King Fahd University of Petroleum & Minerals

College of Computer Science & Engineering

COE-541 Local & Metropolitan Area Networks

Simulation & Performance Evaluation of CSMA/CA

Project Report

Presented by

Muhamad Khaled Alhamwi
260212

Submitted to

Prof. Mayez Al-Mohammad

1 Introduction

Wireless networks are becoming of more interest in recent days. They are easy to use and configure. The most famous and widely-used standard is IEEE802.11 for WLAN. As in wired networks, collisions can occur in wireless LANs when more than on station transmits at the same time while using a shared transmission medium. Of course, the air is the shared medium in wireless LANs. IEEE802.11 [1]standard [1] defines the medium access methods in the MAC layer. It has two main functions namely: distributed coordination function (DCF), and point coordination function (PCF). DCF is mandatory while PCF is optional in the standard. In this work, we are concerned with DCF functionality only. So, we will not consider PCF further in our discussion.

DCF employs carrier sense multiple access with collision avoidance (CSMA/CA). This differs from CSMA/CD which is carrier sense multiple access with collision detection. Stations that transmit over the wireless channel can not detect collisions because they are not capable of transmitting and receiving at the same time. So, CSMA/CA tried to avoid collisions instead of detecting them. It employs the mechanism called: "listen before talk", which means that every station has to sense the medium for possible transmission. If the medium was sensed idle for a certain period of time, the station proceeds with transmission of the packet. Otherwise, the station defers the transmission until the medium becomes idle. Then, it calls the backoff algorithm. More details will be presented about this protocol in the next sections.

2 Related Work

CSMA/CA protocol is there in the literature since a long period of time. There exist many papers that discuss the performance issues of this protocol and try to propose different ways of enhancing its performance. A lot of research has been done to improve the backoff procedure of the CSMA/CA protocol by changing the contention window adaptively. The backoff algorithm greatly affects the performance of CSMA/CA because it determines the time that the stations wait for before transmission when the medium is sensed busy. So, if we have big delays without a clear advantage, it would cause low utilization of the channel and waste in the available bandwidth. On the hand, if we have very small delays between retransmission trials, we would have more and more collisions and again the result will be low utilization of the channel and wasting of the bandwidth.

The work of [1][2] proposes a two-stage algorithm to enhance the performance of the backoff algorithm of CSMA/CA protocol and consequently enhancing the performance of the IEEE802.11 WLAN. This algorithm works in a similar way to the standard. It increases the contention window when a station encounters a collision, and decreases it when a station has a successful transmission. However, this algorithm restricts the possible values of the contention window to only two values.

A 2-D Markovian model for the binary exponential algorithm is presented in [4]. In this model, different stages were defined for the backoff algorithm with their probabilities. Also the authors provided an accurate analysis of the network throughput for basic access and RTS/CTS modes. The authors of [3] provided a simpler 1-D Makovian model for BEB (Binary Exponential Backoff) algorithm. This was provided under the assumption that the collision probability is constant regardless of the backoff stage. Surprisingly, this assumption was valid and they got exactly the same formula for network throughput. Different backoff algorithms were presented and analyzed with their Markovian models in [3]. The BEB works by doubling the contention window in case of collision and resetting it to its minimum value in case of successful transmission. The EIED (Exponential Increase Exponential Decrease) algorithm works by doubling in case of collision and halving in case of a successful transmission. The EILD (Exponential Increase Linear Decrease) algorithm works similarly to EIED, but it decrements one from the window in case of a successful transmission. These algorithms were compared under saturation assumption.

An enhanced backoff algorithm is also presented in [5]. In this work, the authors proposed a modified backoff algorithm that changes its contention window dynamically around the optimal value. After a successful transmission, the window w is set to Max(w/2, CWmin), and set to Min(2w, CWmax) after collision. We can see that this algorithm works like EIED described in [3] with increase rate of 2 and decrease rate of 2.

The authors of [6] found formula for the theoretical throughput of IEEE802.11 and proved that the standard works far from this optimal throughput. Then, they proposed techniques to get higher throughput closer to the theoretical limit that they defined. They did extensive simulations to show the effectiveness of their approach.

An optimal value for the contention window was found by [7]. This was done for both access modes: basic access and RTS/CTS. The linear feedback model was used to compute the throughput and delay of the protocol. They showed that the protocol performance degrades greatly when the hidden terminal problem dominates in the network, and when the packets size is very large. In this case, the usage of RTS/CTS becomes essential.

An adaptive contention window was proposed in [8] which depends on the estimated number of active stations in the network. Then, a formula is applied to find the contention window size according to the estimated value and packet transmission time. Then, the performance of this approach was evaluated. It concluded that this approach outperforms the standard protocol.

3 Proposed Approaches

We have proposed two approaches to implement CSMA/CA protocol simulator: the single-threaded event-driven approach which depends on events to simulate the protocol operations, and the multithreaded approach which requires two threads per station. These two approaches are described in more details in the next subsections.

3.1 Single-Threaded Event-driven approach

In this approach, the simulator is programmed using one single thread that performs all the operations like processing all types of events, and updating the simulation statistics. This approach is simple to program when you have the events diagram ready and correct, but it is not so efficient in terms of performance if it is not programmed properly in a way that tries to minimize the number of events created and processed.

```
Initialize variables;
while (overallProcessedEvents < MAX_EVENTS {
    events = getNextEvent(list, simTime);
    simTime = events(0).time;
    while ( length(events) > 0 )
        foreach event in events {
            processed  = processEvent(list, event, simTime);
            if ( processed < length(events) )
                Set 'events' to the newly created events
            else
                events = [];
        }
    }
}
Generate Report;
```

First, we have to initialize the variables that will be used in the simulator. Then, the outer while loop checks whether we have to terminate the simulation. The termination condition can be the number of processed events so far, or it can be the time spent till now. In the shown pseudo-code, the termination condition checks for the total number of processed events. During each iteration, the new events (earliest ones) should be selected from the list of events and processed. The 'for' loop goes over all of the events in 'events' variable and call the function 'processEvent'.

Some new events may be created while processing the current events and need to be process in the same iteration. This happens when the newly created events have the same time as the simulator time 'simTime' variable.

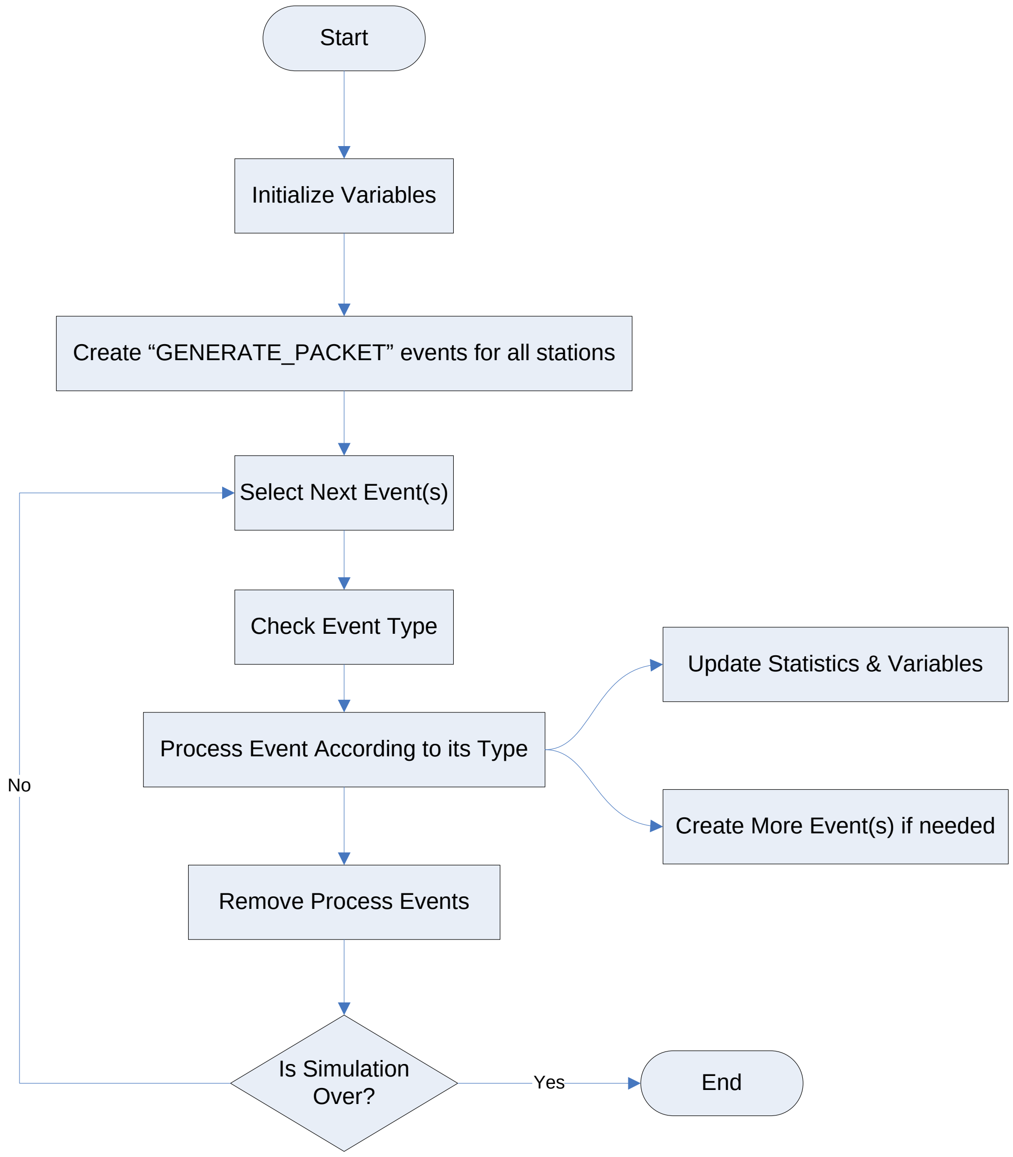

Start
Initialize Variables
Create “GENERATE_PACKET” events for all stations
Select Next Event(s)
Check Event Type
Process Event According to its Type
Update Statistics & Variables
Create More Event(s) if needed
Remove Process Events
Is Simulation Over?
No
Yes
End

Figure 1. Simulator Flowchart

A simplified flowchart of the event-driven simulator is shown in Figure 1. When processing the current event(s), one or more events can be created if needed according to the event types as will be shown later. After processing the current event(s), they should be removed from the events list.

3.2 *Multi-Threaded Approach*

In this approach, the simulator program is composed of multiple threads that cooperate in order to achieve the simulation purpose. Each station can be assigned two threads: one for packets generation and the other one for processing these packets and trying to send them. So, each station has a producer thread and a consumer thread to generate and consume packets respectively. In addition to these threads, there is one more main thread that checks for waiting threads and wakeup some or all of them depending on a waiting condition that is associated with each thread.

This approach is more realistic because it reflects the real operation of each station independently. Each station is represented by two threads one to generate and another to process packets. However, this approach is less extensible when compared with other approaches because of the high cost associated with creating so many threads to support the simulation for a big number of stations. This may slow down the simulation speed as the need for context switching between threads will increase, and we will have more sleeping threads occupying space in memory without doing any useful work for most of the time. Here is the pseudo-code for this approach.

3.2.1 Thread TrafficGeneratorThread

```
void run() {
    while (true) {
        p = generatePacket();
        stationQueue.addPacket(p);
        if (stationQueue.packetsCount == 1) {
            stationServer.wakUp();
        }
        duration = getInterArrival();
        waitingList.addThread(this, WAIT_TIME, duration);
        suspend current thread;
    }
}
```

3.2.2 Thread StationServerThread

```
void run() {
    while (true) {
        if (stationQueue.packetsCount == 0) {
            waitingList.addThread(this, WAIT_PACKETS, 0);
            suspend this thread;
        }
         p = stationQueue.getFirstPacket();
        if (medium.isIdle()) {
            waitingList.addThread(this, WAIT_TIME, DIFS);
            suspend this thread;
            transmit(p);
        }
        else {
            waitingList.addThread(this, WAIT_MEDIUM_FREE, 0);
            this.suspend();
            waitingList.addThread(this, WAIT_TIME, DIFS);
            this.suspend();
            Initialize backoffTimer
            while (backoffTimer != 0) {
                waitingList.addThread(this, WAIT_TIMER_MEDIUM_BUSY, backoffTimer);
                Update backoffTimer
                this.suspend();
                if (medium.isBusy()) {
                    waitingList.addThread(this, WAIT_MEDIUM_FREE, 0);
                    this.suspend();
                    waitingList.addThread(this, WAIT_TIME, DIFS);
                    this.suspend();
                }
            }
            transmit(p);
        }
    }
}
```

3.2.3 Thread MainThread

```
medium = new Medium();
stations = new Stations[numOfStations];
for i = 1 to numOfStations {
    stations[i] = new Station(new TrafficGeneratorThread(),new StationServerThread(medium));
    stations[i].startThreads();
}
simTime = 0;
while(simTime < MAX_SIM_TIME) {
    if (waitingList.getSleepingThreadsCount() == 0) {
        sleep for some time;
    }
    waitingElems = waitingList.getFirstThreads(medium);
    simTime += waitingElems.elementAt(0).value;
    for i = 1 to waitingElems.size() {
        waitingElems.elementAt(i).thread.resume();
    }
}
```

The following are the data structures that are needed by the previous code shown earlier. We can see that we have the keyword 'synchronized' before the declaration of some functions. This is needed for synchronization among the running threads as they may compete to access some data structure. If we allow the running threads to access the data structures without controlling them properly, we will have race condition problems and inconsistent data.

3.2.4 Data Structures

StationQueue:
```
Vector packets;
synchronized void addPacket(Packet p);
synchronized Packet getFirstPacket();
```

Medium:
```
int state;
synchronized boolean isIdle();
synchronized boolean isBusy();
synchronized void changeState(newState);
```

WaitingListElem:
```
Thread threadP;
int reason;
int value;
const int WAIT_PACKETS = 0;
const int WAIT_TIME = 1;
const int WAIT_MEDIUM_FREE = 2;
const int WAIT_TIMER_MEDIUM_BUSY = 3;
```

4 Implemented Simulator

The approach that we used in this work to implement CSMA/CA protocol simulator is the event-driven one. We have eleven different types of events. The events' types are shown in the following table with a brief description for each type:

Event Type	**Meaning / Usage**
GENERATE_PACKET	This event triggers the creation of a new packet when fired.
START_CSMA	This event triggers the beginning of the frame transmission trial.
DEFER	This event is used to defer the transmission of a station

	because the medium was sensed busy.
START_BACKOFF	This event starts the backoff period of a station
DEC_BACKOFF	This event decrements the backoff timer of a station and possibly trigger transmission event.
STOP_BACKOFF	This station is similar to DEC_BACKOFF. However, it is only created when a station has DEC_BACKOFF and the medium becomes busy before firing this event.
TRANSMIT_FRAME	This event starts transmitting the packet that is the head of the station queue. It changes the medium state from idle to busy or collision depending on the number of simultaneous transmissions.
TRANSMIT_ACK	This event starts sending acknowledgement to the sending station. It also changes the medium state from idle to busy.
FREE_MEDIUM	This event frees the medium by changing its state from busy or collision to idle again.
NO_ACK	This event is fired when the sending station did not receive an acknowledgment within the allowed timeout. It increments the transmission attempts counter, and adjusts the contention window according to the backoff algorithm. This event creates a START_CSMA event for retransmission of the lost packet if the retry limit has not been reached.
RECV_ACK	This event is fired when the sending station receives an acknowledgment for a previously sent packet. This will cause the removal of the sent packet from the station queue.

Each event has three attributes: the station to which it belongs, its type, and at which time it should fires. The following diagram shows the relation among the different types and how they are created:

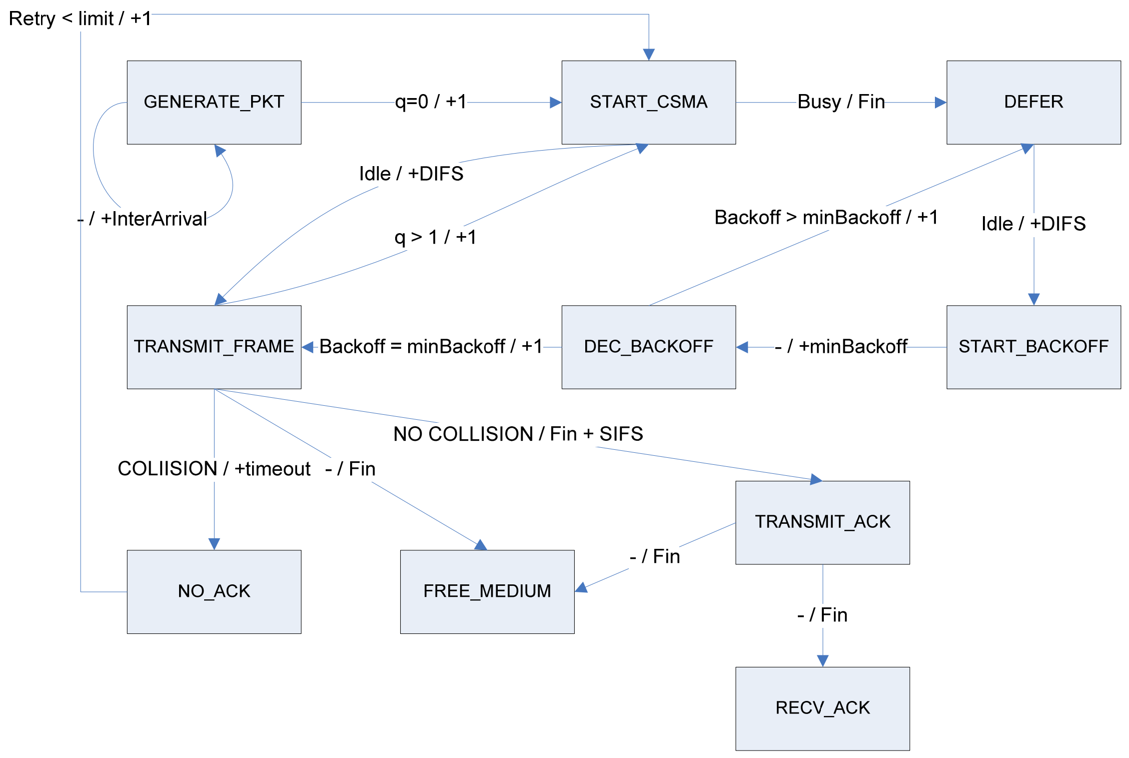


Figure 2. The events diagram of CSMA/CA simulator

Each event type is represented by a rectangular box. Each arrow represents the creation of a new event as the result of processing the current event. On each arrow, we can see that there is a '/' character separating two statements. The statement on the left represents the condition under which the new event is created, and the statement on the right represents the time associated with the newly created event. If we have '+' at the beginning of the time, this means that the new event time will be the current simulator time + the amount after it. If we don't have a '+', this means that the new event time is equal to the indicated time. For example, the FREE_MEDIUM event will be created from TRANSMIT_FRAME event under no condition and at time 'Fin', where 'Fin' is time at which the frames transmission completes. We can see STOP_BACKOFF event is not shown in the events diagram. This will be explained later.

The GENERATE_PACKET event is always creating another event of the same type after the inter-arrival time. This is indicated by the loop shown in the above figure. This event can generate a START_CSMA event when the queue is empty. This is necessary to initialize the transmission for the first time, or when the queue becomes empty for some time and station becomes inactive. The START_CSMA event can generate TRANSMIT_FRAME event if the medium is sensed idle. Otherwise, it generated a DEFER event until the medium becomes idle again (Fin). The DEFER event generates START_BACKOFF event after DIFS interval if the medium is sensed idle. The START_BACKOFF event generates DEC_BACKOFF event after the minimum backoff period.

The minimum backoff period is calculated as the smallest value of backoff timers generated by the backlogged stations. The DEC_BACKOFF can generate DEFER event if the station backoff timer is greater than the minimum, which means this station will not win in this trial and the medium will be utilized by some other station. However, if the backoff timer of this station is equal to the minimum backoff value, the TRANSMIT_FRAME event is created in next time slot (this is what +1 means). Of course, a collision will occur when we have more than one station which their backoff timers are equal to the minimum value.

The TRANSMIT_FRAME event generates the FREE_MEDIUM event after the transmission completes (at Fin). It also generates either TRANSMIT_ACK event in case there is no collision or NO_ACK in case of collision. The TRANSMIT_ACK event generates FREE_MEDIUM event at acknowledgment transmission completion time (at Fin), and it generates RECV_ACK event also at Fin time without any condition. This means implicitly that the acknowledgement transmission is reliable and collision of ack with other transmission or ack loss can not happen. RECV_ACK and NO_ACK events do not generate any other event.
In order to continue transmitting packets while the station does not have an empty queue, we need to generate START_CSMA event from TRANSMIT_FRAME event in case there are more packets to transmit after the current one.

As we know from the standard [1], the backlogged station should not decrement its backoff timer if the medium is sensed busy. However, in the shown diagram, this may happen. This is because we are not sensing the medium continuously as this will impose high overhead on the simulator especially if the number of stations is high. In order to avoid such a situation, when the medium becomes busy as a result of processing TRANSMIT_FRAME or TRANSMIT_ACK event, the function "updateAffectedEvents" is called. This function will handle this issue. Fro example, if the station has a DEC_BACKOFF event, it will convert it to STOP_BACKOFF event. Here the usage of STOP_BACKOFF event can be seen. The STOP_BACKOFF event then generates DEFER event to wait until the medium becomes idle again. Actually, this function "updateAffectedEvents" does many similar things and the conversion of DEC_BACKOFF to STOP_BAKCOFF is one of them.

5 Implementation

The simulator has been implemented in MATLAB. We used the events diagram shown earlier. MATLAB is a very powerful tool that can be used to write programs and plot the results using the many built-in functions. An ordered list of all the current events is maintained. The events are sorted according to the time. When a new event is created, it should be added to the sorted list in the right position to keep the list sorted. So, the simulator always processes the event(s) that are on the head of the list. These events are the ones that will fire first. If we look carefully at the events diagram shown earlier, we can see that each station should have at most one of the following events: START_CSMA, DEFER, START_BACKOFF, DEC_BACKOFF, STOP_BACKOFF, and TRANSMIT_FRAME. Let us call these events transmit events. Each station, according to its state, can have at most one of the transmit events. This is an intuitive conclusion. One station can not be transmitting a frame (TRANSMIT_FRAME) while it is deferring its transmission (DEFER).

In order to implement the function "updateAffectedEvents", we have to search the whole events list for transmit events. Then, we have to update these events times and/or types depending on the event type. This is inefficient especially if we have a very large events list. Since, we are only interested in updating the transmit events, we can keep a separate list of indices to the events list that corresponds to each transmit event in it. Let us call this new list transmitEvent. The transmitEvent vector will have a length equal to the number of stations. If the station is currently idle (does not have a packet to transmit), the corresponding element in transmitEvent will be set to zero. Otherwise, it will have the index value of the corresponding element in the events list.

Now, we need to search only the transmitEvent vector for the non-zero elements to update what is needed.

To run the simulator, we have two main MATLAB scripts. One is Main.m which is command line-based. The other one is MainGUI.m which uses graphical user interface. The Main.m script is not recommended to use as it does not provide the capabilities provided in MainGUI.m. When we started our implementation, we used it for running and testing the simulator. However, it can be ignored after providing the GUI.

5.1 Scheduling Scenarios

Usually, the user does not simulate only one scenario. But rather, he simulates many different scenarios to compare them to evaluate the protocol performance. For example, one scenario could be to set packet size to 300 bytes, another could be for 500 bytes, and so on. By fixing all other parameters, the user can check the effect of changing the packet size on the available performance measures under these specific conditions. Then, different set of scenarios can be run to compare their performance and come up with conclusions. In order to be able to schedule the running of multiple simulations, multiple scenarios can be created and run. Each scenario has a name and a complete set of simulation parameters. The simulation can be terminated by specifying the maximum number events to process or by specifying the maximum time in minutes.

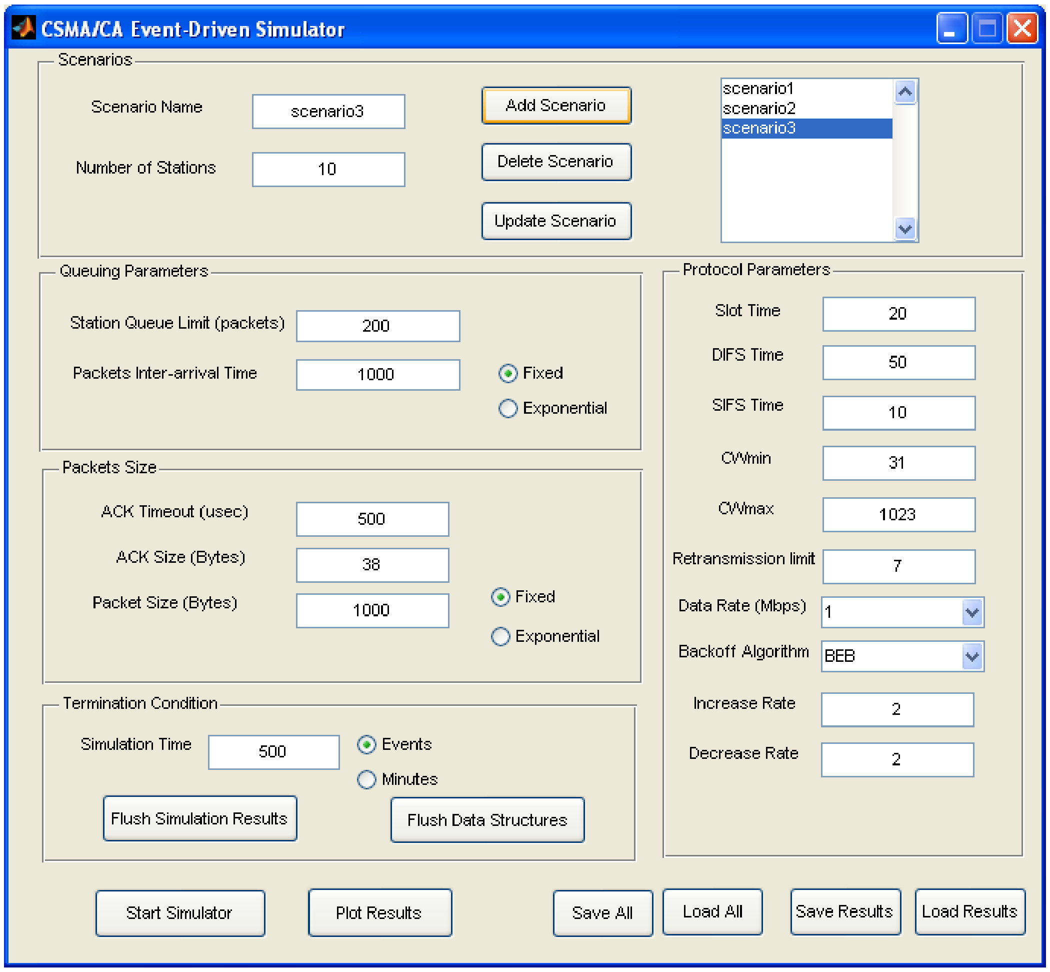
The simulation data structures (events list, stations states, medium, and others) and results (collected statistics) can be saved on an XML file. We used the toolbox xml_toolbox-3.1.2 to save and load XML files. Two functions were used to save/load MATLAB data structures. The function "xml_save" is used to save the data structure given as argument to the specified file name. The function "xml_load" is used to load the data structure saved in the specified file name. The user can run the simulation for some time or for a specific number of events. Then, he can save the results and quit the simulator. After that, he can run the simulator again for some other period of time. He can keep the simulation results or flush them to start a new simulation. After running the simulator, the user can save the results and plot them using the same simulator. A window is provided for the user for plotting the collected statistics versus some criteria chosen by the user. For example, the user can plot the throughput against the number of stations. Each scenario represents a point in the plot. After plotting the results, the user can decide to resume the simulation to get better plots.

5.2 MATLAB Implementation issues

MATLAB is a single-threaded program and it can not create other threads to parallelize the developed simulator. So, if the user presses a button on the GUI, he has to wait until the corresponding function terminates. During this period, the GUI becomes unresponsive. This is due to the fact that MATLAB uses only single thread to manage all the operations. An option to solve this issue and make the GUI always responsive is to create another thread that does the requested operation. This can not be done using MATLAB only. A Java thread has to be created for this purpose, and this thread should be created from MATLAB. Then, the created Java thread will call some MATLAB function to perform the request task. In this case, the GUI is released directly after pressing a button and enables the user to use it again for other commands. This issue has not been resolved in this work, because of time limitation. The main focus of this work is to get a running simulator that gives correct results. Of course, developing a simulator from scratch, debugging it, and verifying the results require a lot of time and efforts.

5.3 Simulator Snapshots

The main GUI of the simulator is shown in Figure 3. Three scenarios have been created in this example.



Scenario Name: scenario3
Number of Stations: 10
scenario1
scenario2
scenario3

Station Queue Limit (packets): 200
Packets Inter-arrival Time: 1000 [✓ Fixed] Exponential

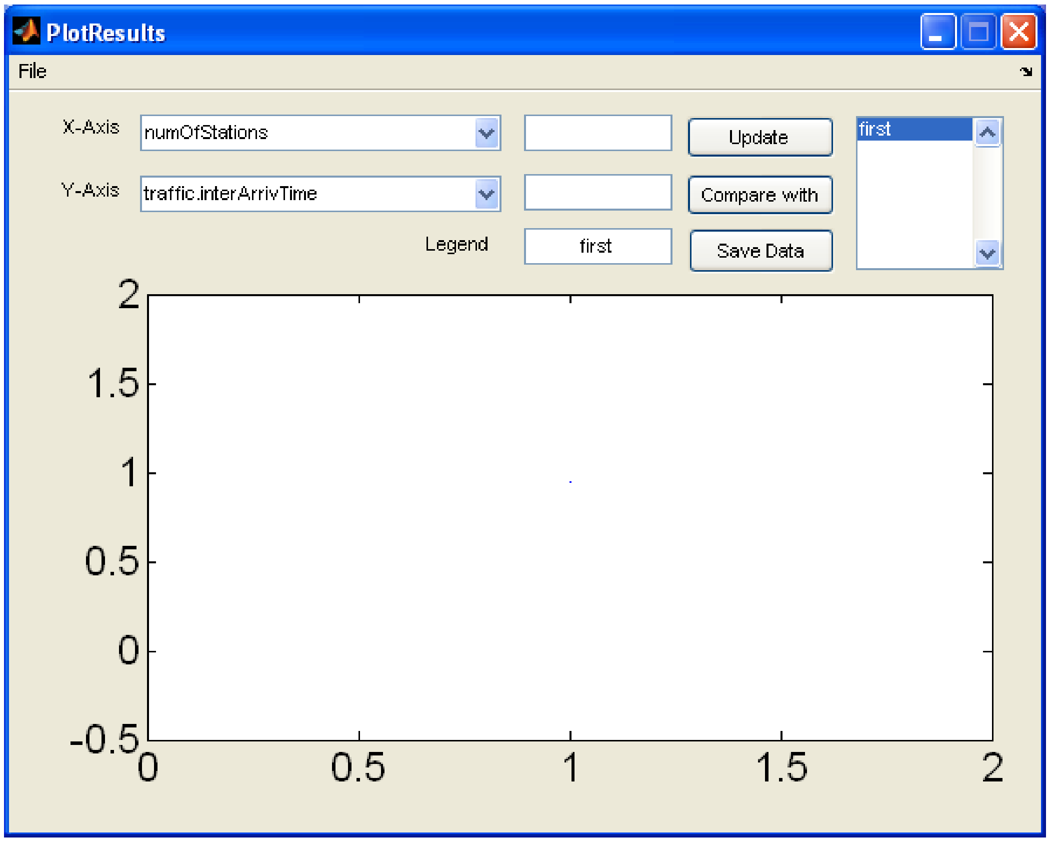
ACK Timeout (usec): 500
ACK Size (Bytes): 38
Packet Size (Bytes): 1000 [✓ Fixed] Exponential

Simulation Time: 500 [✓ Events] Minutes

Slot Time: 20
DIFS Time: 50
SIFS Time: 10
CWmin: 31
CWmax: 1023
Retransmission limit: 7
Data Rate (Mbps): 1
Backoff Algorithm: BEB
Increase Rate: 2
Decrease Rate: 2

Figure 3. CSMA/CA Simulator Main GUI snapshot

After adding the required scenarios, the simulator can be started by pressing 'Start Simulator' button. When finished, you can save your work by pressing 'Save All' button. This will save all the variables used in the simulation which requires relatively a big amount of storage. In order to plot the results, 'Save Results' button should be pressed to save the collected statistics on an XML file. Then, the 'Plot Results' button can be pressed to show the plotting dialog which is shown in Figure 4. This dialog plots the results loaded from the XML file. You can choose the x-axis and y-axis values that you want from the drop-down menu. You can enter the values for x-label and y-label. You can save the plotted data to a text file in two-column format. Furthermore, previous plots can be compared by pressing 'Compare with' button. The user will be prompted to enter the name of the XML file to load. A new item will be added to the list of scenarios' sets shown on the right. The legend value will take the file name by default, but it can be changed from the corresponding text field.



X-Axis numOfStations

Y-Axis traffic.interArrivTime

Legend first

first

Figure 4. Plot Results Dialog snapshot

Whenever a change is made to the previously described fields in this dialog, the 'update' button should be pressed to update the plotted figure.

6 Simulation Results

6.1 Simulation Parameters

The used simulation parameters are similar to those found in [1][2]. The following table summarizes them:

Parameter	**Value**
ACK Size	38 bytes
Data Rate	1Mbps
CWmin	31
CWmax	1023
Packets Inter-arrival Time	2 msec
Propagation Delay	0

Slot Time	20 microseconds
SIFS	10 microseconds
DIFS	50 microseconds
Retry Limit	7
ACK Timeout	500 microseconds after sending the frame completely
Station Queue Length	300 packets

In our simulation, the propagation delay is assumed to be negligible. The packet size is varied from one simulation to another. The packet size is exponentially distributed. The inter-arrival time is exponentially distributed with mean 2000 microseconds

6.2 Performance Metrics

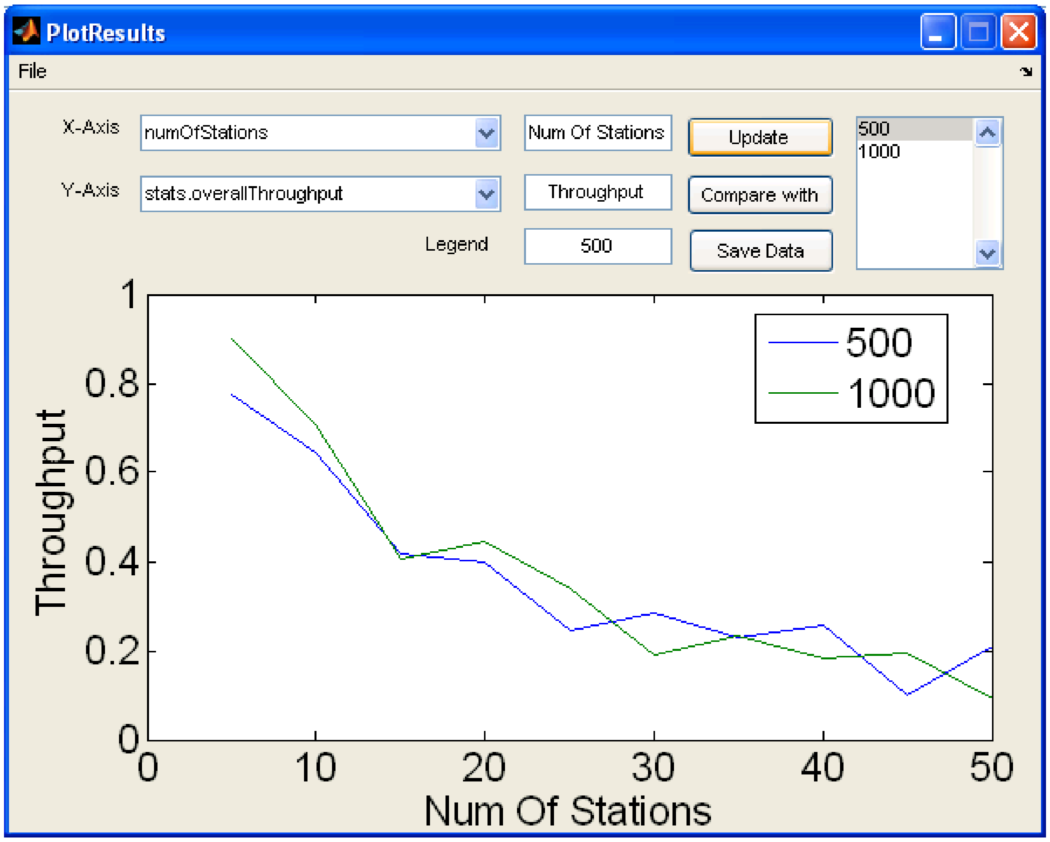
The main performance metrics that were used are summarized in the following list:

- LAN Throughput
- Collision Ratio
- Average Queue Length
- Average Queuing Time

The LAN throughput can be calculated by dividing the overall time spent in successful transmissions by the overall simulation time. The collision ratio is the number of collisions divided by the overall number of transmissions (including those that are collided). The average queue length is the average length of the stations queues. The average queue length for a station is time-average value. The average queuing time is the sum of the waiting time for all sent packets divided by the number of transmissions averaged for all stations.

6.3 Results & Analysis

The simulation parameters were set to the values described in section 6.1. The plot shown in Figure 5 shows the LAN throughput versus the number of stations for packet sizes of 500, and 1000 bytes. The packet size is exponentially distributed with the mean values indicated. We can see that the throughput is decreases as the number of stations increases. This is due to the increase in the number of collisions which is caused by the increased contention to access the medium. As the number of stations increases, the load on the network also increases which causes higher number of collisions. This figure of throughput is similar to the one shown in [2] for IEEE802.11. There is some difference between the two plots. This may be due to some variation in the simulation parameters. For example, that work did not state the value used for the acknowledgment timeout. Two curves are shown: one with label 500 and the other one with label 1000. These two numbers represent the packet size.



PlotResults

X-Axis numOfStations Num Of Stations

Y-Axis stats.overallThroughput Throughput

Legend 500

500
1000

Figure 5. LAN Throughput vs. Number of stations

We can say that for bigger packet size we have higher throughput. This is not so clear in the shown figure because we have some fluctuations in the plots. However, we can generalize this conclusion before studying other possible values for the packet size.

The following figure shows the collision ratio versus the number of stations. This plot explains the previous. For the same reason, as the number of stations increases, the number of collisions increases. Of course, higher collision ratio means lower throughput.

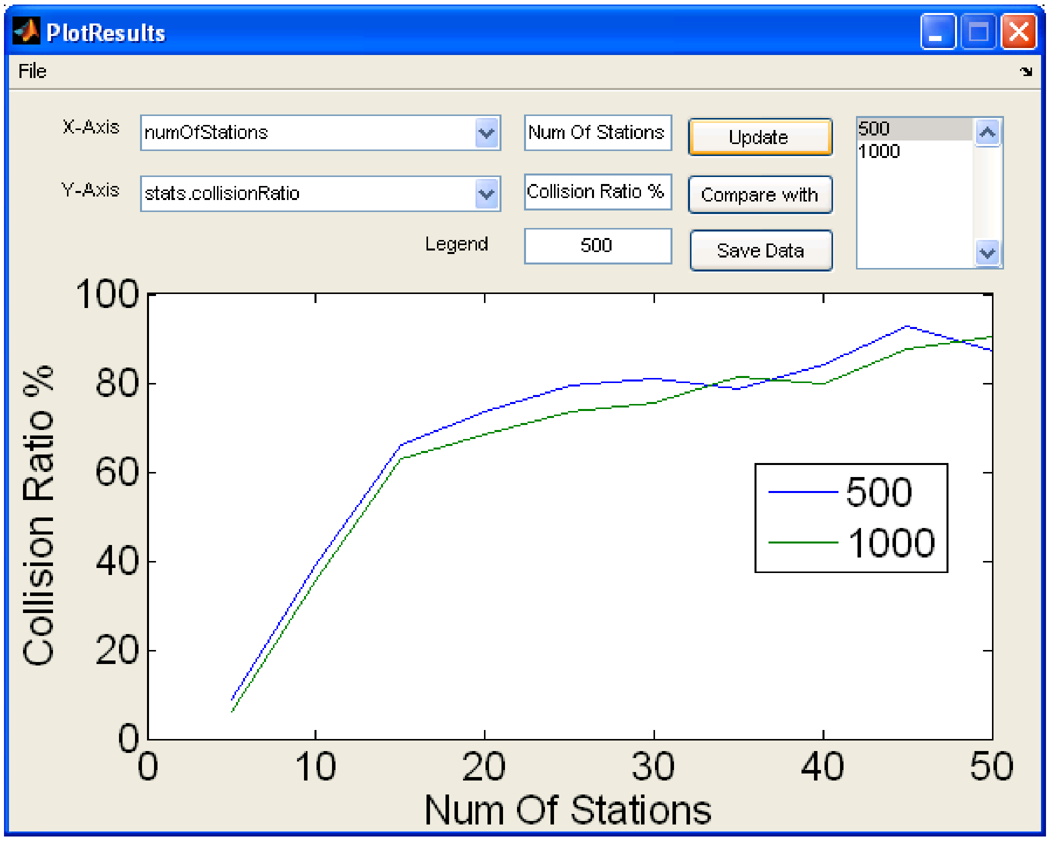
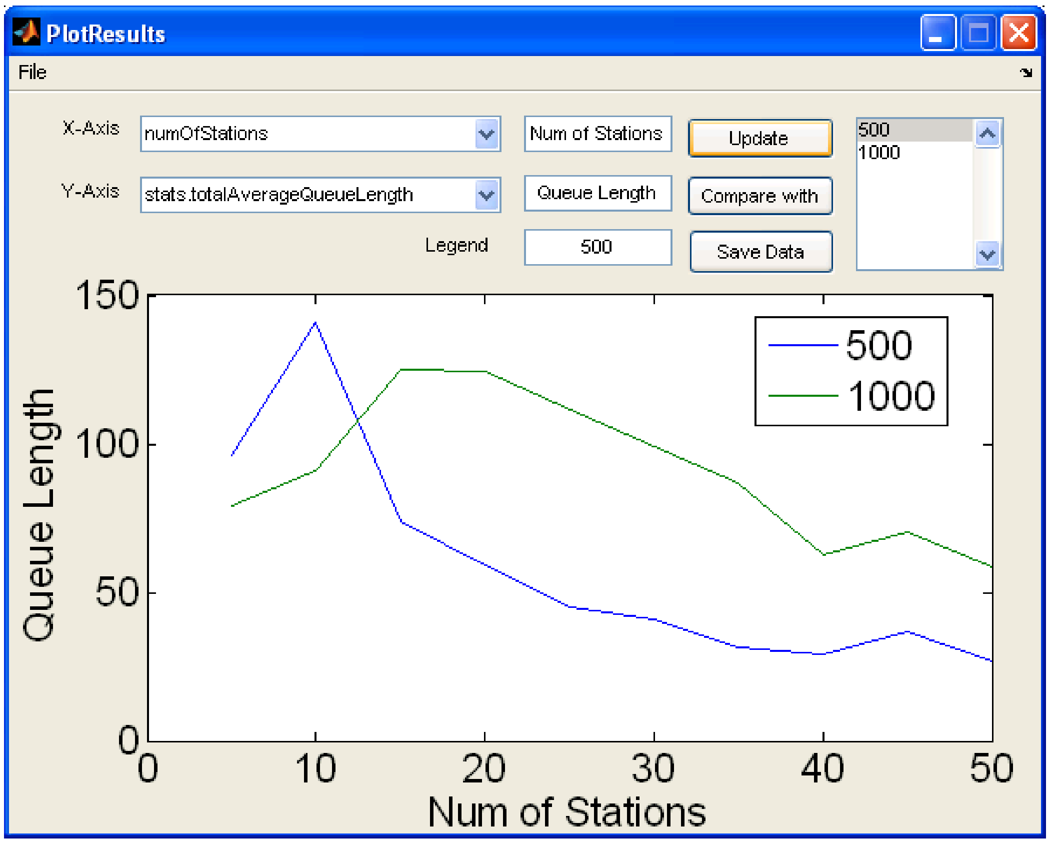


Figure 6. Collision Ratio vs. Number of stations

The average queue length is plotted in the next figure. For bigger packet size, we have larger queues. This is due to the fact that larger packets take longer time to be transmitted. So, the queue will keep the packets for more time. This will cause larger queues for the stations.



X-Axis numOfStations Num of Stations

Y-Axis stats.totalAverageQueueLength Queue Length

Legend 500

Figure 7. Average queue length vs. Number of stations

The average queuing time versus the number of stations is shown in the following figure. Similarly to the previous plot, the queuing time increases when we increase the packet size. Larger packets stay longer in the queue which means higher delay.

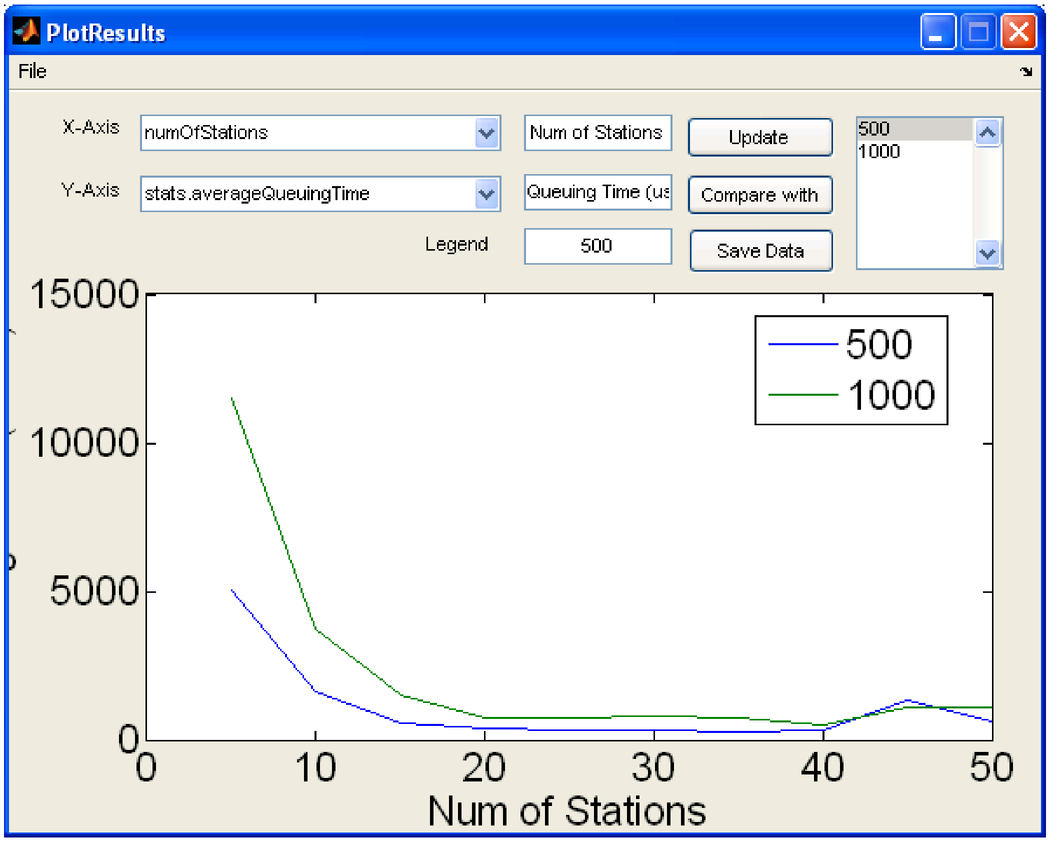
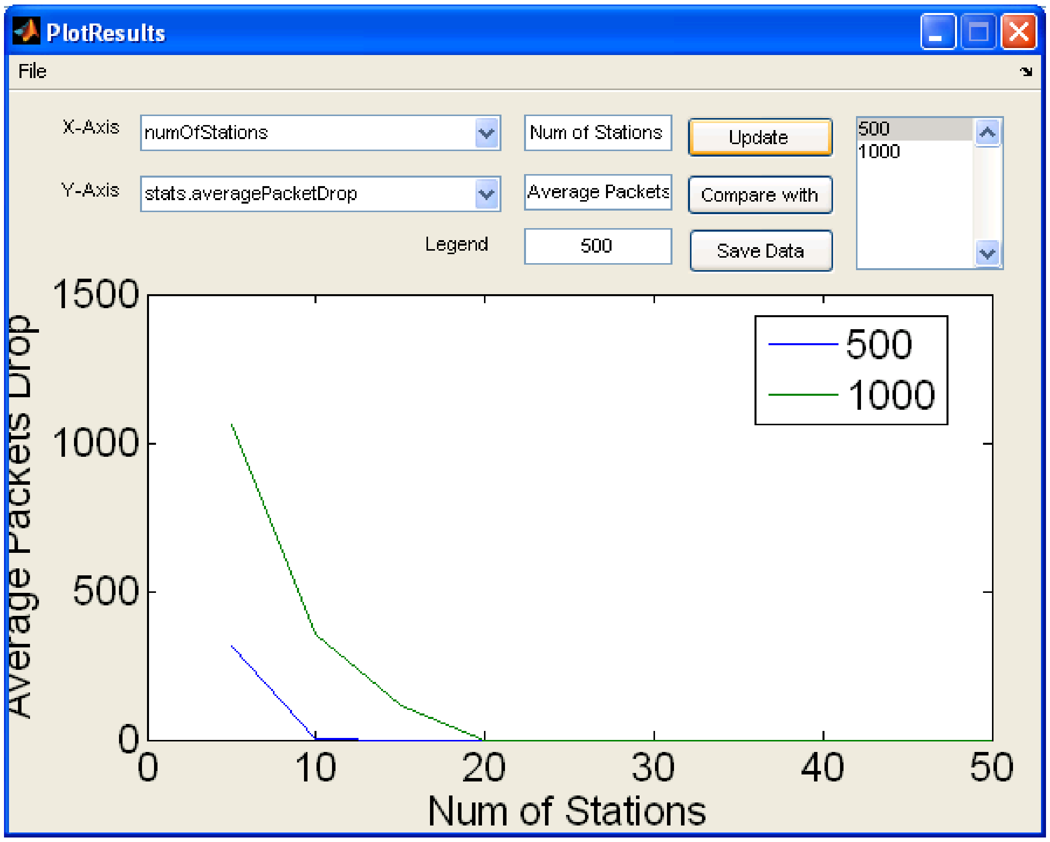


Figure 8. Average queuing time vs. Number of stations

The next figure shows the packets' drops. As we have larger queues and higher delays, we have larger number of packets dropped.



numOfStations

stats.averagePacketDrop

Num of Stations

Average Packets

500

Figure 9. Average packets drop vs. Number of stations

7 Simulator Capabilities

In summary, the simulator enables the user to do the following:

- Add / Change / Delete scenarios.
- Configure many protocol parameters and traffic characteristics.
- Choose the backoff algorithm.
- Save the simulation results.
- Resume the simulation after loading the last work.
- Plotting the results and comparing different set of scenarios.
- All of the above features are available through single comprehensive GUI.

8 Conclusions

An event-driven simulator can be efficient if it is programmed carefully. It can be even made faster if it is exploiting multithreading capability in the programming language used for implementation. Simulators are useful tools to evaluate the performance of the existing protocols and the proposed ones. They facilitate this without the need to actually implementing the protocol to be evaluated, and having a real setup for this purpose.

Larger packets may give better performance in terms of throughput. However, it may cause more delays to other waiting packets and consequently higher number of dropped packets due to limited queuing.

9 Future Work

The simulator can be enhanced more to be a flexible simulator and able to simulate not only CSMA/CA but also simulating other protocols like CSMA/CD. Also, more parameters and algorithms can be included in the simulator in the purpose of testing them and verifying the results. Someone can study the possibility of creating a multithreaded simulator that is also event-driven. This simulator can run faster than the current one by exploiting the available SMP (Symmetric Multi Processing) architecture.

10 References

[1] IEEE Std 802.11, 1999 Edition, Information technology – elecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements. Part 11: MAC and PHY specifications.

[2] A. Al-Akeel, "Optimizing Backoff Procedure for Enhanced Throughput and Fairness in Wireless LANs", MS Thesis, King Fahd University of Petroleum & Minerals 2007.

[3] Vukovic, I.N.; Smavatkul, N., “Saturation throughput analysis of different backoff algorithms in IEEE802.11,” Personal, Indoor and Mobile Radio Communications, 2004. PIMRC 2004. 15th IEEE International Symposium on , vol.3, no., pp. 1870-1875 Vol.3, 5-8 Sept. 2004

[4] G. Bianchi, “Performance Analysis of The IEEE802.11 Distributed Coordination Function”, IEEE Journal on Selected Areas in Communications, pp. 535-547, Vol. 18, March 2000.

[5] Wen-Kuang Kuo and C.-C. Jay Kuo, "Enhanced Backoff Scheme in CSMA/CA for IEEE 802.11", 2003.

[6] F.Cali, M.Conti, E.Gregori, “Dynamic Tuning of the IEEE 802.11 Protocol to Achieve a Theoretical Throughput Limit”, IEEE/ACM Trans. On Networking, V8, N6, pp.785-799, Dec. 2000.

[7] Yunli Chen, Qing-An Zeng, and Dharma P. Agrawal, "Performance Analysis and Enhancement for IEEE 802.11 MAC Protocol", 2003.

[8] Giuseppe Bianchi, Luigi Fsatta, Matteo Oliveri, "Performance Evaluation and Enhancement of the CSMA/CA MAC Protocol for 802.11 Wireless LANs", 1996.