Bilkent University, Department of Computer Engineering
CS 514 – Wireless Ad Hoc and Sensor Networks, Fall 2004
Instructor: Ibrahim Korpeoglu

Development Project 1:
Multi-threaded Simulation of 802.11 CSMA/CA Distributed Wireless MAC Protocol

In this project you will implement a multi-threaded simulation of the CSMA/CA distributed MAC protocol of 802.11 technology for a simplified environment. The followings are the assumptions for the simplified environment:

- A set of *N* mobile stations will send data traffic to an access point. This means that we have a single destination for all data traffic and all data traffic is one-way. The single destination station, in fact, does not have to be an access point. It could be just another mobile station. The MAC logic you will implement in this project will be the same for all the mobile stations and the access point. You should not distinguish the mobile stations and the access point while implementing your protocol. We call a station sending data to the access point as a mobile station just for the popularity of the term. You will not simulate any mobility.
- All mobile stations and access point are sharing a single wireless radio channel.
- All stations can hear each other. There is no hidden terminal problem.
- There is no central coordination by the access point.
- There are no bit errors occurring on the channel. A transmitted frame can get corrupted only if it collides with another frame transmitted by another station.
- There is no mechanism used in a mobile station to detect collisions. Hence a mobile station will continue transmitting a frame until completion even through the frame has collided with some other frame. The only way to detect a collision is by detecting a lacking acknowledgment.
- Frames are not fragmented.
- No RTS/CTS exchange will occur.
- When a data packet is successfully received by the access point, it has to be acknowledged. Therefore, a successfully frame exchange sequence consists of a data frame plus an ACK frame: DATA + ACK.
- We will ignore propagation delay, and take it as zero.

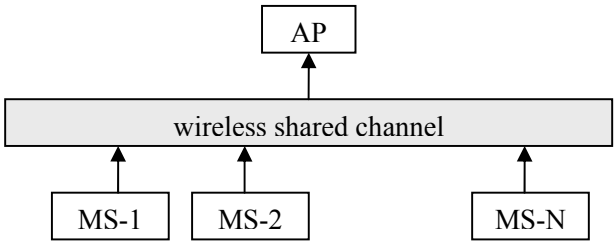

Figure 1. A simple wireless local area network model where traffic will flow only in one direction: from mobile stations to a single destination which is an access point here.

The project can be implemented using C, C++, C#, or Java, either on Linux or Windows XP. The project has to be done in groups of 3-4 people.

CSMA/CA protocol to implement:

Follow the CSMA/CA algorithm given in the book plus the following clarifications.

If a station has a frame to transmit, it waits for a DIFS interval and if the channel is still idle, it transmits immediately. If the channel is busy, it will wait until medium is idle for a DIFS period and then start the backoff process.

When backoff process is started at a station, the station first selects a random slot between 0 and *CWmin* (31). The slot number gives the initial value of the backoff counter. The station will decrease the backoff counter as long as the channel is idle. When the channel becomes busy, the backoff counter will be frozen. It will be decreasing again after the channel becomes idle for a period of DIFS. When the backoff counter of the station reaches to zero, the station starts transmitting the frame.

When a transmitted frame from a station collides and the station therefore does not receive any ACK frame, the station will double the contention window and select another random slot in the new window to be used to initialize the backoff counter before the next frame transmission.

To understand if a frame is transmitted successfully or not, the sending station will wait for an ACK. The time period to wait for an ACK will be equal to the SIFS period in this project. This is because, if the transmission is successful, then the receiver will send an ACK frame immediately after waiting an SIFS period of time. Therefore, the maximum amount of time that can elapse at the sender side between the end of transmission of a data frame and the reception of the corresponding ACK frame can be at most SIFS. Hence, if after sending a data frame the sender still did not receive a corresponding ACK after a SIFS period, it can conclude that a collision has occurred during the transmission of the data frame.

If still no ACK is received after a SIFS period, then the sending station can conclude that the frame has collided. After this detection, the sending station will wait for an EIFS period and then will increase the *CW*, and select a new random slot to wait for before the next retransmission of the frame.

If total number of retransmissions is reached its maximum value at a sending station, the sending station will give up with the current frame and drop it. The sending station will also set the congestion window value to *CWmin*. A frame can be retransmitted at most *M* times, including the initial transmission.

If a frame has been successfully transmitted after some number of retransmissions, the congestion window value will be set to *CWmin* again for use in the next frame transmission.

At each retransmission, the contention window (*CW*) is doubled. The maximum value can be 1023 (*CWmax*). The minimum value can be 31 (*CWmin*). Doubling of the contention window should be done according to the following formula:

new $CW = 2 * (\text{current } CW + 1) - 1$.

A current frame transmission will get a collision if another frame is transmitted before the current frame transmission ends.

Implementation requirements:

The simulation has to be implemented as multi-thread process. For each mobile station you should have a different thread simulating the autonomous behavior of a mobile station. The radio channel should be simulated as a shared data structure (a global variable) that is shared among multiple threads. Each thread can access and modify a global variable in a multi-threaded program.

Make sure that you handle race conditions that can possibly occur while multiple independent threads are accessing the shared data structure.

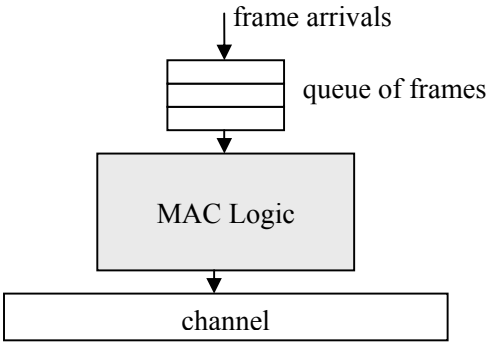

Figure 2. Illustration of frame arrivals to the MAC layer from higher layers and the queuing of those frames. Frames are of variable length.

Parameters of the simulator:

- ***N***: number of mobile stations.
- ***M***: maximum number of retransmissions including the first transmission that a station can attempt.
- *CWmin*: minimum congestion window size (default value = 31).
  CWmax: maximum congestion window size (default value = 1023).
- *SIFS*: short interframe space (default value = 10 μsec).
- *DIFS*: distributed coordination function interframe space (default value = 50 μsec).
- *EIFS*: extended interframe space (default value = 80 μsec).
- t_s: time-slot length (default value = 20 μsec).
- *R*: data rate (default value = 10 Mbps)
- *Fmin, Fmax*: minimum and maximum length of a data frame in microseconds (default values = 100 and 1000). The length of a data frame can be a number between *Fmin* and *Fmax*, and it should be a multiple of 20. For example, 220 is valid frame length, but 210 is not. Assume a frame that has a length of 100 microseconds corresponds to a length of 1000 bits, and a frame that has a length of 1000 microseconds corresponds to a length of 10000 bits. Assume that the valid frame lengths are uniformly distributed between *Fmin* and *Fmax*. You should use this property while generating a random frame length.
- *L_ACK*: length of an ACK frame in microseconds. (default value = 20).
- ***T***: total simulation time in microseconds.
- ***avgIAT***: average interarrival time between two frames. It has to be a multiple of 20. The minimum interarrival time should be 20. You should obtain a random interarrival time by first generating an exponentially distributed random variate with average value of *avgIAT* and then truncating or rounding the value to be a multiple of 20.

The parameters that are shown in bold above will take values from command line when the simulator is invoked.

Simulating the load of a mobile station:

The frame arrivals to a mobile station has to be modeled in the following way. A random *IAT* (in microsecods) will be selected according to the approach described earlier. The selected value (*IAT*) has to be a multiple of 20. This value will be the time interval between the current frame and the next frame. Next frame arrival then can be computed as follows: the arrival time of the current frame + *IAT*. Assume, when simulation is started up, time starts at 0 and the first *IAT* computed in this way will be the arrival time of the first frame.

Assume that all mobile stations are exposed to a load with the same parameters. But you should use a different random number sequence for simulating interarrival times of frames in each mobile station.

Invoking the simulator:

The possible invocation of the simulator will be as follows. You should name you program as "`mac`".

```
mac –n stationcount –m retransmitcount –t simulationtime
    -avgiat averageinterarrivaltime
    [-f inputfilename]
```

The meaning of the options are as follows:

`-n`: number of mobile stations.

`-m`: maximum number of retransmissions

`-t`: simulation time in microseconds.

`-avgiat`: average interarrival time between consecutives frames arriving to a station.

`-f`: this is an optimal parameter. It is used only when a tester would like to obtain the interarrival times between frames from input text files (one file for each station). If for example this option has a value of "xyz", then the input file corresponding to the mobile station 3 will have a name "xyz3". An input file will just contain a sequence of pair of integers, one pair per line. In a line, the first integer denotes the interarrival time before a frame arrives and the second integer denotes the length (in microseconds) of the frame. If this option is missing, then the interarrival times and frame lengths have to be generated using random number generators.

Required output at each run of the simulator:

- Percent of time channel was totally idle (*TI*).
- Percent of time channel was carrying uncollided data frames (*U1*).
- Percent of time channel was carrying uncollided data + ACK frames (*U2*).
- Average wait time of a frame: time between arrival of a frame and start of transmission of the frame (*D*).
- For each station *i*, total number of bits successfully transmitted and received by the access point (A_i).
- For each station *i*, the average goodput the station obtained during the simulation time (G_i).
- Total goodput achieved over the channel (*TG*).
- Total number of collisions occurred during the simulation time (*TC*).

The goodput of a transfer (in bits per second) is computed as follows: (the total amount of application data bits transmitted successfully during a time interval) / (duration of the time interval). Here the duration of the time interval will be equal to the total simulation time.

Experiments and results:

Fix the number of stations to be 5. By running your simulator with different parameters (i.e. by doing experiments), obtain the following charts.

- x-axis: decreasing *avgIAT* (increasing load); y-axis: *TC*
- x-axis: decreasing *avgIAT* (increasing load); y-axis: *TG*
- x-axis: station number (*i*); y-axis: G_i
- x-axis: station number (*i*); y-axis: A_i
- x-axis: decreasing *avgIAT* (increasing load); y-axis: *TI*
- x-axis: decreasing *avgIAT* (increasing load); y-axis: *U1*
- x-axis: decreasing *avgIAT* (increasing load); y-axis: *D*

Include the charts that you obtained in the final report that you will give to the instructor. Try to interpret the charts also and include your interpretation in the final report also.

Deliverables:

You should tar the directory containing your project and email it to the instructor as an attachment. You will also make a demo of your program. The instructor may look through you code and ask you questions about your program. You should also give a report to the instructor in hardcopy form. The report should describe your design and implementation, and also should provide your results.