CMPUT 313 Assignment 2

Stephen Jahns and Marek Kundera

Tuesday, March 13

Introduction

Choosing an appropriate backoff strategy for the Slotted ALOHA MAC layer protocol depends on a series of factors. In this report, we have simulated the Probabilistic Backoff (P), Interval-Based Backoff (I), and Truncated Binary Exponential Backoff (B) strategies. In addition, we have simulated the Time Division Multiplexing (T) protocol for additional comparison. Our simulation was based on several assumptions found in the problem specification and on a terse description of each protocol:

T: a station i only transmits at slot number $mN+i$, $m = 0, 1, 2, ... N$, where m is the cycle number, and N is the number of stations

P: a station will try to transmit immediately on generation, or retransmit with $1/N$ probability at each subsequent slot

I: a station will try to transmit immediately on generation, or retransmit at a slot 1 to N slots later, chosen randomly

B: a station will try to transmit immediately on generation, or retransmit at a slot 1 to 2 slots later, then 1 to 4 slots later, or eventually 1 to 2^i slots later, chosen randomly until the frame successfully transmits

To set some groundwork, we will begin by outlining our expectations with a preliminary analysis. Next we will briefly explain the main functionality and design of our `psim` simulator. Then we will reveal the simulation results with the help of some tables and graphs, paired with the parameters we used to generate the media. Finally, we will examine each protocol's performance by analyzing their average throughputs and the overall average per-frame slot time delays of the successfully transmitted frames. In addition, we will examine the fairness of each protocol by observing the number of undelivered frames under each protocol at the individual stations.

Time Division Multiplexing

Here we expect that as the probability of frame generation, p, increases, the average frame delay should increase. This is because in any given cycle of N slots, the probability of a frame being generated at that slot increases. Therefore as p increases, up to N frames might be generated in a cycle. On the other hand, the mean throughput should increase in direct proportion to p. As p increases, it is more likely frames will be available for transmission in a given cycle. In other words, it is more likely a station will not waste its designated slot time. Furthermore, since each station has a designated slot only it can transmit on, it is not possible for collisions to occur between the stations.

Probabilistic Backoff

Notice that for a probability of successful transmission of $1/N$, we need N stations to each have a frame queued for transmission, in order for one to *likely* transmit. We can build on this observation and hypothesize that at some p, every station will be generating frames faster than it can transmit them. And therefore, the throughput will level off in concert with the probability that *exactly one* station transmits, i.e. let event $ExactlyOneTx$ represent the case where exactly one station is transmitting. Thus, when N=20 stations, $Prob(ExactlyOneTx) = C(20,1) \times (1/20) \times (19/20)^{19} = 0.377$. For 20 stations, it is likely then that when $p = 0.377/20 = 0.019$, we can expect the throughput to begin leveling off and for the average frame delay to start increasing. This is because as p increases, frames are more likely to generate, while the throughput remains the same. And so, the queue can only get larger thus each frame must wait longer before moving to the front of the queue for transmission.

Interval-Based Backoff

Methods

The `psim` simulator was implemented in C++. It requires several parameters at the command line:

```
psim Protocol N p R T seed1 seed2 ... seedT
```

`Protocol` can taken on a value of `T`, `P`, `I`, or `B`, depending on what protocol is being simulated. The parameter `N` represents the number of stations that will be transmitting under the protocol. `p` represents the probability that a frame will be generated at each station. `R` represents the number of slots that the experiment will run, and is therefore the time-measure for the experiment. And `T` represents the number of trials the simulation will run, while the following `seed1...seedT` inputs are used to reseed the randomization functions for each trial.

For the purpose of this simulation, the values `N=20`, `R=50000`, and `T=5` will be fixed. In other words, we will simulate 5 trials where 20 stations transmit over a time period of 50,000 slots. The observations shown later are based on 5 trials that were seeded with values 1, 2, 3, 4, 5, respectively.

`psim` functions by simulating `T` independent trials of hub of `N` stations running for `R` time slots, using the random number generator seeds separately specified for each trial. At the beginning of each timeslot, each station generates a new frame on its transmission queue with probability `p`. Then, the delay value in time slots is incremented for each frame on each stations transmission queue. Then, the simulator determines which and how many stations will attempt transmission in that timeslot, according to the MAC protocol specified in `Protocol`. If there is only one station attempting transmission, the transmission is deemed successful and the frame is removed from that station's queue. If there are more than one transmissions, a `tx_collide(slot)` method is executed on each station that partly implements the station's backoff protocol as specified in `Protocol`.

The simulator outputs the parameters (minus the seeds) on the first line. After running all trials, it calculates and prints 95% confidence intervals for the overall average throughput in frames delivered per time slot, and delay in slots for each frame successfully delivered, on lines 2 and 3 respectively. Confidence intervals were calculated using appropriate t-statistics for the number of trials from a file labelled `tvalues.dat`. Per-station statistics were output on the remaining lines, where each line started with `nN`, the id of the station, followed by confidence intervals for throughput and delay per frame delivered, and finally the ratio of undelivered frames to total frames generated by that station for each trial separately, all on the same line.

Experiments were automated using the BASH script `make_plot.sh` to run the simulator for each protocol over ranges of `p` values ranging from 0.00 up to 0.04. The scripts parsed data from the output into appropriately formatted data files that were sent to

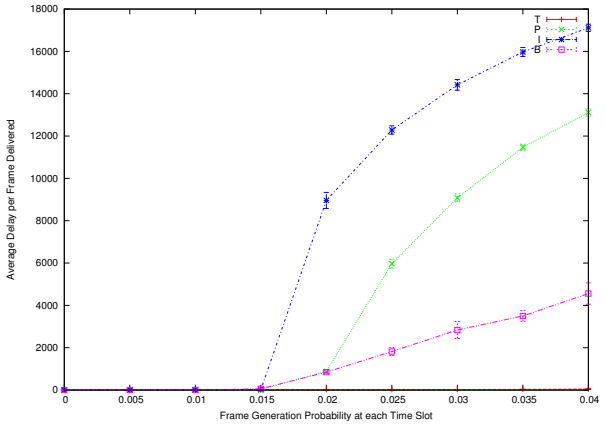

Figure 1: Average delay per frame under each protocol for p:(0,0.04; @0.005 increments). Used to illustrate what happens as p nears 0.04.

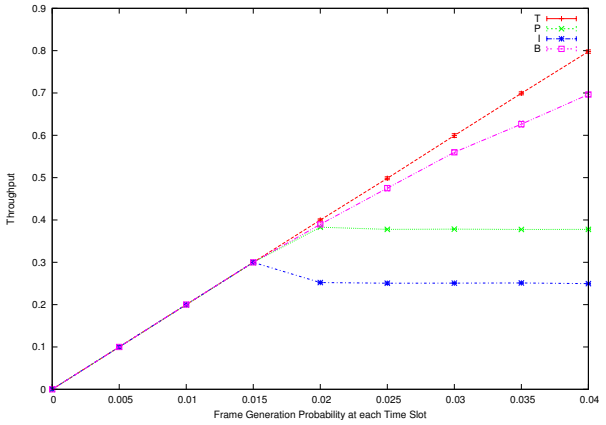

Figure 2: Mean throughput of all stations under each protocol for p:(0,0.04; @0.005 increments). Used to illustrate what happens as p nears 0.04.

GNUPLOT for plotting.

Results

Describe results TODO: tables of undelivered/generated

Discussion

Explain results, contrast with expectations

—— NB - One thing to note could be limitations of our simulation - a global probability constant for frame generation assumes that all stations are providing the same level of load to the hub, which may not necessarily be true and could possibly have significant repercussions on fairness, ie if most stations are pretty quiet, but one is SUPER active, could it

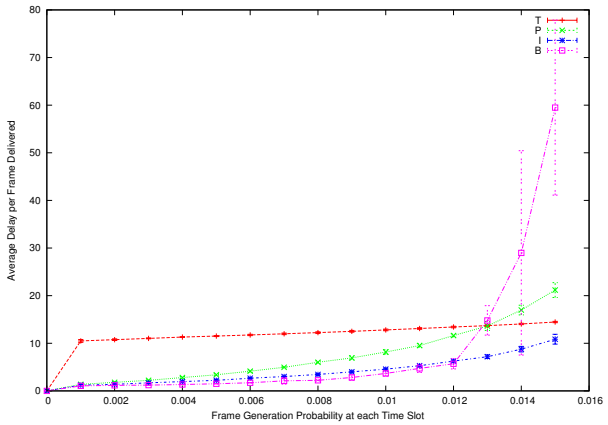

Figure 3: Average delay per frame under each protocol for p:(0,0.015; @0.001 increments). Used to illustrate what happens as p nears 0.

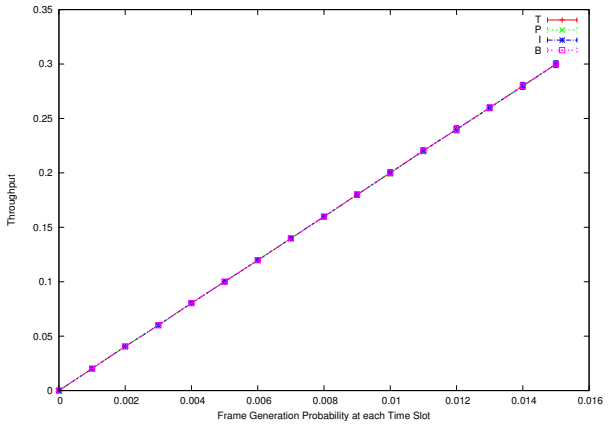

Figure 4: Mean throughput of all stations under each protocol for p:(0,0.015; @0.001 increments). Used to illustrate what happens as p nears 0.

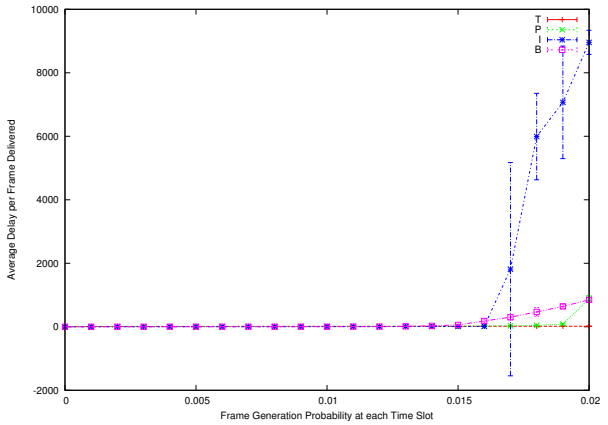

Figure 5: Average delay per frame under each protocol for p:(0,0.02; @0.001 increments). Used to illustrate where protocols begin to diverge significantly. Especially, protocol I at p=0.015.

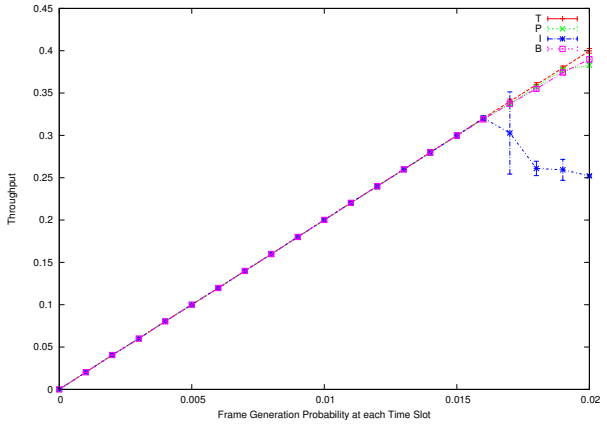

Figure 6: Mean throughput of all stations under each protocol for p:(0,0.02; @0.001 increments). Used to illustrate where protocols begin to diverge significantly. Especially, protocol I at p=0.016.

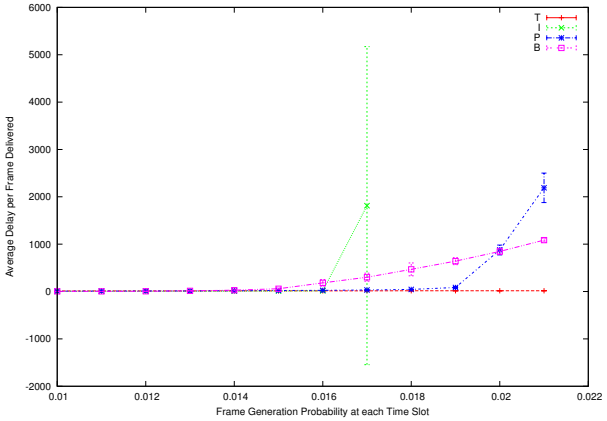

Figure 7: Average delay per frame under each protocol for p:(0.01,0.021; @0.001 increments). Used to illustrate the points where the protocols intersect.

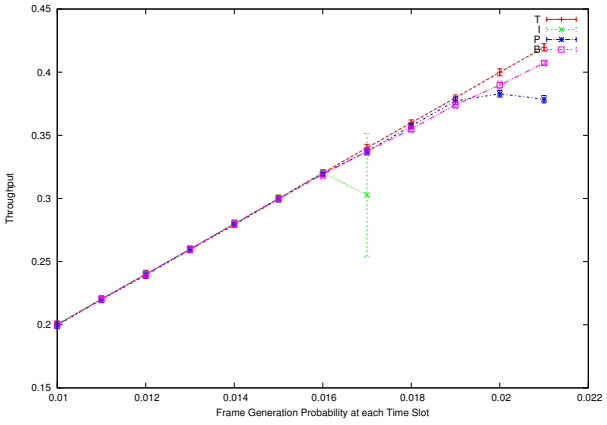

Figure 8: Mean throughput of all stations under each protocol for p:(0.01,0.021; @0.001 increments). Used to illustrate the points where the throughput for protocols I and P begins to level off.

drown out the others in the various protocols in a way that is somehow unfair? —

Conclusions

Rehash discussion?

(Under these applcations, use this protocol! Under others, use that!)