CMPUT313 - Assignment 2 Report
Mikus Lorence
ID: 1227388

Overview

Overall Implementation:

Each station is kept track of with an array of structs that contains information and statistics on
the station. Pending frames of each station are kept track of in a linked list structure in order to implement a FCFS system for frames. Upon the incrementation of a “slot time”, each frame in the pending frames stack of every station has its delay incremented by 1.

Time Division Multiplexing:

At each slot time, a check is performed on the station corresponding to the current slot on the bus. If that station has frames on its stack, then it transmits one frame.

Slotted ALOHA with probabilistic backoff:

At each slot time, an array is initialized to keep track of stations wanting to transmit in that slot. If a station has frames on its queue, and it hasn’t already tried to transmit and failed (tryingToTx == 0), then add it to the array. If a station has already tried to transmit and failed, then add it to the array with a probability of 1/N.

If the size of the array is greater than 1 (more than one station is trying to transmit), then the variable tryingToTx of all those stations to 1, so that next slot, they will be added to the array with probability of 1/N. If the size of the array is 0, then transmit the frame of the one station that wants to transmit.

Slotted ALOHA with interval-based backoff:

This protocol was implemented similarly to the previous one, but here, the tryingToTx variable has a different meaning. If it is set to ‘-1’, that means that the station has not yet tried to transmit a frame, and if it has a frame to transmit, it will attempt to do so with probability one. Any other number (greater than -1) represents how many slots until the station tries to transmit again (with 0 essentially having the same effect as setting it to -1).

Stations are added to the “current array” if they have frames on the queue, and if tryingToTx is less than 1. If the size of the array is greater than 1, each station in the array has its tryingToTx set to a random number between 1 and N. At the end of each slot, each station has its tryingToTx value decremented.

Slotted ALOHA with a truncated binary exponential backoff:

This protocol is again implemented in a similar way to the previous one. Each station keeps track of the "binary exponent", with it being initially set to 1, and reset to 1 upon every transmission. When choosing the interval (as in the previous protocol), a random number between 1 and 2^(intExp) is generated, where intExp is the current value of that station's "binary exponent". Upon every collision, that exponent is increment by 1, until it reaches 10.

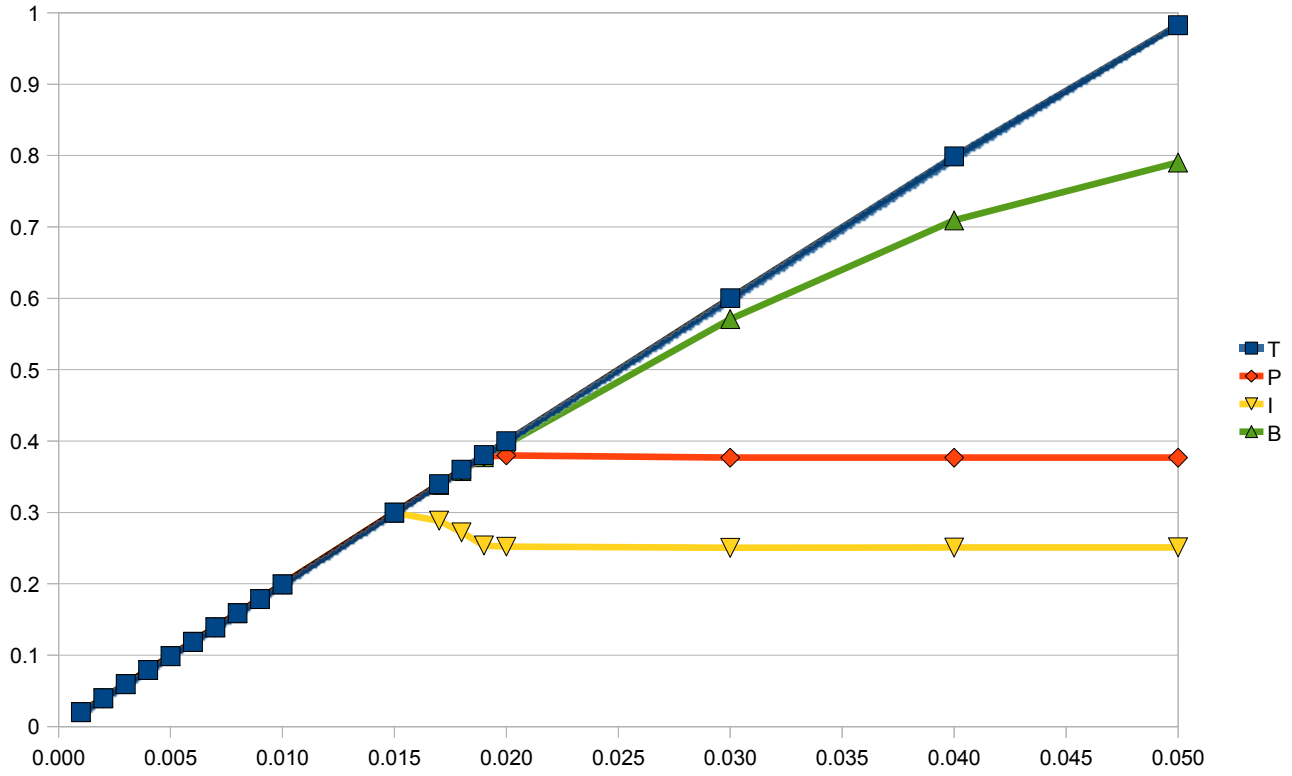
Analysis

After completing the simulation I started to choose values for the probability of frame generation to run my simulation with. I found that, as suggested, a value around 0.05 is a good upper bound for the set of values, as at this probability, all the protocols begin to perform relatively inefficiently. For the lower bound I went with a value of 0.001, as this seemed to be the probability at which each protocol was able to achieve its best possible performance. I then chose some value in between the two bounds to get the set of values for p : 0.001, 0.002, 0.003, 0.004, 0.005, 0.006, 0.007, 0.008, 0.009, 0.01, 0.015, 0.017, 0.018, 0.019, 0.02, 0.03, 0.04. The reason for so many values between 0.001 and 0.010 is because those are the more realistic values for p. The many values between 0.01 and 0.02 are due to the fact that between those two values, the average delay for 3 of the 4 protocols rises from single to triple digits. For the values of N, R, and T, I used the suggested values of 20, 50000, and 5 respectively. The seeds were altered for each trial, but the same set of 5 seeds was used for each of the protocol and each of the p values. The input values were as follows:

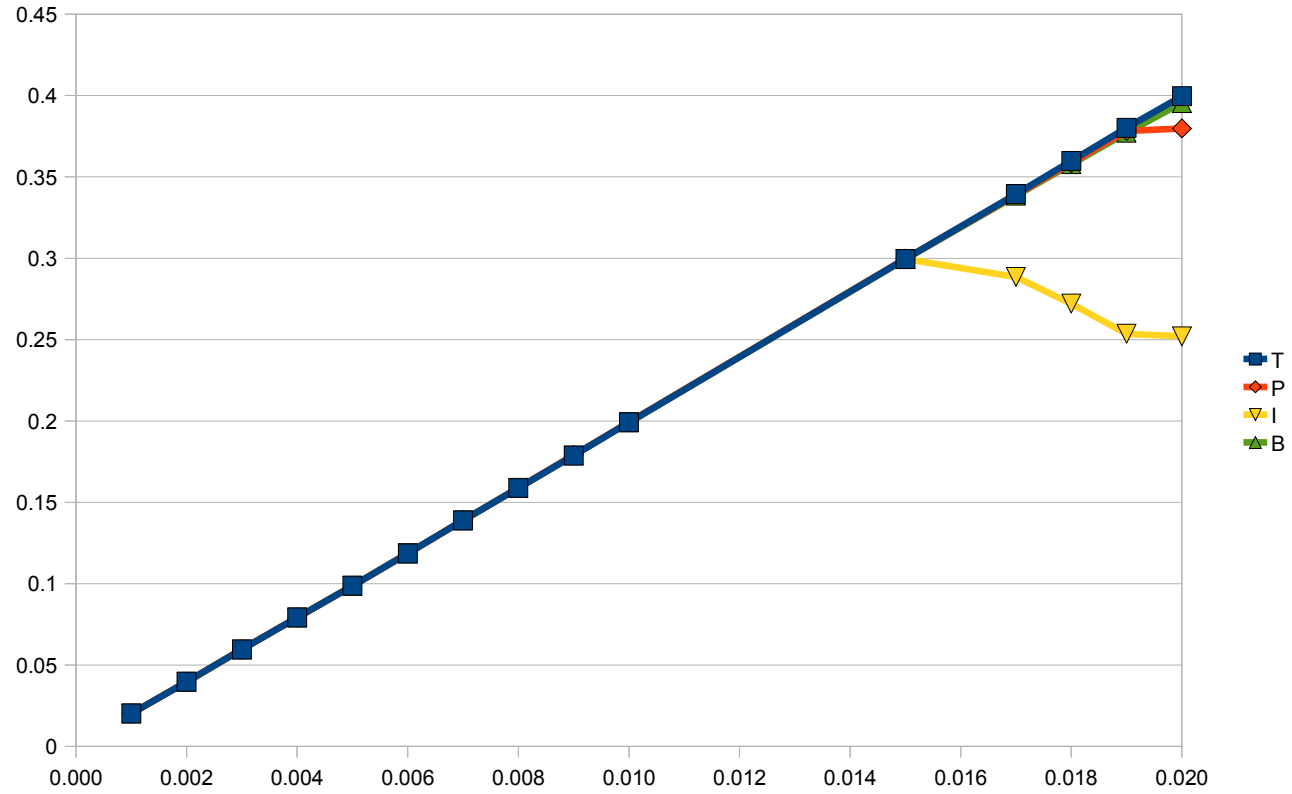
Protocol:	{'T', 'P', 'I', 'B'}
N:	20
p:	{0.001, 0.002, 0.003, 0.004, 0.005, 0.006, 0.007, 0.008, 0.009, 0.01, 0.015, 0.017, 0.018, 0.019, 0.02, 0.03, 0.04}
R:	50,000
T:	5
Seeds:	{1, 2, 3, 4, 5}

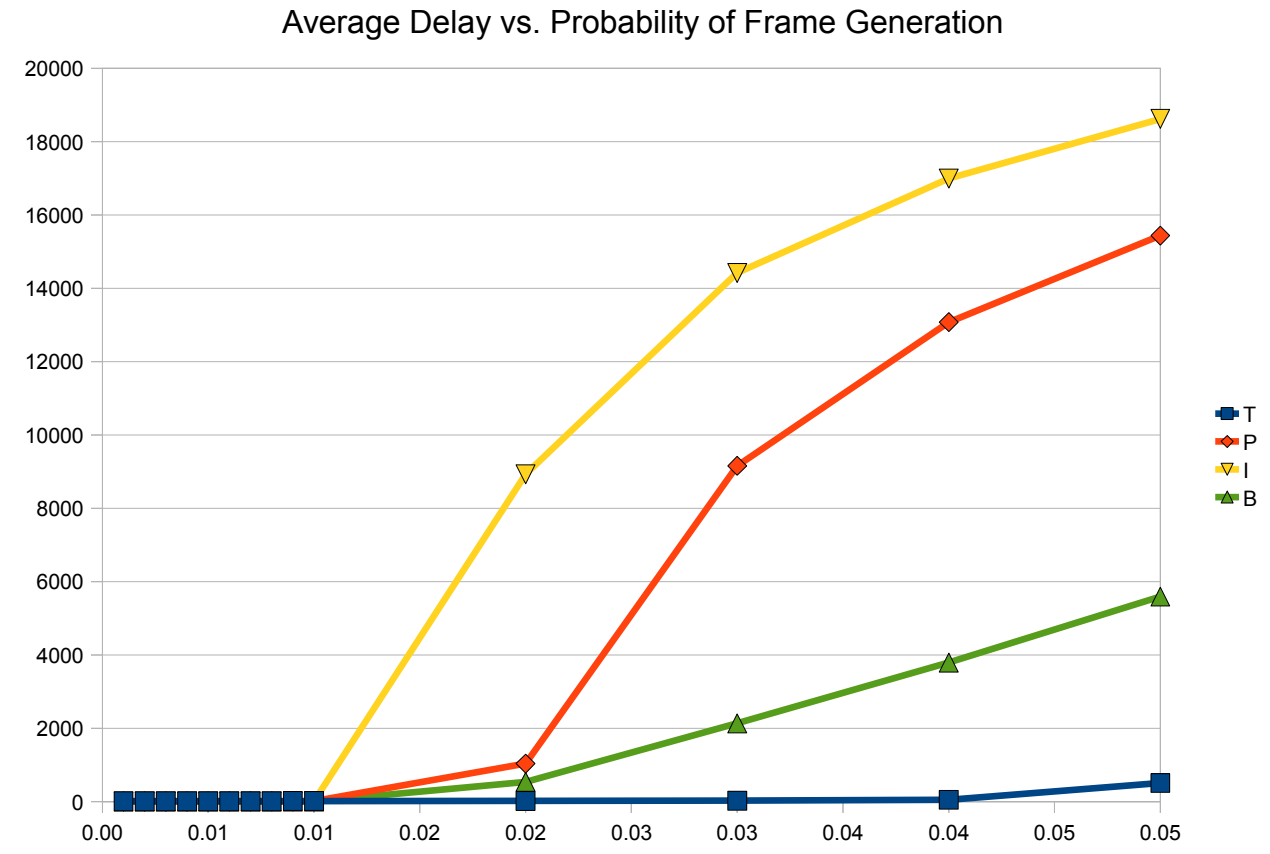
I then graphed the curves for throughput of the four protocols on one graph, and the curves for the average delay of the four protocols on another (graphs on the following pages):

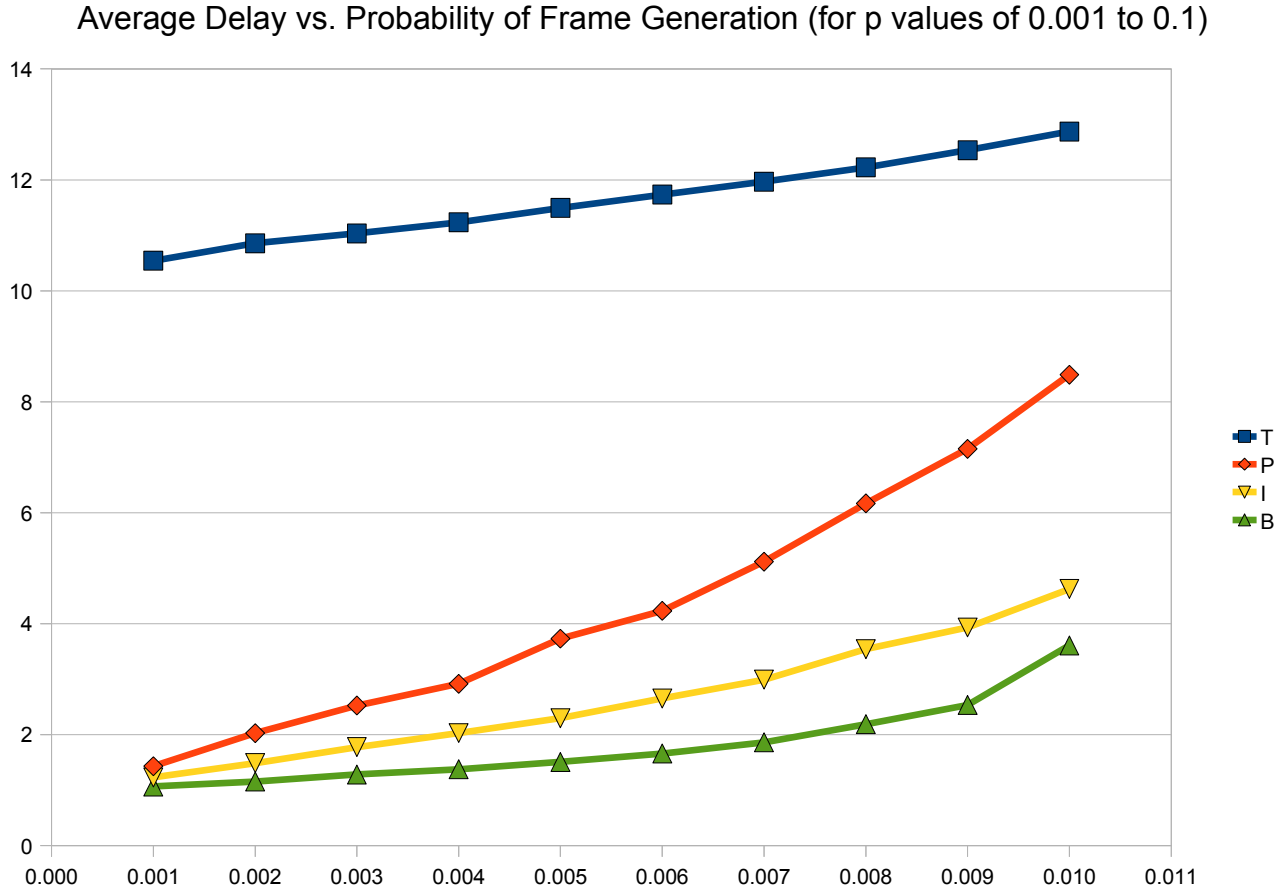
Throughput vs. Probability of Frame Generation

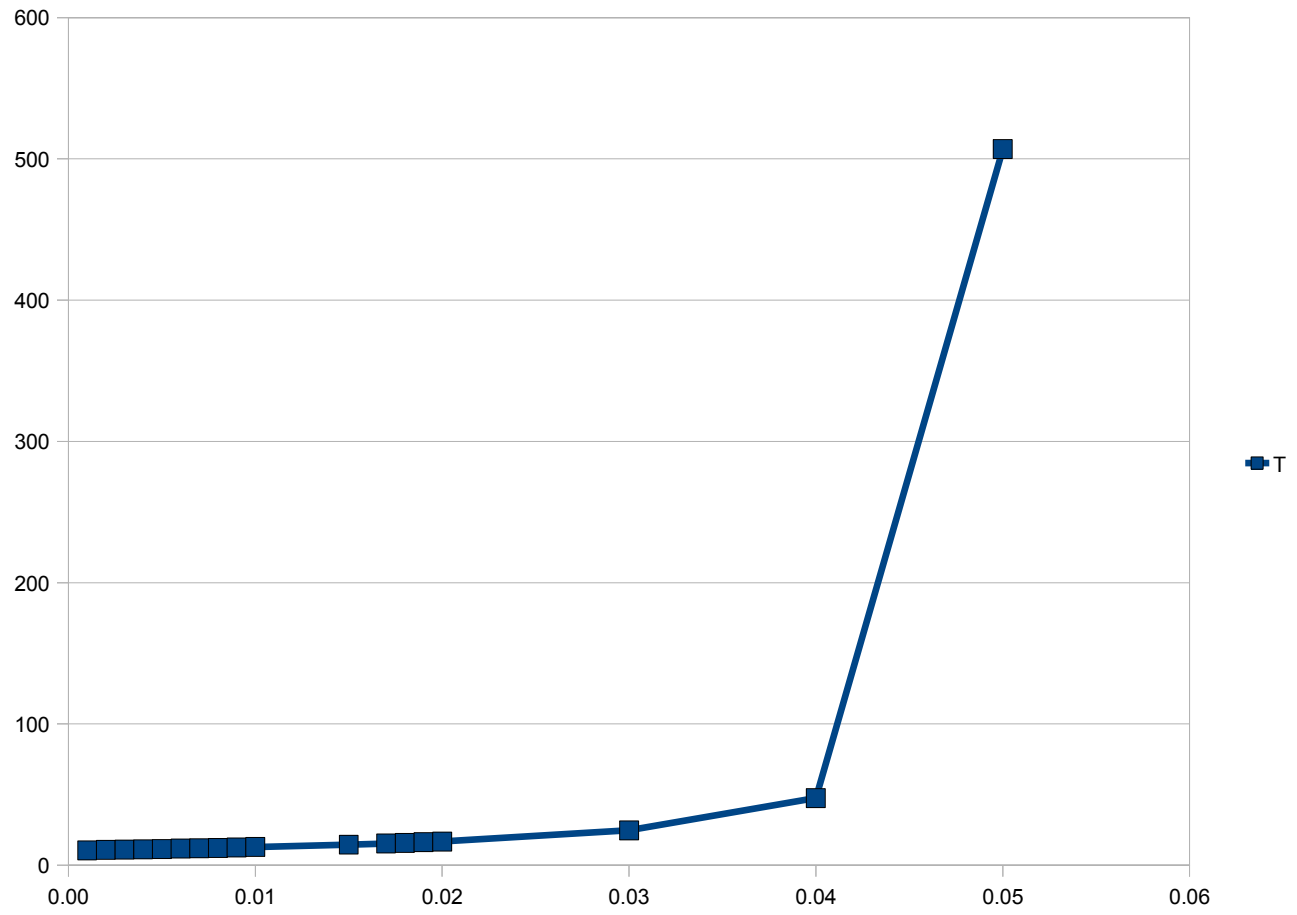


Throughput vs. Probability of Frame Generation (for p values of 0.001 to 0.2)




Average Delay vs. Probability of Frame Generation
20000
18000
16000
14000
12000
10000
8000
6000
4000
2000
0
0.00
0.01
0.01
0.02
0.02
0.03
0.03
0.04
0.04
0.05
0.05
T
P
I
B


Average Delay vs. Probability of Frame Generation (for p values of 0.001 to 0.1)
14
12
10
8
6
4
2
0
0.000
0.001
0.002
0.003
0.004
0.005
0.006
0.007
0.008
0.009
0.010
0.011
T
P
I
B



Time Division Multiplexing

p	Throughput	Confidence	Interval	Avg. Delay	Confidence	Interval
0.001000	0.020084	0.019185	0.020983	10.542121	9.867815	11.216427
0.002000	0.039608	0.038198	0.041018	10.857864	10.616056	11.099673
0.003000	0.059496	0.057845	0.061147	11.034711	10.756354	11.313068
0.004000	0.079172	0.075182	0.083162	11.235039	10.898932	11.571147
0.005000	0.098636	0.094771	0.102501	11.496101	11.110295	11.881907
0.006000	0.118576	0.112459	0.124693	11.735053	11.442584	12.027523
0.007000	0.138864	0.131555	0.146173	11.968849	11.660495	12.277203
0.008000	0.158764	0.151535	0.165993	12.225883	11.911772	12.539993
0.009000	0.178728	0.169721	0.187735	12.535601	12.382121	12.689080
0.010000	0.199176	0.190502	0.207850	12.872375	12.754127	12.990623
0.015000	0.299532	0.290206	0.308858	14.526127	14.327383	14.724872
0.017000	0.339520	0.329690	0.349350	15.357761	15.181443	15.534080
0.018000	0.359828	0.349237	0.370419	15.806143	15.558821	16.053464
0.019000	0.380232	0.370696	0.389768	16.297781	15.913525	16.682038
0.020000	0.399664	0.389705	0.409623	16.798908	16.394751	17.203065
0.030000	0.600228	0.584910	0.615546	24.628745	23.940627	25.316863
0.040000	0.798680	0.787116	0.810244	47.509933	44.879553	50.140314
0.050000	0.982704	0.977115	0.988293	506.978936	368.200624	645.757247

Slotted ALOHA with probabilistic backoff

p	Throughput	Confidence	Interval	Avg. Delay	Confidence	Interval
0.001000	0.020100	0.019206	0.020994	1.430182	0.343290	2.517074
0.002000	0.039600	0.038269	0.040931	2.026110	1.610931	2.441289
0.003000	0.059504	0.057882	0.061126	2.523955	2.061595	2.986315
0.004000	0.079144	0.075292	0.082996	2.915689	2.550377	3.281001
0.005000	0.098660	0.094497	0.102823	3.729597	2.897931	4.561263
0.006000	0.118420	0.112365	0.124475	4.230441	3.573319	4.887564
0.007000	0.138956	0.131312	0.146600	5.117397	4.426780	5.808014
0.008000	0.158892	0.151667	0.166117	6.167001	5.435984	6.898017
0.009000	0.178972	0.169467	0.188477	7.150185	6.052878	8.247492
0.010000	0.199244	0.190843	0.207645	8.487443	7.163203	9.811683
0.015000	0.299816	0.290726	0.308906	19.609450	18.722658	20.496243
0.017000	0.338984	0.326601	0.351367	31.305652	23.272598	39.338706
0.018000	0.358584	0.349993	0.367175	46.709499	35.779771	57.639227
0.019000	0.378316	0.371843	0.384789	79.920726	42.176516	117.664937
0.020000	0.379864	0.371690	0.388038	1034.346402	472.045098	1596.647705
0.030000	0.376732	0.372723	0.380741	9156.333487	8626.958377	9685.708597
0.040000	0.376616	0.373921	0.379311	13075.042958	12873.475952	13276.609963
0.050000	0.376700	0.372914	0.380486	15437.315472	15119.564624	15755.066319

Slotted ALOHA with interval-based backoff

p	Throughput	Confidence	Interval	Avg. Delay	Confidence	Interval
0.001000	0.020096	0.019194	0.020998	1.226683	0.857648	1.595718
0.002000	0.039624	0.038205	0.041043	1.484938	1.197330	1.772546
0.003000	0.059500	0.057914	0.061086	1.773975	1.584789	1.963162
0.004000	0.079188	0.075239	0.083137	2.027137	1.782084	2.272190
0.005000	0.098668	0.094795	0.102541	2.294533	1.902361	2.686706
0.006000	0.118620	0.112571	0.124669	2.650681	2.557738	2.743623
0.007000	0.138864	0.131579	0.146149	2.991016	2.343759	3.638273
0.008000	0.158804	0.151710	0.165898	3.541008	3.110190	3.971825
0.009000	0.178708	0.169719	0.187697	3.930656	3.693960	4.167353
0.010000	0.199252	0.190592	0.207912	4.624124	4.110889	5.137359
0.015000	0.299560	0.290364	0.308756	10.530045	8.120660	12.939431
0.017000	0.288496	0.195230	0.381762	2754.014382	-4539.070780	10047.099544
0.018000	0.272060	0.214831	0.329289	4775.102154	-1151.944327	10702.148635
0.019000	0.253572	0.244778	0.262366	7863.755218	5999.498761	9728.011675
0.020000	0.251988	0.246863	0.257113	8927.989161	8404.057666	9451.920656
0.030000	0.250416	0.245073	0.255759	14417.991283	13999.738138	14836.244428
0.040000	0.250716	0.247391	0.254041	16990.398788	16838.813391	17141.984185
0.050000	0.251112	0.247047	0.255177	18618.000772	18314.128257	18921.873286

Slotted ALOHA with a truncated binary exponential backoff

p	Throughput	Confidence	Interval	Avg. Delay	Confidence	Interval
0.001000	0.020092	0.019183	0.021001	1.063026	0.988529	1.137523
0.002000	0.039612	0.038226	0.040998	1.154172	1.059571	1.248773
0.003000	0.059520	0.057888	0.061152	1.281645	1.156165	1.407124
0.004000	0.079164	0.075181	0.083147	1.374942	1.315520	1.434364
0.005000	0.098640	0.094721	0.102559	1.506597	1.345607	1.667588
0.006000	0.118584	0.112507	0.124661	1.659447	1.429209	1.889685
0.007000	0.138860	0.131782	0.145938	1.861234	1.685492	2.036976
0.008000	0.158780	0.151713	0.165847	2.187665	1.910375	2.464955
0.009000	0.178792	0.169638	0.187946	2.532361	2.420227	2.644496
0.010000	0.199180	0.190448	0.207912	3.608164	2.397810	4.818518
0.015000	0.299500	0.289772	0.309228	35.353817	-1.081760	71.789395
0.017000	0.338528	0.328921	0.348135	165.184463	69.587281	260.781644
0.018000	0.357868	0.346960	0.368776	264.762230	180.006094	349.518366
0.019000	0.377148	0.366239	0.388057	409.336101	342.315840	476.356361
0.020000	0.395444	0.385747	0.405141	539.815504	474.745795	604.885214
0.030000	0.570820	0.552450	0.589190	2134.079945	1836.780951	2431.378939
0.040000	0.709264	0.699480	0.719048	3792.902784	3028.070042	4557.735527
0.050000	0.790216	0.768032	0.812400	5589.380135	4652.664845	6526.095425

Comparing the throughput of the four protocols, it is clear that the protocols all have virtually identical throughput for value of p up to 0.01. Not only are they similar, but they also seem to rise fairly linearly. The more frames are generated, the higher the throughput of the system. The time division multiplexing (T) protocol continues to rise linearly for all values of p. This is due to the fact that in the T protocol, there are no collisions, therefore the amount of frames that are transmitted is directly proportional to the probability that frames are generated. At a probability of 0.05, when there are 20 stations on the bus, after 50,000 slot times, each station will generate approximately 2,500 frames (50,000 * 0.05). This is why the overall throughput at p = 0.05 is very close to 1. The reason it is slightly below is because there are some stations that will generate less than 2500, which will deduct from the maximum possible throughput of 1. Those stations that generate more than 2500 frames will still not be able to transmit more than 1 frame for every 20 slots. For values of p higher than 0.05, the throughput will continue to get close to 1, and will reach 1 at p = 1. The P and I protocols begin to level off for values of p higher than 0.01 because with a higher number of frames being generated, the transmissions will begin to collide, causing throughput to be below the maximum possible throughput.

The B protocol's throughput starts to deviate from the linear growth of the T protocol, and follows a more logarithmic growth pattern. It's throughput is better than the P and I protocols because as more and more transmissions being to collide, the protocol starts to assign slots to frames that are much later in time (up to 1000 slots later), allowing newly generated frames of other stations to transmit more. The other two protocols only wait up to around 20 more slots before trying to transmit again, causing newly generated frames to collide more and more often. The I protocol performs slightly worse than the P protocol in terms of throughput, and its throughput even drops slightly between p values of 0.015 and 0.02. The reason why the I protocol performs slightly worse is because the largest possible amount of slot time that a frame can wait before retransmission is 20. In the P protocol, because it is probability based, there is a chance that some frames will wait longer than 20 slot times, allowing more newly generated frames of other stations to transmit, slightly

improving the throughput over the I protocol. The reason that the throughputs for I and P level off, and stop rising is because after a certain point, every station will have frames in queue, and the newly generated frames just start to build up in each stations queue. The system will still transmit the frames at the same rate, but with more frames in queue. The B protocol's throughput would eventually level off as well, but it isn't visible on the graph because it's technique of increasing the waiting interval up to (1,1024) allows the throughput to keep rising with the probability of frame generation.

Looking at the average frame delay of the protocols, it is clear that the T protocol performs well for high values of p, and comparably terrible for low values of p. For very low p values such as 0.001 through 0.010, the P, I, and B protocols all have a very good average delay from almost 1 to as high as 8. The B protocol is the closest to 1, because with so little frames being generated and almost no collisions, in the case of a collision it will almost always retransmit in the next 1 or 2 slots. The T protocol, however, has roughly 10 times more average frame delay. This is because even with no collision, the stations will have to wait an average of N/2 slot times before it is their turn to transmit. So the average delay for T is approximately 20/2 = 10. This trend continues for the four protocols up to the p value of 0.01. The B protocol's throughput rises slowly compared to the P and I protocols because for such low values of p, most frames will still retransmit in the next 1 or 2 slots.

For the T protocol, the average delay experiences almost no increase (when compared to the other 3 protocols) when the p values increase from 0.01 to 0.05. This is because for these values of p, most stations will likely have only one frame in their queue because the probability dictates that they won't generate more than 1 frame in a 20 slot time cycle. It is only when the probability increase to 0.05 that the delay starts to go up faster than linearly. This is because for with a 20 slot cycle, there is a much higher chance for one station to generate more than 1 frame in a cycle. The reason the average delay for the T protocol doesn't rise as much as the other protocols is because every station is guaranteed to get a chance to transmit soon (when the stations slot comes up). How soon a frame in the queue of a station gets to transmit is a function of how many frames that specific station has in the queue, which is positively correlated with the p value.

The P and I protocol experience almost logarithmic growth between p values of 0.01 and 0.05. The B protocol only starts to grow seemingly linearly. This is because at very high values of p, with many frames being generated, the B protocol holds off the collided transmissions for much longer periods of time, allowing newly generated frames of other stations to transmit. The station that ended up holding off its transmission is the one whose frames end up with a really high delay, but the other stations that are able to transmit with less collision during that time period end up with lower average delay. The P and I protocols would keep retransmitting the collided transmissions at intervals of approximately 1 to N, causing more collision to occur causing the delay of all stations to rise until one of the stations is able to get a transmission through.

According to the above analysis, the B protocol seems to behave fairer compared to the P and I protocols because it allows stations that have not yet collided to have a higher chance to transmit, and stations that have collided many times have to wait a long time before attempting to retransmit their frame. The T protocol is however the fairest by design, because each station gets a an equal slot in which it can transmit, and if a station has to wait, it is only because it wants to transmit a larger number of frames.

This analysis leads me to conclude that for lower (and possibly more realistic) p values between 0.001 and 0.010, the P, I, and B protocols are more efficient by almost an order of magnitude (at least for the p = 0 .001 case), compared to the T protocol. Efficiency being

defined by the average frame delay and not throughput because for this range of p, the throughput rises directly proportional to p for all 4 protocols. For higher (although less realistic) values of p than 0.010, the T protocol has the best performance in terms of both average delay and throughput. The B protocol isn't that much worse than the T protocol for high p values, and since such high load is not as realistic and less likely to occur, I would prefer to use the B protocol over any other protocol. It performs the best under low load, and in the rare cases of high load it wouldn't perform as terribly as the P and I protocols.