24.765 Multiple Channels Experimentation

Jay Kraut

Abstract:

Sometimes it is not possible to further increase the throughput of a channel using optimization techniques. At this point one way to increase the throughput while not changing the channel standard is to simply use multiple channels to transmit. This paper examines performance issues of the selection algorithms to choose which channel to transmit on, and what happens when the bit error rate is different on one channel.

Keywords:

802.11b, Multiple Channel Selection Algorithms. Bit error rate affects.

1. Introduction

Given a transmission frequency and a protocol standard there is a limit to the maximum bit rate a channel could support. When it is not possible to increase the throughput further using optimization techniques it is still possible to increase the overall throughput if more then one channel is used. This report uses a simulator to look at implementation and performance issues of using more then one channel per device to increasing the maximum bit rate.

2. Simulator implementation

In order to simulate the multi-channel devices the simulator from assignment #2 is modified. As of assignment #2 the simulator architecture is as shown in figure 1.

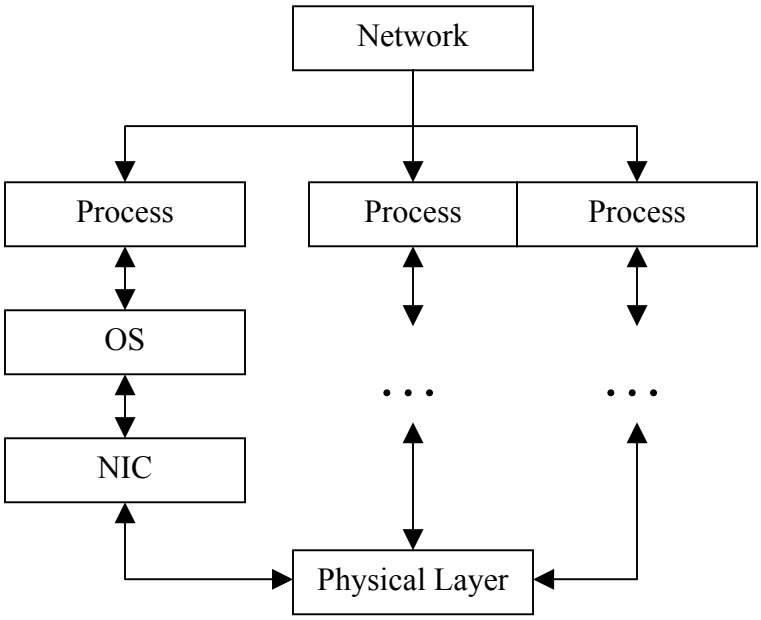

Figure 1.

The question is, how to modify the simulator to support multiple channels? This can be done by creating multiple instances of the Physical Layer and NICs. This is a relatively simply modification because of the fact that the simulator is implemented using object oriented programming. Each block on figure 1 is its own class. The architecture of the modified simulator is shown in figure 2. For each channel

(number of which is modifiable to any number) a Physical Layer class is created. Each OS now creates one NIC class per channel and assigns each NIC to its respective channel.

In order to send out packets there has to be a way to choose a channel for each packet. There are several ways to do this. One of them is on creation of a socket specify which channel to use. This would work well if a dedicated channel is required for a certain function. An example of this may be to maintain a certain QoS (Quality of Service) for a video stream. A second way is to simply let each socket have access to all of the channels and route each packet to a channel depending on the traffic volume per channel. This is the way that is implemented. When the OS wants to send a packet is checks each NIC to see which one has the smallest outgoing buffer size. The one with the smallest outgoing buffer is the one used to send the data.

To receive data no change is required. The OS and each NIC maintain separate buffers. When the NIC has data to send up to the OS it simply gives the data to the OS to copy into its buffer. It doesn't matter which channel is used to send because for the OS it doesn't care. The only possible problem is with message fragmentation. If sending data in consecutive packets it is possible that they now may appear out of order. It is possible to resolve this problem in the application layer by including a packet number that can be used to reorder the packets. Fortunately the OS has this internally implemented simply for convenience and no changes are required.

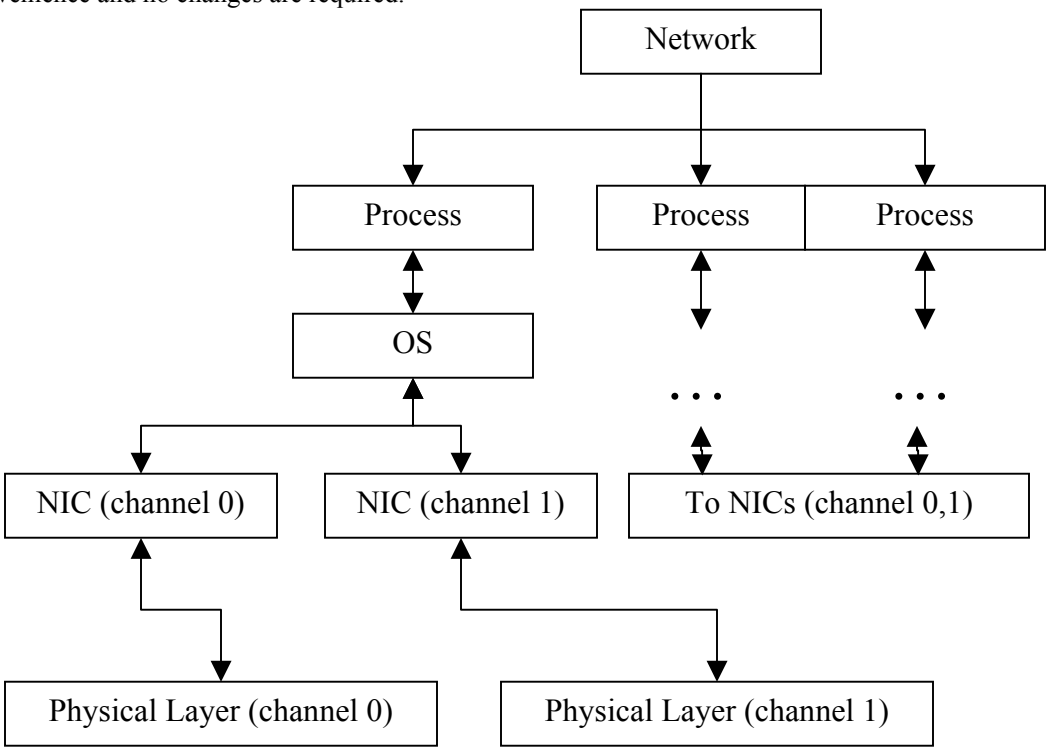

Figure 2.

In summary a simple modification to the lower levels of the simulator architecture is the only thing required to add multiple channels to the simulator. Only the OS class had major modifications in the fact that it must include and then choose between multiple NICs to send a packet. Other then that all of the upper level hierarchy is not change and is oblivious to the fact that multiple channels now can be used.

3. Verification

In order to verify that the modified simulator works two things must be checked. First is the data integrity must be checked. That is if data is sent through the simulated network it must be received without error. This is done with a mock FTP program that sends over a file in variable size data sizes to check for

fragmentation errors. The modified simulator using two channels correctly sent over a jpg image and reconstructed it correctly.

The other thing that must be checked is the fact that having two channels doubles the maximum bit rate of only having one channel. This is tested and the bit rate is effectively double.

4. Experimentation

Theoretically adding a second channel simply doubles the maximum bit rate. Instinctively if only one device is using the channels there should not be any draw backs to using two channels. However under certain conditions having more then one channel can lead to inefficiencies. The experimentation examines some performance issues with having multiple channels with multiple servers and having multiple channels with different BER (bit error rates).

4.1 Channel Selection Algorithms

The first performance issues is, what happens when multiple severs have access to multiple channels? The experimental setup used is 5 channels at 400 kbit/s each with 4 servers broadcasting through an access point at increasing amounts of bit rates with each channel having equal bit error rates. During early experimentation something odd occurred. The algorithm that was first used to pick which NIC to send a packet was very simple. It simply picked the one with the smallest buffer size at the time. If they all are empty it defaults to channel 0. Originally this was thought as the best algorithm to use, as it should equalize the load level over all of the channels, but at that time multiple servers were not considered.

The results in terms of bit rate per channel, wait time per channel and resends wait per channel are shown in figure 3-5. Using this algorithm more data goes into the first few channels then the last few. This negatively affects wait time and the resend rate. From previous experiments it was shown that for certain domain models using only deterministic algorithms when the amount of information is not complete can cause failures. After the results shown in figure 3-5 were obtained, two additional algorithms are then used to pick which channel to use. One of them at all times pick the channel randomly, and an in between algorithm that uses random numbers but weights the channel according to buffer size. Both of them perform similarly and Figure 5-7 shows the results for using random numbers to pick the channel. These algorithms have greater consistency then always picking the smallest one.

When the network load is low the only important metric is the wait time. Having an uneven distribution of data increased the wait time of one of the channels. Over the different channels the wait time evens out though. When the network load is reaching capacity having a poor selection algorithm can degrades the maximum performance. Figure 8 shows this occurring as using the smallest buffer selection algorithm degrades the maximum bit rate. At the worst point the channels bit rate is decreased by 600 Kbits out of 20 Mbits.

All of the differences between the three different channel selection algorithms are relatively minor with the worst case being a 3% difference in maximum bit rate. However, it is interesting to note that randomly picked channels is slightly more efficient then looking at the buffer size and always picking the smallest one.

4.2 BER Issues

Using multiple channels should mitigate the affects of one of the channels having a high bit error rate. Figure 9-11 show what happens when one channel has a higher bit rate then another channel. The wait time and resend rates are much higher with a higher bit rate. In terms of percentage of load each channel carries the one with the higher bit rate stays even until it has reached maximum capacity. At this point the other channel takes over. Figure 12 shows the load percentage with different bit error rates of channel 1 while keeping channel 0 BER constant. Notice that the higher the bit error rate the less load it takes to completely use up channel 0. Figure 13 shows what happens to the total bit rate available as the BER of one of the channels goes higher.

The channel selection algorithm used for the first BER test is the random channel algorithm. This algorithm performed better with multiple servers but is not ideal when each channel is not equal in terms of actual bit rate. Generally when one channel is performing poorly compared to another one, that channel

should be avoided up to the point where using it would decrease the wait time. The selection algorithm is changed once again to the one that simply selects the channel that has the smallest outstanding queue. This is shown in figure 14 to decrease the load of the channel with the high BER compared with the random selection algorithm. This shows an important tradeoff with channel selection algorithms.

5. Discussion

Generally speaking, using multiple channels to transmit data improves the over all bit rate available. As the bit error rates of the channel fluctuate or the channels become congested performance can be modified by changing the channel selection algorithm. Only simple, local algorithms are explored in this paper. Each algorithm has its strength and weakness. It appears that using only information about the current state of the each channels queue is insufficient to make the best possible choice of channel. Global knowledge of the channel such as overall congestion, wait times and BER are required in order to select the best channel for all cases.

6. Future work

The BER model used is constant during the simulation. Adding BER blackouts e.g for a while a certain channel goes dead would perhaps be interesting. Also seeing what happens to an application that requires low latency during black outs could be interesting.

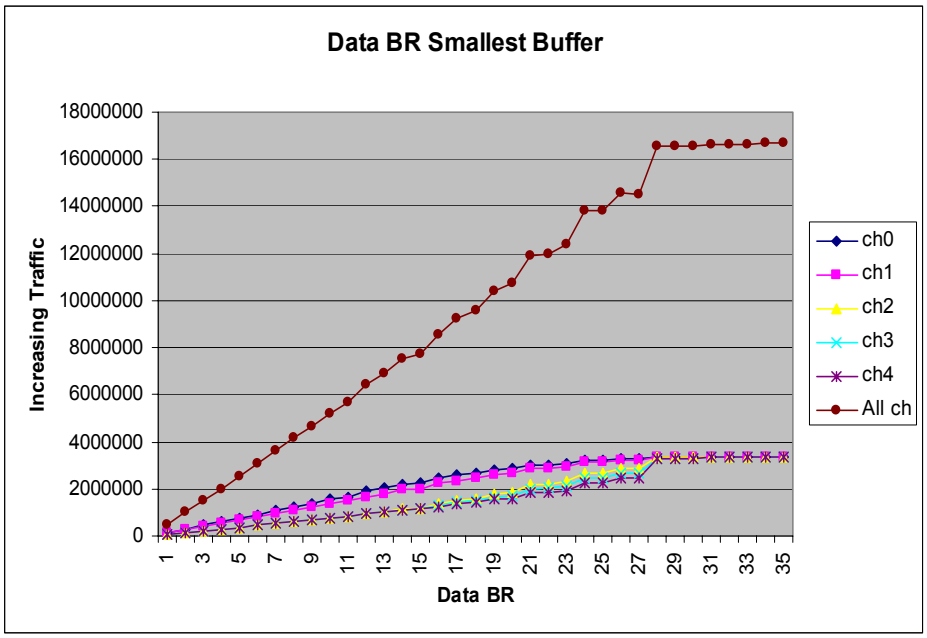

Figure 3

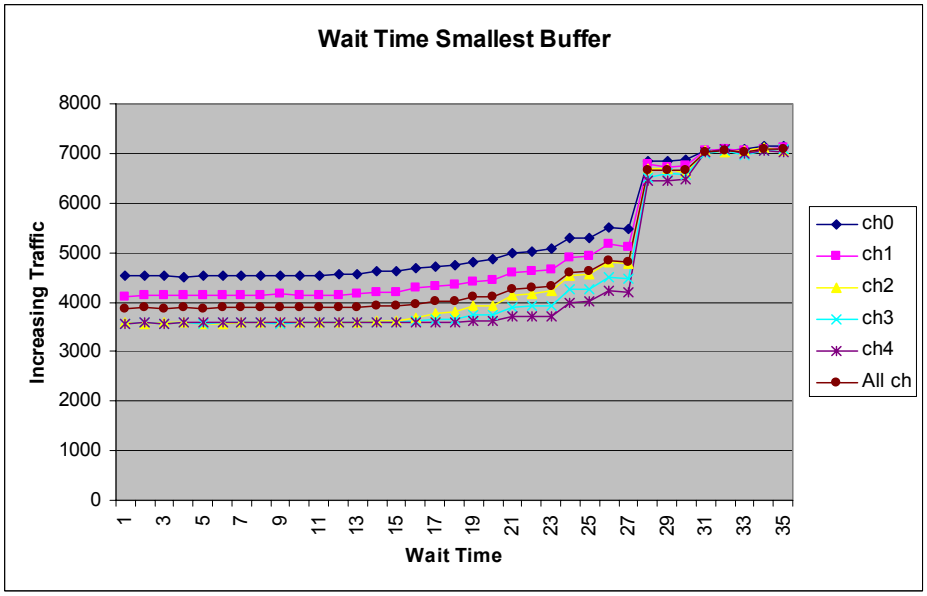

Figure 4

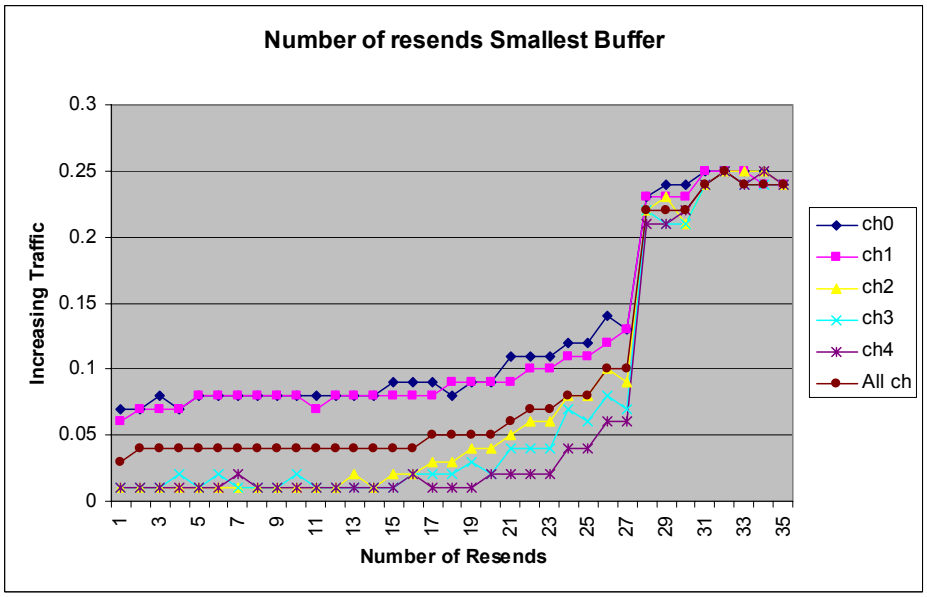

Figure 5

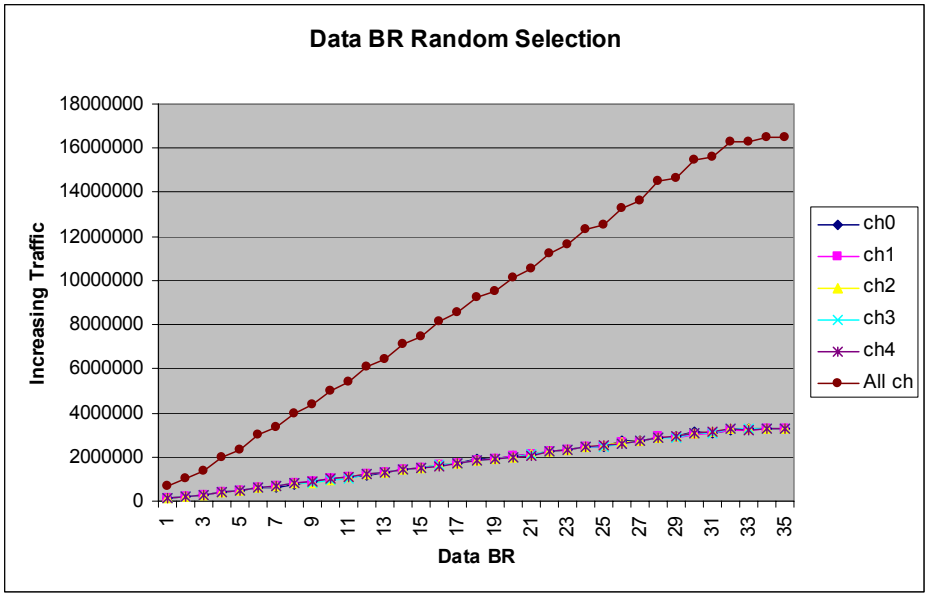

Figure 5

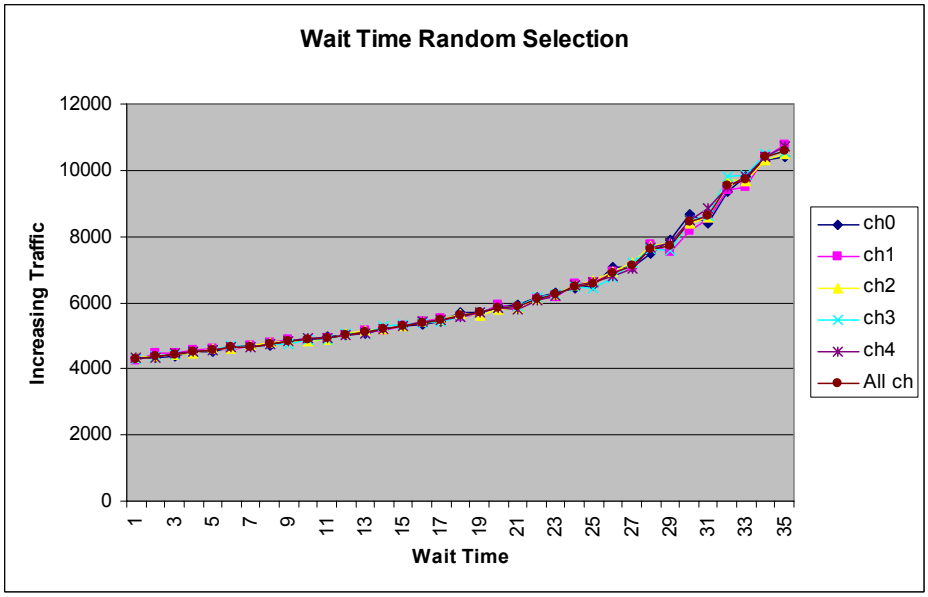

Figure 6

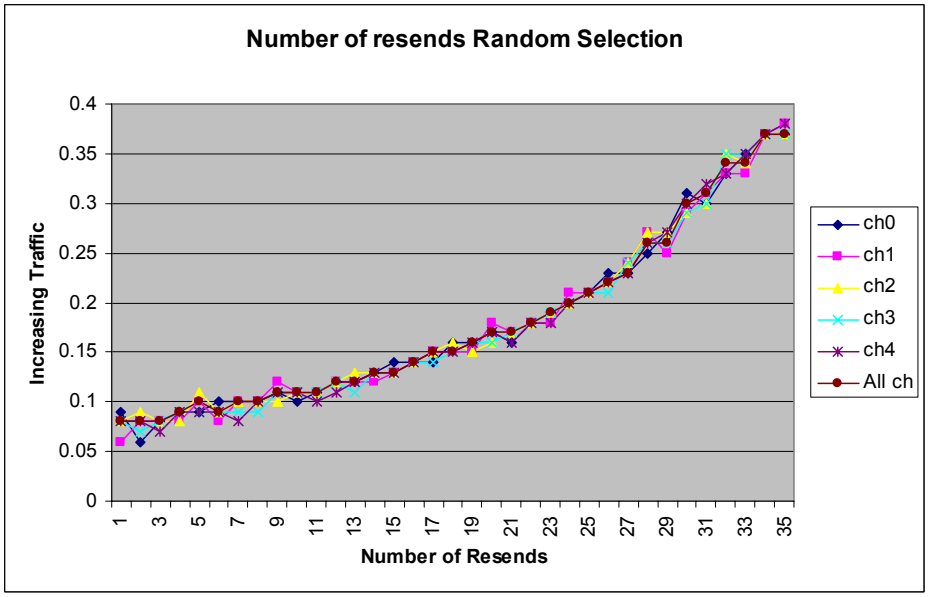

Figure 7

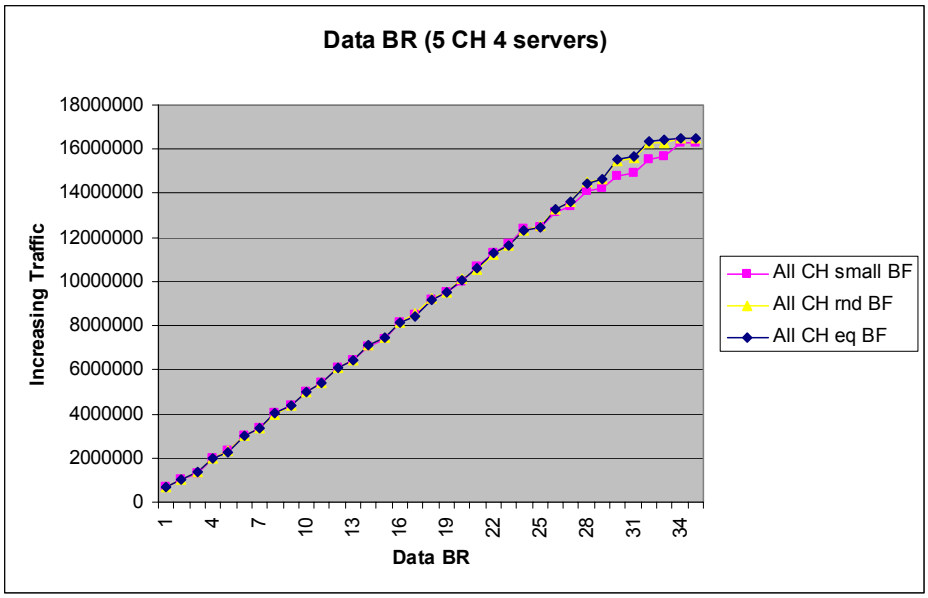

Figure 8

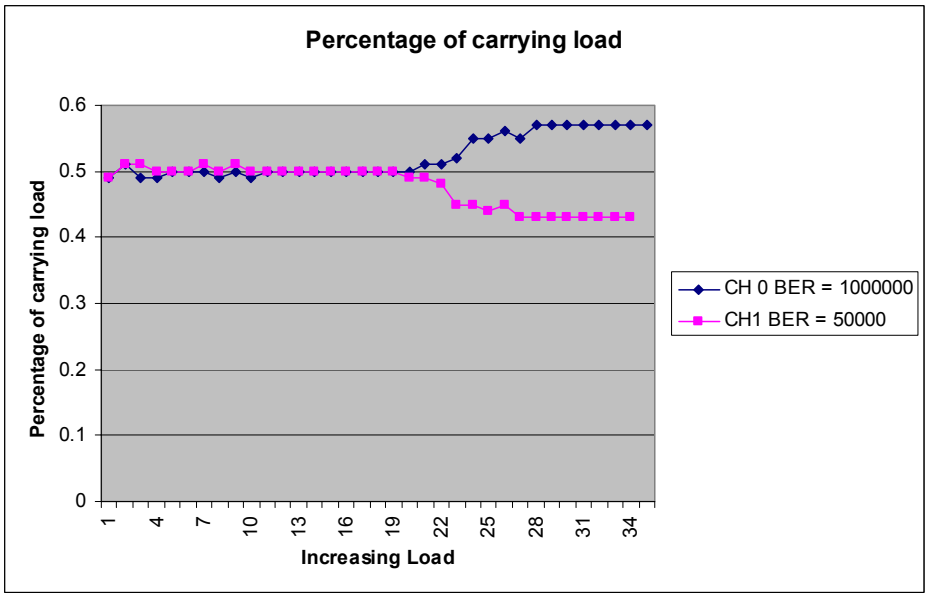

Figure 9

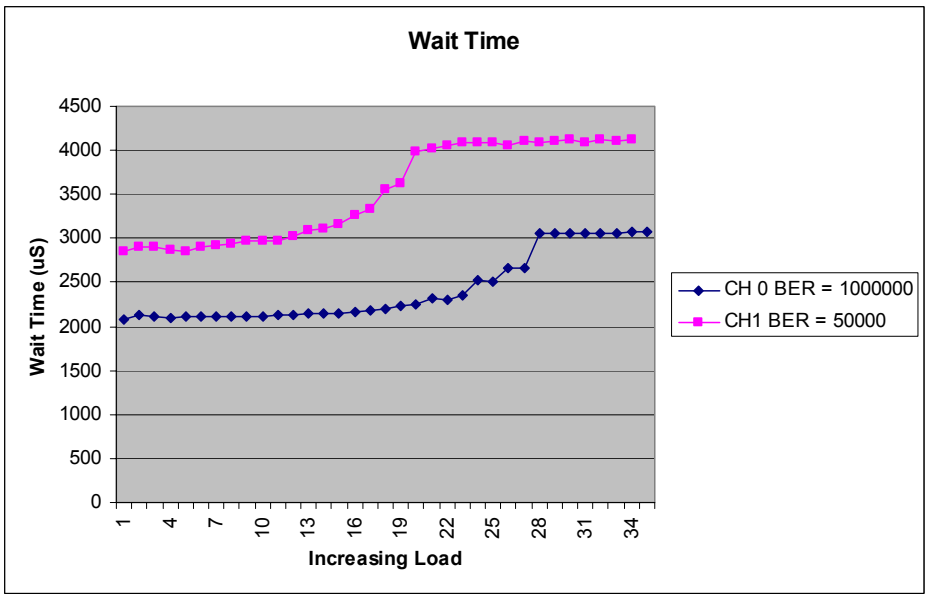

Figure 10

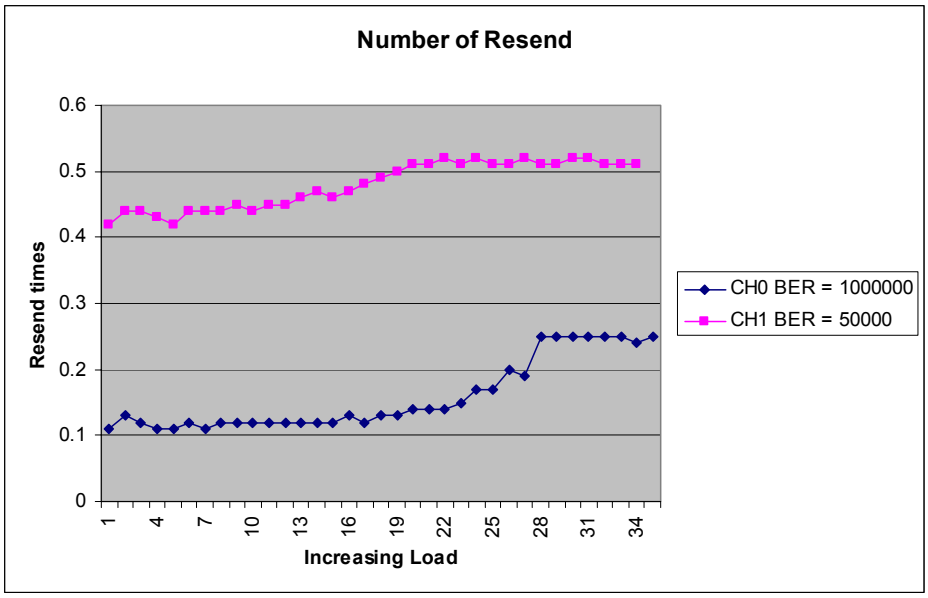

Figure 11

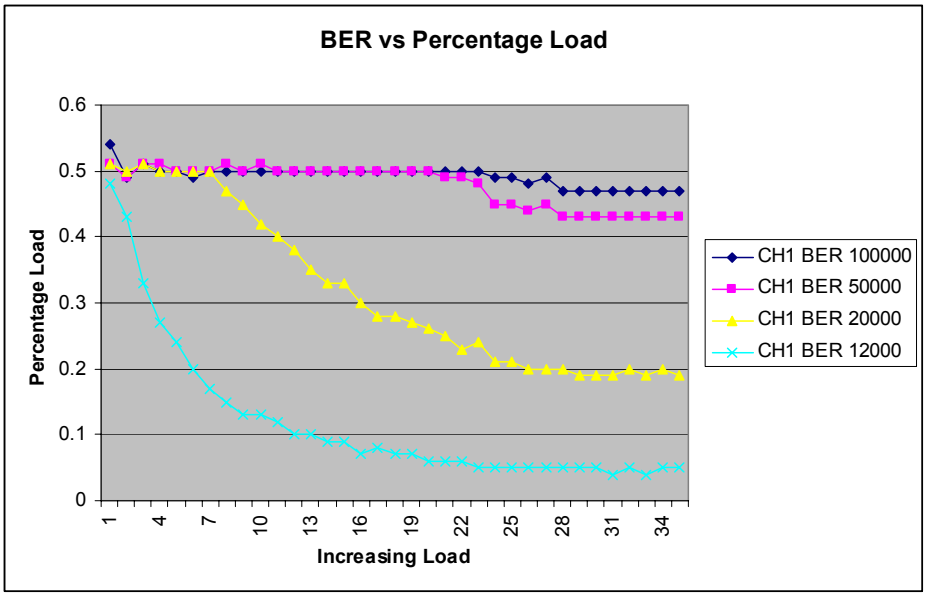

Figure 12

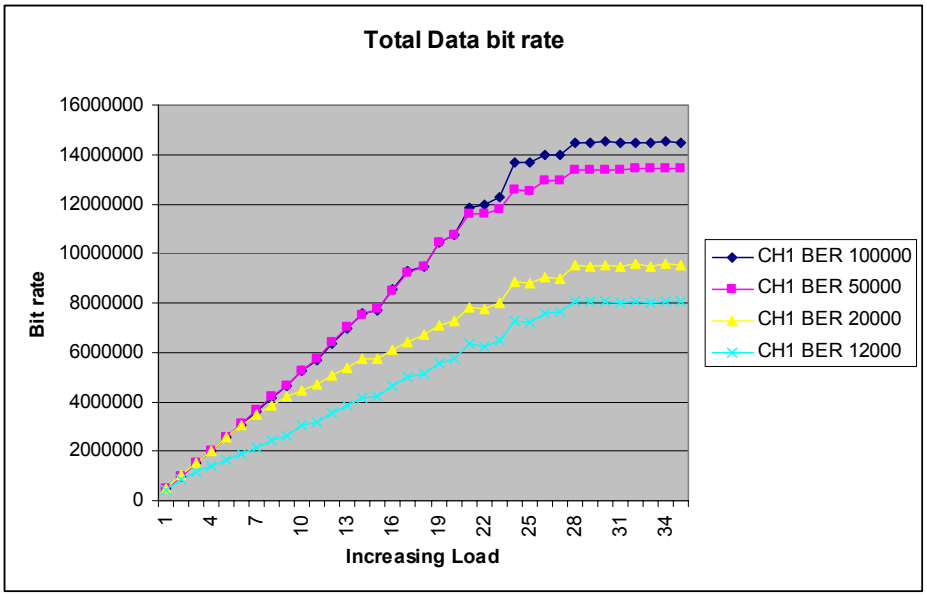

Figure 13

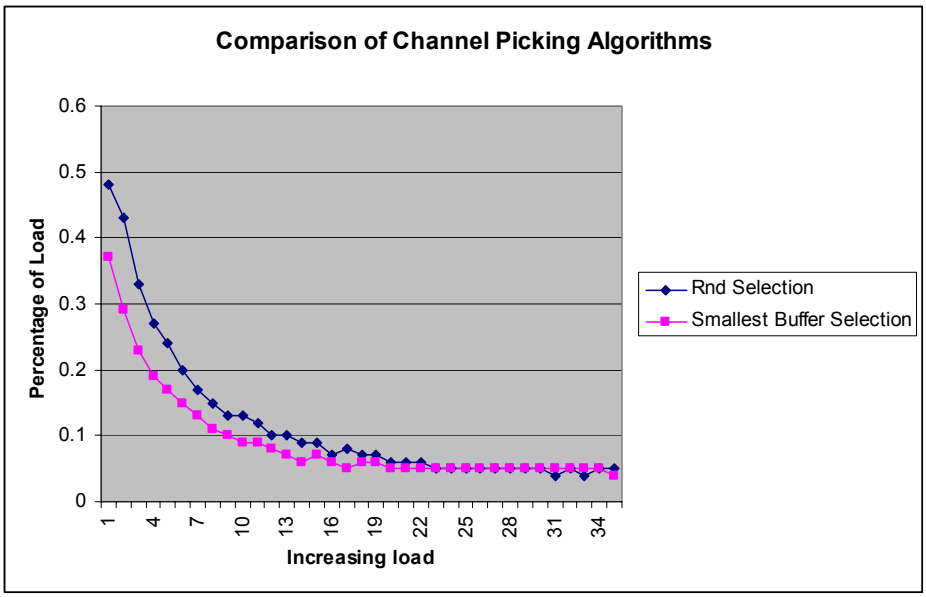
Comparison of Channel Picking Algorithms
Percentage of Load
0.6
0.5
0.4
0.3
0.2
0.1
0
1
4
7
10
13
16
19
22
25
28
31
34
Increasing load
Rnd Selection
Smallest Buffer Selection

Figure 14