Assignment #1
Evaluation of the performance of CSMA-CD

Jay Kraut

Introduction

CSMA-CD (Carrier sensing multiple access – Collision detection) is a protocol for multiple devices to share a single channel. It is the protocol used for the physical layer of 802.3 LAN connections when multiple devices are connected to the same transmission medium rather then hooked up to a hub or router.

The protocol is best explained by described its operation by example. When a device has a packet to transmit it senses the transmission channel and waits until the channel is idle. When the channel becomes idle the device starts transmitting the packet on the channel. Once a device starts transmitting if it detects that there is no other device attempting to transmit within two propagation delays it knows it has control of the channel and continues to transmit. After two propagation delays it is guaranteed that all other devices on the network realize the device is transmitting and won't interfere. However, if another device starts transmitting before it detects the first device transmissions (this is possible within one propagation delay) there will be a collision. At this point both devices are listening to the channel and realize the data on the channel is not the same as they are transmitting so both devices realize that some other device is attempting to transmit. As soon as this occurs both devices transmit a short jamming signal and cease to transmit. The devices then set a back off time to wait before attempting to transmit again. How this back off time is generated is crucial to the proper operation to the network. If both devices are configured to immediately begin retransmitting right after a collision there will never be a time when both devices will be able to transmit. However, if the back off time is too large the network will be inefficient.

There are several attributes that a CDMA-CD network has. The first is maximum bit rate. The maximum bit rate is the total amount of bits that can be transmitted on a channel per a time period. This is the ideal rate and is not achievable if more then one device is sharing a channel. Another important attribute is the minislot time. This time is specified as twice the propagation delay. The minislot time determines that smallest packet size. If a packet is smaller then the minislot time there is a possibility that even if there is a collision the transmitting device will assume the transmission was successful

because it won't be able to detect the collision. Lastly the network has is maximum packet size and the back off algorithm both of which can be changed to maximize efficiency.

To evaluate CDMA-CD a simulation is set up to mimic a standard 10 mBits/s 802.3 LAN. The maximum bit rate is specified at 10Mbits/Seconds. The minislot time is 52uS, the maximum packet size is 1518 and the minimum packet size is 64. The back off algorithm used is that the number of minislots that the device backs off after a collision is determined by taking a random number between 0 and $2^k - 1$ where k is the number of retransmitting attempts and is limited to a maximum of 10.

The results of running the simulator are statistical graphs that show the performance. The relevant statistics generated are average waiting time to send a packet, number of packets sent successfully vs dropped and total bandwidth efficiency.

The kind of network traffic simulated is bursty UDP traffic. This traffic simulates the kind of traffic that video streaming servers generate. The main focus of the experimentation of this report is on evaluating the efficiency of the network that is completely loaded. This is important in video streaming that requires low latency when streaming video in real time. In particular the efficiency of the back off algorithm that is used in 802.3 is evaluated and different back off algorithms are tested and compared to the standard.

Simulator architecture

For this assignment the decision was made to program a simulator from scratch. The goal of the simulation is that it is accurate in terms of the various statistics and to be robust for future additions to be added on. The architecture is designed using object oriented programming with each component of the network being one class. This also allows for future modifications at which different components can be swapped in and out for different types of networks. The simulator is ran as a single thread and uses its own clock for timing. Although the simulation is coded is visual c+ for windows the only windows specific elements are the user interface and the single thread that is used to run the simulation.

To be realistic the network is run in "real-time" fashion. The physicals layer clock (at 10 mbits) is the clock used to time the network. This does not mean that the simulator has a loop that loops 10 million times for each second of execution. To speed things up the simulation uses an event driven architecture. This means that rather then simulating every fraction of a second the main loop uses event based execution. An example of this is when transmitting a packet. Once it is verified that the device can transmit successfully, the simulation clock is simply advanced the length of the packet transmission. The only problem with designing the simulator this way is how to have processes run asynchronously.

Each process on the network is run at a fraction of the main clock, at intervals of approximately 1ms. The physical layer loop runs a large multiple of the main clock before letting the processes run to perform any timing based operations. This can potentially lead to a large problem. In a real network all the devices are operating on their own clock thus they are completely independent. With all of the process on the same clock, synchronization could occur that could invalidate the simulation. An example of this occurring is if there are several servers on the network that transmit several packets a second. If the servers are all synchronized far more collision occur if they were not synchronized. In order to avoid this situation a certain amount of randomness is introduced. When queuing up a packet in the NIC layer, the NIC could add a random amount of time that is smaller then the interval between which process are

ran. This means that although all of the processes are running on the same clock they are all randomly out of phase with each other.

The simulator is designed as a virtual network meaning it allows for complete client server applications that can be ran as separate processes and actually send data through the network. This is illustrated by the architecture of the network as shown by figure 1. The network class holds all of the process classes which represent network applications. Each process has one NIC/OS class to link the process to the network. The NIC/OS class has several familiar socket based functions. An example is the "create" and "send" function calls. Currently only broadcast sockets are implemented. In future connection sockets and full event handling will be implemented. This will mean any network application, such as an ftp will actually work correctly over the network and will actual be able to transfer data.

Another feature of the simulator is that the primary classes such as the process class and physical layer class are implemented as base classes with virtual functions. This makes it much easier to change the physical layer and to quickly add different kinds of processes to the network. The goal of the simulator is to simulate many kinds of networks with different standards such as 802.3 and 802.11 without changing too much code.

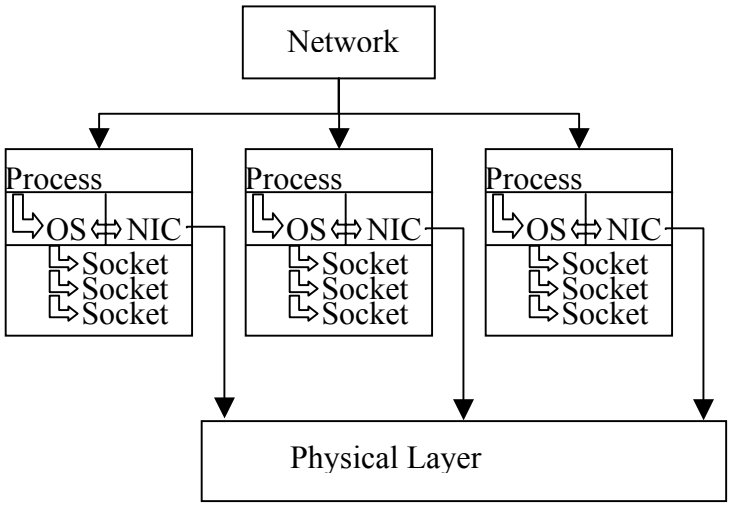

Figure 1. Simulator Architecture

The network generates several statistics that show the performance of the network as show by the GUI, figure 2. Starting on the top the simulator shows the number of seconds that have elapsed since the start of the simulation. Then it shows the maximum bit rate and the actual bit rate in terms of bits used to send packets. Idle bit rate is the bit rate that is left and not being used for either sending data or being wasted by the collision detection. The efficiency percentage is defined as the number of bits used for packets divided by number of bits used for both packets and collisions. The graph window then displays the efficiency, with the y axis being efficiency from 0-100% and the x axis being time and is scrolled in time. Each process generates its own statistics. The avg wait time is calculated as the time from when the processes NIC has a packet available to send to the time it can send it. There were two other choices such as counting the time the data is sent to the NIC to be packatized and then queued, and counting the time from when the data is available to when the data completely reached the receiver. However, since only the physical layer protocol is being evaluated using the wait time calculation used makes more sense. Average resend time is the average amount number of times each packet has a collision when trying to get sent and has to be resent. There are two ways packets are not sent. There is the possibility that the packet cannot get access time on the physical layer and exceed 16 resend attempts. In this case the packet is considered dropped. There is also the case that under extreme network traffic the buffer on the NIC can no longer hold any more data and it simply ignores the socket send calls. In this case the packet is classified as missed. In both of these cases, these packets are not used for the calculation of average wait time or average resend time.

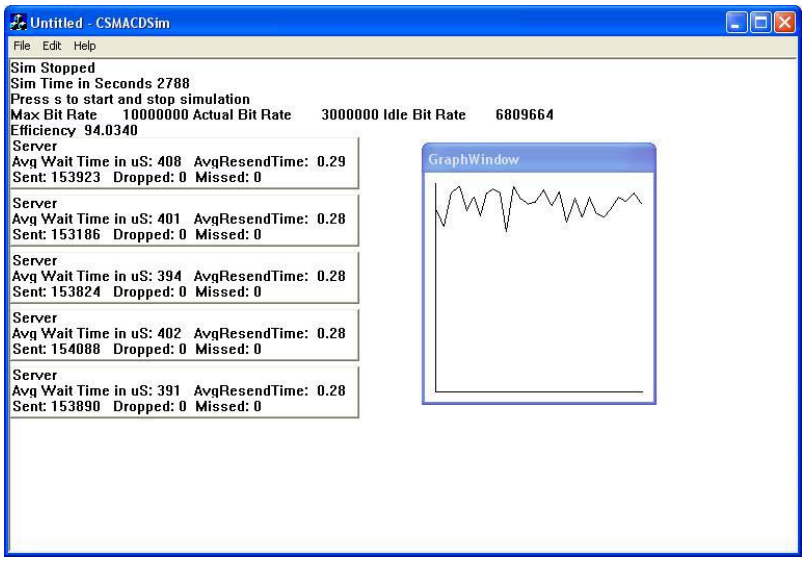

Figure 2. GUI of the simulator

Experimentation

The first test on the simulator was to verify that it is working correctly. The network was configured with two servers that send 15000 bytes approximately 5 times a second at random intervals. The bit rate reported is on average 120000 which is correct (15000*8*5*2). The avg wait time is 74us and avg resend time at 0.07. The efficiency bounces from 100-85% showing occasionally there are a few collisions but most times both servers get to send their data immediately. These values seem reasonable for a network under very little loading

The next test is when the network has been loaded to the point where data just is not getting through. As expected there is no idle bits available because the network is loaded to 100% of bit rate. Under these conditions avg wait time is 110ms which is extremely high. But average resend is surprisingly small at around 1.35. This will be explored later. The actual bit rate is around 9.3 mBits. After around 558 seconds of simulation each server sent around 19000 packets successfully however each had around 2500 dropped packets and 8000 missed packets. Efficiency was measures at 93% and stayed very consistent in the simulation.

The results of the second experiment led to some interesting questions. Why is the efficiency so high and the resend rate so low for a completely loaded network. To find this out further tests are conducted. These test measure the statistics of the network vs. number of servers (traffic). The tests are conducted three times, once for each back off method. One of the methods used is the one used for 802.3 lans at which back off time is determined in terms of minislots at $2^k - 1$ where k = min(n,10). Another is a simple modification that k = min(n,5). The other back off algorithm used is just a random amount of minislots between 0 and 10.

Out of all of the statistics gather from the tests, five of them are shown in graph form. Figure 7-9 shows the percentage of packets sent in comparison to the ones not sent. These graphs show at what point in terms of amount of servers the network becomes congested and packets started getting dropped at large quantities. Figure 3 shows the efficiency of each back off algorithm is comparison to each other. Notice that the 2^k 0-10 algorithm has a dip when the network traffic starts to become heavy and

recovers at the point at which the network becomes 100% utilized. Figure 5 correlates to figure 3 in that at some point the bit rate of 2^k 0-10 is less then the other two algorithms and then recovers after the network becomes loaded. This is also show in figure 6. Remember that wait time only counts for packets that are sent not for ones that are dropped. Figure 4 shows something that might not be as obvious. The resend times or 2^k 0-10 is less then 2^k 0-5 which is less then 0-10 random.

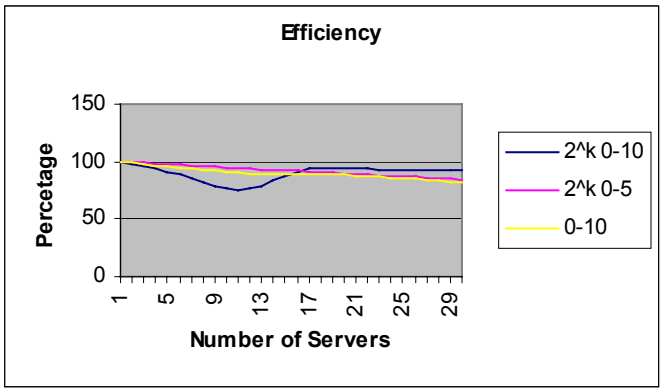

Figure 3.

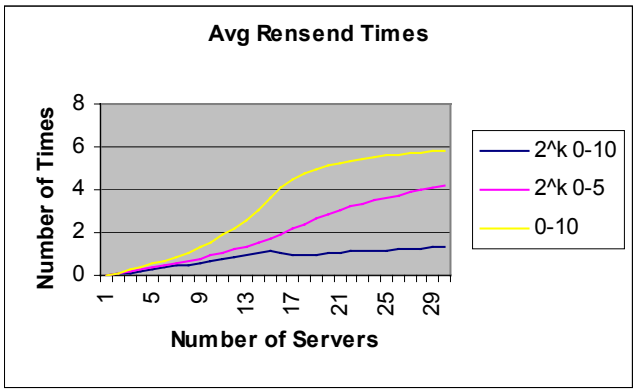

Figure 4.

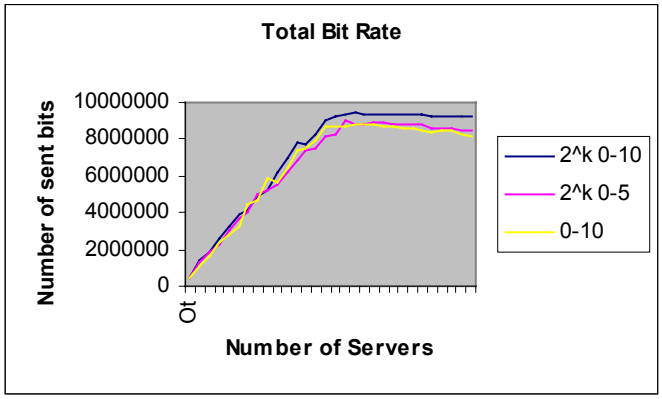

Figure 5.

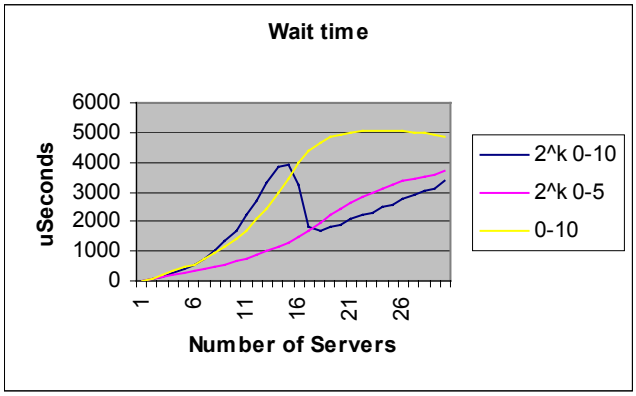

Figure 6.

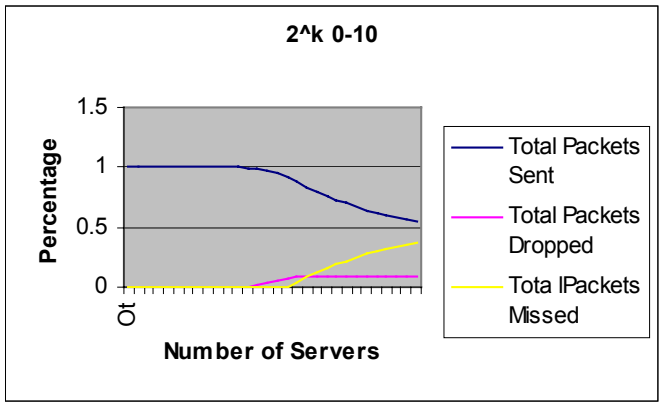

Figure 7.

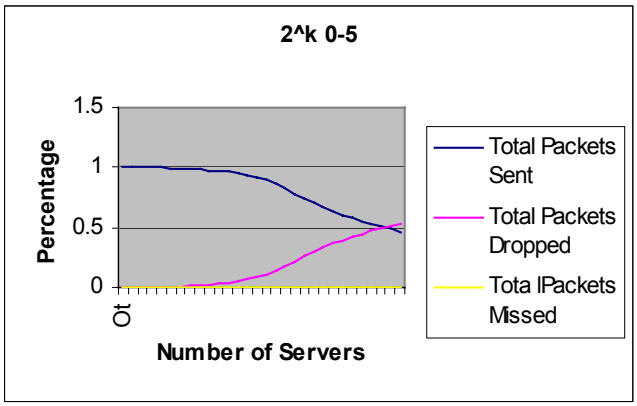

Figure 8.

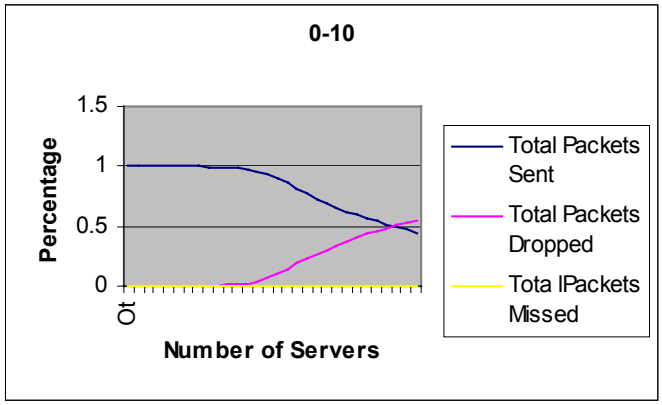

Figure 9.

Discussion

A good question is why is there a dip in 2^k 0-10 performance when the network becomes loaded. The reason is shown in figure 4 and in figure 7 and is not obvious. When the network starts to become loaded the resend times go up. At around 75% loading a packet that is resent for the 2,3,4 time etc still has a chance to get sent. However once the network becomes loaded the likelihood that a packet after the first few tries get sent is very low. In figure 4 when the network gets loaded under 2^k 0-10 the average resend rate is only around 1.5. This means that after this point it becomes likely that the packet is never sent. As a consequence of this and the long back off time the NIC buffer fills up and packets start getting missed rather then dropped because the drop time is rather large.

Looking at the bigger picture, when the network becomes very congested the probability of successfully sending a packet gets lower. Also due to the 2^k 0-10 algorithm large back off time, a packet has a very small chance of getting sent after the first few resend attempts. When this happens, rather then interfering with future idle time attempts to send, the large back off time essentially stops the packet from interfering too much. This is why 2^k 0-10 has the greatest efficiency except for a small dip. This dip is when there is just enough idle time for the larger back off times to involve the packets in collisions. However after the network gets loaded, back off times get even larger which means fewer packets competing for collision event thus greater over all efficiency.

Conclusion

This report evaluates the efficiency of CSMA-CD by looking at the back off algorithm. The 2^k 0-10 algorithm is the most efficient when the network is completely loaded. In addition to the analysis done in this report there are probably other reasons that the 2^k 0-10 algorithm was chosen as the back off algorithm. However on a pure physical layer analysis over the range of different network loads, it on average has greater bit rate.

The experimentation also explains partly how to make a CSMA-CD more efficient. It is key to reduce the amount of devices that at any given idle time want to send data. This reduces the amount of time taken up resolving collisions.