print[currentmobile," is ReceivingBB"];  
(*Decrement Hop Timer*)

HTimer[currentmobile]==HTimer[currentmobile]-1;  
(*If Hop Timer is expired Hop no matter what*)

Which[HTimer[currentmobile]==0,  
(State[currentmobile]="Hopping;  
ImproperHop[currentmobile]="ImproperHop[currentmobile]+1;  
SynthTimer[currentmobile]="InitSynthTimer;),  
(*If its not time to Hop check for AP transmitting BB*)

State[controllingAP]==TransmittingBB,  
(*If AP is transmitting BB check S/I and if it is NOK jump to wait for hop and increment the lost access point counter*)

Which[CheckForReceive[controllingAP,currentmobile,Stations]==NOK,  
(State[currentmobile]="WaitingForHop;  
LostAP[currentmobile]==LostAP[currentmobile]+1)],

(*If everything is OK check to see if AP has completed BB period*)

State[controllingAP]==ReceiveBPeriod,  
(*If it is done jump to ReceiveBPeriod State*)
If[Number[BHHeader[currentmobile][1]],
State[currentmobile]^=WaitBPeriod;
HTimer[currentmobile]^=HTimer[currentmobile]-1;
TBTimer[currentmobile]^=TBTimer[currentmobile]-1;
(*If Hop Timer is expired Hop no matter what*)
Which[HTimer[currentmobile]==0,
(State[currentmobile]^=Hopping;
ImproperHop[currentmobile]^=ImproperHop[currentmobile]+1;
SynthTimer[currentmobile]^=InitSynthTimer;),
(*If it's not time to Hop check for time to transmit*)
TBTimer[currentmobile]==0,
(*If it is done jump to TransmitBPeriod State*)
(State[currentmobile]^=TransmitBPeriod;
TBTimer[currentmobile]^=BHHeader[currentmobile][2])],

TransmitBPeriod,
(Print[currentmobile," is TransmitBPeriod"];
(*Decrement Hop Timer*)
HTimer[currentmobile]^=HTimer[currentmobile]-1;
TBTimer[currentmobile]^=TBTimer[currentmobile]-1;
(*If Hop Timer is expired Hop no matter what*)
Which[HTimer[currentmobile]==0,
(State[currentmobile]^=Hopping;
ImproperHop[currentmobile]^=ImproperHop[currentmobile]+1;
SynthTimer[currentmobile]^=InitSynthTimer;),
(*If it's not time to Hop check for expiration of transmit timer and
everything NOK*)
TBTimer[currentmobile]!=0&&
CheckForReceive[currentmobile,controllingAP,Stations]==NOK,
Failure[currentmobile]^=Failure[currentmobile]+Floor[BHHeader[currentmobile][2]/message_length];
StationTxQueue[currentmobile]^=Drop[StationTxQueue[currentmobile],
Floor[BHHeader[currentmobile][2]/message_length];
State[currentmobile]^=WaitForCB,
(*If it is expired and everything is ok increment success and
continue*)
TBTimer[currentmobile]==0,
Success[currentmobile]^=Success[currentmobile]+
Floor[BHHeader[currentmobile][2]/message_length];
StationRxQueue[controllingAP]^=AppendTo[
    StationRxQueue[controllingAP],
    Take[StationTxQueue[currentmobile],
    Floor[BHHeader[currentmobile][2]/message_length]];]
WaitingForCB,

(Print[currentmobile," is WaitingForCB");

(*Decrement Hop Timer and waiting to transmit timer*)

HTimer[currentmobile]=HTimer[currentmobile]-1;

(*If Hop Timer is expired Hop no matter what*)

Which[HTimer[currentmobile]==0,
(State[currentmobile]=Hopping;
ImproperHop[currentmobile]=ImproperHop[currentmobile]+1;
SynthTimer[currentmobile]=InitSynthTimer;),

(*If its' not time to Hop check for AP transmits CH & OK*)

State[controllingAP]=TransmittingCH,

(*If AP is transmitting AH check S/I and if it is NOK jump to wait for hop and increment the lost access point counter*)

Which[CheckForReceive[controllingAP,currentmobile,Stations]==NOK,
(State[currentmobile]=WaitingForHop;
LostAP[currentmobile]=LostAP[currentmobile]+1),

(*If everything is OK check to see if AP has completed CH period*)

State[controllingAP]=ReceiveCPeriod,
State[currentmobile]=SetupCPeriod],

SetupCPeriod,

(Print[currentmobile," is SetupCPeriod");

(*Decrement Hop Timer and waiting to transmit timer*)

HTimer[currentmobile]=HTimer[currentmobile]-1;

(*If Hop Timer is expired Hop no matter what*)

Which[HTimer[currentmobile]==0,
(State[currentmobile]=Hopping;
ImproperHop[currentmobile]=ImproperHop[currentmobile]+1;
SynthTimer[currentmobile]=InitSynthTimer;),

(*If the TXQueue goes empty generate traffic and request bandwidth*)

StationTxQueue[currentmobile]=={empty},
(GenerateTraffic[={currentmobile},mobiles];
Print["Traffic Generator Executed"];)
Which[StationTxQueue[currentmobile]=={empty},
State[currentmobile]=WaitingForHop,

(*There is new traffic a reservation is required, pick a slot*)

True,
TCTimer[currentmobile]=Random[
Statistics::DiscreteDistributions::DiscreteUniformDistribution[
(Floor[HTimer[currentmobile]/messagelength]-1) messagelength]];)
Print["Traffic Generator Executed"];,
State[currentmobile]=WaitCPeriod),
True,
State[currentmobile]=WaitingForHop],

WaitCPeriod,

( (*Print[currentmobile," is InactiveCPeriod"];*))

(*Decrement Hop Timer and waiting to transmit timer*)
HTimer[currentmobile] = HTimer[currentmobile] - 1;
TCTimer[currentmobile] = TCTimer[currentmobile] - 1;

(*If Hop Timer is expired Hop no matter what*)
Which[HTimer[currentmobile]] == 0,
(Sta[currenltmobile] = Bopping;
ImproperHop[currentmobile] = ImproperHop[currentmobile] + 1;
SynthTimer[currentmobile] = InitSynthTimer;),

(*If it's not time to Hop check for time to transmit*)
TCTimer[currentmobile] == 0,
(*If it is done jump to TransmitCPeriod State*)
(Staqe[currentmobile] = TransmitCPeriod;
TCTimer[currentmobile] = reservationlength)),

TransmitCPeriod,

(*Print[currentmobile," is TransmitCPeriod"]*)
(*Decrement Hop Timer and waiting to transmit timer*)
HTimer[currentmobile] = HTimer[currentmobile] - 1;
TCTimer[currentmobile] = TCTimer[currentmobile] - 1;

(*If Hop Timer is expired Hop no matter what*)
Which[HTimer[currentmobile]] == 0,
(Sta[currenltmobile] = Bopping;
ImproperHop[currentmobile] = ImproperHop[currentmobile] + 1;
SynthTimer[currentmobile] = InitSynthTimer;),

(*If TC Timer expires try to make reservation*)
TCTimer[currentmobile] == 0, Which[
CheckForReceive[currentmobile, controllingAP, Stations] == NOK,
(Failure[currentmobile] = Failure[currentmobile] +
Length[StationTxQueue[currentmobile]];
StationTxQueue[currentmobile] = empty;
State[currentmobile] = WaitingForHop)],

(*If everything is ok increment success and continue*)
TCTimer[currentmobile] == 0,
(ReservationQueue[controllingAP] =
PrependTo[ReservationQueue[controllingAP],
Prepend[Drop[StationTxQueue[currentmobile], -1], currentmobile]];
State[currentmobile] = WaitingForHop)),

WaitingForHop,

(*Print[currentmobile," is WaitingForHop"]*)
Which[HTimer[currentmobile]] == 0,
(Sta[currenltmobile] = Bopping;
SynthTimer[currentmobile] = InitSynthTimer;),
True, HTimer[currentmobile] = HTimer[currentmobile] - 1]),

{n, 1, Length[mobiles]}

Simulation Execution and Results
Finally, the stage is set for performing an actual simulation. The following cell executes 1000 cycles of the defined simulation and provides diagnostic information as it is executing as well as summary statistics upon completion.

```math
maxtime = 1000;
Do[ExecuteAccessPoint[{Access1}]; ExecuteMobile[Mobiles], {tt, 1, maxtime}]
```

Output of this command detailed in Appendix

```
Success[Mobiles]  
Failure[Mobiles]   
IncomingSuccess[Mobiles]  
IncomingFailure[Mobiles]  
LostAP[Mobiles]  
{7, 5, 7, 10, .8}  
{0, 2, 0, 2, 0}  
{3, 12, 5, 2, 4}  
{0, 0, 0, 0, 0}  
{0, 0, 0, 0, 0}
```

## Conclusion

This document has expanded upon the framework for evaluating MAC/PHY performance presented in 92/123. This expansion has taken the form of a preliminary implementation of one of the MACs that has been presented to the MAC group. The requested action from this submittal is to close issue 29.1 and to direct the parties working on channel models, MACs and MAC/PHY throughput presentations to begin working with a common set of tools developed in Mathematica.
Evaluating TX Conditions between Access1 and Station1
Evaluating TX Conditions between Access1 and Station2
Evaluating TX Conditions between Access1 and Station3
Evaluating TX Conditions between Access1 and Station4
Evaluating TX Conditions between Station1 and Access1
Evaluating TX Conditions between Station2 and Access1
Evaluating TX Conditions between Station3 and Access1
Evaluating TX Conditions between Station4 and Access1

Station1 is Transmitting
Station2 is Transmitting
Station3 is Transmitting
Station4 is Transmitting
Evaluating TX Conditions between Station5 and Access1

Access1 is ReceiveBPeriod
TXOueues((empty), (empty), (empty), (empty), [Station3, empty, empty, empty),
(Station5, Station2, Station2, Station2, Station2, Station2, empty),
(Station2, Station2, Station2, Station2, empty))
RXOueues((empty, [Station5, Station3, Station2]), [empty, empty], [empty, empty],
[empty, empty, empty, empty])

Station1 is WaitingForCH
Station2 is InactiveBPeriod
Station3 is WaitingForCH
Station4 is InactiveBPeriod
Station5 is TransmitBPeriod

Evaluating TX Conditions between Station5 and Access1

Access1 is ReceiveBPeriod
TXOueues((empty), (empty), (empty), (empty), [Station3, empty, empty, empty),
(Station5, Station2, Station2, Station2, Station2, Station2, empty),
(Station2, Station2, Station2, Station2, empty))
RXOueues((empty, [Station5, Station3, Station2]), [empty, empty], [empty, empty],
[empty, empty, empty, empty])

Station1 is WaitingForCH
Station2 is InactiveBPeriod
Station3 is WaitingForCH
Station4 is InactiveBPeriod
Station5 is TransmitBPeriod

Evaluating TX Conditions between Station5 and Access1

Access1 is ReceiveBPeriod
TXOueues((empty), (empty), (empty), (empty), [Station3, empty, empty, empty),
(Station5, Station2, Station2, Station2, Station2, Station2, empty),
(Station2, Station2, Station2, Station2, empty))
RXOueues((empty, [Station5, Station3, Station2]), [empty, empty], [empty, empty],
[empty, empty, empty, empty])

Station1 is WaitingForCH
Station2 is InactiveBPeriod
Station3 is WaitingForCH
Station4 is InactiveBPeriod
Station5 is TransmitBPeriod

Evaluating TX Conditions between Station5 and Access1

Access1 is ReceiveBPeriod
TXOueues((empty), (empty), (empty), (empty), [Station3, empty, empty, empty),
(Station5, Station2, Station2, Station2, Station2, Station2, empty),
(Station2, Station2, Station2, Station2, empty))
RXOueues((empty, [Station5, Station3, Station2]), [empty, empty], [empty, empty],
[empty, empty, empty, empty])

Station1 is WaitingForCH
Station2 is InactiveBPeriod
Station3 is WaitingForCH
Station4 is InactiveBPeriod
Station5 is TransmitBPeriod

Evaluating TX Conditions between Station5 and Access1

Access1 is ReceiveBPeriod
TXOueues((empty), (empty), (empty), (empty), [Station3, empty, empty, empty),
(Station5, Station2, Station2, Station2, Station2, Station2, empty),
(Station2, Station2, Station2, Station2, empty))
RXOueues((empty, [Station5, Station3, Station2]), [empty, empty], [empty, empty],
[empty, empty, empty, empty])

Station1 is WaitingForCH
Station2 is InactiveBPeriod
Station3 is WaitingForCH
Station4 is InactiveBPeriod
Station5 is TransmitBPeriod

Evaluating TX Conditions between Station5 and Access1

Access1 is ReceiveBPeriod
TXOueues((empty), (empty), (empty), (empty), [Station3, empty, empty, empty),
(Station5, Station2, Station2, Station2, Station2, Station2, empty),
(Station2, Station2, Station2, Station2, empty))
RXOueues((empty, [Station5, Station3, Station2]), [empty, empty], [empty, empty],
[empty, empty, empty, empty])

Station1 is WaitingForCH
Station2 is InactiveBPeriod
Station3 is WaitingForCH
Station4 is InactiveBPeriod
Station5 is TransmitBPeriod

Evaluating TX Conditions between Station5 and Access1

Access1 is ReceiveBPeriod
TXOueues((empty), (empty), (empty), (empty), [Station3, empty, empty, empty),
(Station5, Station2, Station2, Station2, Station2, Station2, empty),
(Station2, Station2, Station2, Station2, empty))
RXOueues((empty, [Station5, Station3, Station2]), [empty, empty], [empty, empty],
[empty, empty, empty, empty])

Station1 is WaitingForCH
Station2 is InactiveBPeriod
Station3 is WaitingForCH
Station4 is InactiveBPeriod
Station5 is TransmitBPeriod

Evaluating TX Conditions between Station5 and Access1

Access1 is ReceiveBPeriod
TXOueues((empty), (empty), (empty), (empty), [Station3, empty, empty, empty),
(Station5, Station2, Station2, Station2, Station2, Station2, empty),
(Station2, Station2, Station2, Station2, empty))
RXOueues((empty, [Station5, Station3, Station2]), [empty, empty], [empty, empty],
[empty, empty, empty, empty])

Station1 is WaitingForCH
Station2 is InactiveBPeriod
Station3 is WaitingForCH
Station4 is InactiveBPeriod
Station5 is TransmitBPeriod

Evaluating TX Conditions between Station5 and Access1

Access1 is ReceiveBPeriod
TXOueues((empty), (empty), (empty), (empty), [Station3, empty, empty, empty),
(Station5, Station2, Station2, Station2, Station2, Station2, empty),
(Station2, Station2, Station2, Station2, empty))
RXOueues((empty, [Station5, Station3, Station2]), [empty, empty], [empty, empty],
[empty, empty, empty, empty])

Station1 is WaitingForCH
Station2 is InactiveBPeriod
Station3 is WaitingForCH
Station4 is InactiveBPeriod
Station5 is TransmitBPeriod

Evaluating TX Conditions between Station5 and Access1

Access1 is ReceiveBPeriod
TXOueues((empty), (empty), (empty), (empty), [Station3, empty, empty, empty),
(Station5, Station2, Station2, Station2, Station2, Station2, empty),
(Station2, Station2, Station2, Station2, empty))
RXOueues((empty, [Station5, Station3, Station2]), [empty, empty], [empty, empty],
[empty, empty, empty, empty])

Station1 is WaitingForCH
Station2 is InactiveBPeriod
Station3 is WaitingForCH
Station4 is InactiveBPeriod
Station5 is TransmitBPeriod

Evaluating TX Conditions between Station5 and Access1

Access1 is ReceiveBPeriod
TXOueues((empty), (empty), (empty), (empty), [Station3, empty, empty, empty),
(Station5, Station2, Station2, Station2, Station2, Station2, empty),
(Station2, Station2, Station2, Station2, empty))
RXOueues((empty, [Station5, Station3, Station2]), [empty, empty], [empty, empty],
[empty, empty, empty, empty])

Station1 is WaitingForCH
Station2 is InactiveBPeriod
Station3 is WaitingForCH
Station4 is InactiveBPeriod
Station5 is TransmitBPeriod

Evaluating TX Conditions between Station5 and Access1

Access1 is ReceiveBPeriod
TXOueues((empty), (empty), (empty), (empty), [Station3, empty, empty, empty),
(Station5, Station2, Station2, Station2, Station2, Station2, empty),
(Station2, Station2, Station2, Station2, empty))
RXOueues((empty, [Station5, Station3, Station2]), [empty, empty], [empty, empty],
[empty, empty, empty, empty])

Station1 is WaitingForCH
Station2 is InactiveBPeriod
Station3 is WaitingForCH
Station4 is InactiveBPeriod
Station5 is TransmitBPeriod

Evaluating TX Conditions between Station5 and Access1

Access1 is ReceiveBPeriod
TXOueues((empty), (empty), (empty), (empty), [Station3, empty, empty, empty),
(Station5, Station2, Station2, Station2, Station2, Station2, empty),
(Station2, Station2, Station2, Station2, empty))
RXOueues((empty, [Station5, Station3, Station2]), [empty, empty], [empty, empty],
[empty, empty, empty, empty])

Station1 is WaitingForCH
Station2 is InactiveBPeriod
Station3 is WaitingForCH
Station4 is InactiveBPeriod
Station5 is TransmitBPeriod
Station 4 is Transmitting Period
Evaluating TX Conditions between Station 4 and Access 1
Station 5 is Waiting For CH
Access 1 is Receive Period
TX Queue: \{empty, empty, empty, \}, \{Station 3, empty, empty\}, \{Station 5, Station 2, Station 5, Station 2, Station 2, Station 2, Station 2, Station 2, Station 2, Station 2, Station 2, Station 2, Station 2, Station 2, Station 2\}, \{empty\}
RX Queue: \{empty, \}, \{empty, \}, \{empty, \}, \{Station 3, empty, \}, \{Station 5, Station 3, Station 2, Station 2, Station 2, Station 2, \}, \{empty\}
Station 1 is Waiting For CH
Station 2 is Inactive Period
Station 3 is Transmitting Period
Evaluating TX Conditions between Station 4 and Access 1
Station 5 is Waiting For CH
Access 1 is Receive Period
TX Queue: \{empty, empty, empty, \}, \{Station 3, empty, empty\}, \{Station 5, Station 2, Station 5, Station 2, Station 2, Station 2, \}, \{empty\}
RX Queue: \{empty, \}, \{empty, \}, \{empty, \}, \{Station 3, empty, \}, \{Station 5, Station 3, Station 2, Station 2, Station 2, \}, \{empty\}
Station 1 is Waiting For CH
Station 2 is Inactive Period
Station 3 is Transmitting Period
Evaluating TX Conditions between Station 4 and Access 1
Station 5 is Waiting For CH
Access 1 is Receive Period
TX Queue: \{empty, empty, empty, \}, \{Station 3, empty, empty\}, \{Station 5, Station 2, Station 5, Station 2, Station 2, Station 2, \}, \{empty\}
RX Queue: \{empty, \}, \{empty, \}, \{empty, \}, \{Station 3, empty, \}, \{Station 5, Station 3, Station 2, Station 2, Station 2, \}, \{empty\}
Station 1 is Waiting For CH
Station 2 is Inactive Period
Station 3 is Transmitting Period
Evaluating TX Conditions between Station 4 and Access 1
Station 5 is Waiting For CH
Access 1 is Receive Period
TX Queue: \{empty, empty, empty, \}, \{Station 3, empty, empty\}, \{Station 5, Station 2, Station 5, Station 2, Station 2, Station 2, \}, \{empty\}
RX Queue: \{empty, \}, \{empty, \}, \{empty, \}, \{Station 3, empty, \}, \{Station 5, Station 3, Station 2, Station 2, Station 2, \}, \{empty\}
Station 1 is Waiting For CH
Station 2 is Inactive Period
Station 3 is Transmitting Period
Evaluating TX Conditions between Station 4 and Access 1
Station 5 is Waiting For CH
Access 1 is Receive Period
TX Queue: \{empty, empty, empty, \}, \{Station 3, empty, empty\}, \{Station 5, Station 2, Station 5, Station 2, Station 2, Station 2, \}, \{empty\}
RX Queue: \{empty, \}, \{empty, \}, \{empty, \}, \{Station 3, empty, \}, \{Station 5, Station 3, Station 2, Station 2, Station 2, \}, \{empty\}
Station 1 is Waiting For CH
Station 2 is Inactive Period
Station 3 is Transmitting Period
Evaluating TX Conditions between Station 4 and Access 1
Station 5 is Waiting For CH
Access 1 is Receive Period
TX Queue: \{empty, empty, empty, \}, \{Station 3, empty, empty\}, \{Station 5, Station 2, Station 5, Station 2, Station 2, Station 2, \}, \{empty\}
RX Queue: \{empty, \}, \{empty, \}, \{empty, \}, \{Station 3, empty, \}, \{Station 5, Station 3, Station 2, Station 2, Station 2, \}, \{empty\}
Station 1 is Waiting For CH
Station 2 is Inactive Period
Station 3 is Transmitting Period
Evaluating TX Conditions between Station 4 and Access 1
Station 5 is Waiting For CH
Access 1 is Receive Period
TX Queue: \{empty, empty, empty, \}, \{Station 3, empty, empty\}, \{Station 5, Station 2, Station 5, Station 2, Station 2, Station 2, \}, \{empty\}
RX Queue: \{empty, \}, \{empty, \}, \{empty, \}, \{Station 3, empty, \}, \{Station 5, Station 3, Station 2, Station 2, Station 2, \}, \{empty\}
Station 1 is Waiting For CH
Station 2 is Inactive Period
Station 3 is Transmitting Period
Evaluating TX Conditions between Station 4 and Access 1
Station 5 is Waiting For CH
Access 1 is Receive Period
TX Queue: \{empty, empty, empty, \}, \{Station 3, empty, empty\}, \{Station 5, Station 2, Station 5, Station 2, Station 2, Station 2, \}, \{empty\}
RX Queue: \{empty, \}, \{empty, \}, \{empty, \}, \{Station 3, empty, \}, \{Station 5, Station 3, Station 2, Station 2, Station 2, \}, \{empty\}
Evaluating TX Conditions between Access1 and Station1
Station1 is TransmittingCH
TXQueues[empty], [empty], [empty], [empty], [empty], [empty], [Station3, empty], [empty], [empty], [empty], [empty], [empty], [empty], [empty]
RXQueues[empty], [Station5, Station3], [Station2], [Station2, Station2, Station2, Station2, Station2], [empty], [empty], [empty], [empty], [empty], [empty], [empty], [empty]
Access1 is ReceiveBPeriod
Station1 is Transmitting
Station2 is Transmitting
Station3 is Transmitting
Station4 is Transmitting
Access1 is ReceiveBPeriod
TXQueues[empty], [empty], [empty], [empty], [Station3, empty], [empty], [empty], [empty], [empty], [empty], [empty], [empty], [empty], [empty]
RXQueues[empty], [Station5, Station3], [Station2], [Station2, Station2, Station2, Station2, Station2], [empty], [empty], [empty], [empty], [empty], [empty], [empty], [empty], [empty], [empty]
Access1 is ReceiveBPeriod
Station1 is Transmitting
Station2 is Transmitting
Station3 is Transmitting
Station4 is Transmitting
Station1 is Transmitting
TXQueues[empty], [empty], [empty], [empty], [Station3, empty], [empty], [empty], [empty], [empty], [empty], [empty], [empty], [empty], [empty]
RXQueues[empty], [Station5, Station3], [Station2], [Station2, Station2, Station2, Station2, Station2], [empty], [empty], [empty], [empty], [empty], [empty], [empty], [empty], [empty], [empty]
Access1 is ReceiveBPeriod
Station1 is Transmitting
Station2 is Transmitting
Station3 is Transmitting
Station4 is Transmitting
Station1 is Transmitting
TXQueues[empty], [empty], [empty], [empty], [Station3, empty], [empty], [empty], [empty], [empty], [empty], [empty], [empty], [empty], [empty]
RXQueues[empty], [Station5, Station3], [Station2], [Station2, Station2, Station2, Station2, Station2], [empty], [empty], [empty], [empty], [empty], [empty], [empty], [empty], [empty], [empty]
Access1 is ReceiveBPeriod
Station1 is Transmitting
Station2 is Transmitting
Station3 is Transmitting
Station4 is Transmitting
Station1 is Transmitting