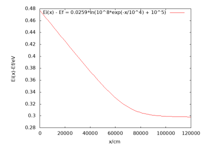
Project2 Report

Ye Feiyang Li Jianliang Yuan Xiaojie

July 25, 2012



Introduction

In the MAC sublayer, Random Access Control is a very important part. There are merely four kinds of Random Access Control protocols: pure ALOHA, slotted ALOHA, CSMA and CSMA/CD. This project will simulate these four protocols to see their transmission efficiency and compare their performance. By doing this, it can be determined which protocol is suitable for a certain situation and helps us to understand how each protocol works.

Pure ALOHA

Theories

ALOHA was devised by Norman Abramson in the 1970s to solve the channel allocation problem. The basic idea of pure ALOHA is that users(or stations) can transimit frames whenever they have, which is totally random. Of course, when different users are transmiting their frames at the same time, those frames will collide and all of them will be damaged. A sender can find out whether its transmission succeed by checking the acknowledgements from the receiver. If the frame was damadged, it waits for a random time and send again.

Given the following parameters:

X: frame transmission time

N: average # of frames genereted per frame time

G: load, average # of transmission attempts per frame time

k: # of transmissions attempts per frame time

P: probablity of a successful frame transmission

S: throughput, average # of successful frames per frame time

We can derive that the probability that k frames are generated during a frame time X is given by the Poisson distribution

$$P(k) = \frac{G^k e^{-G}}{k!}$$

So the probability of zero frames during the $2G$ vulnerable time is

$$P(0)|_{G=2G} = e^{-2G}$$

Then the throughput is given by

$$S = Ge^{-2G}$$

for which the maximum occurs at $G = 0.5$ with $S = 1/2e \approx 0.184$ and the $S - G$ graph is shown on Figure 1.

Assumptions

1. The length of each frame remains the same.
2. There is no propagation delay which means that one user can know whether its frames transmission succeed as soon as it finished its transimission.
3. The probability of an arrival during a short time interval Δt is propotional to the length of interval, and does not depend on the origin of the interval.
4. The probability of having multiple(> 1) arrivals during a short time interval Δt approaches 0;

Simulation

In the simulation program, an two-dimensional array containing 0s and 1s is used to represent states of different users at each short time interval Δt. For example, statest[$userNum$][$slotNum$] is an $userNum \times slotNum$ array, its Nth($0 <$ N $< userNum$) row represents that the Nth user's states from 0th to ($slotNum$-1)th short time interval. "0" is used when the user is not transmitting data at a certain short time interval and "1" is used otherwise.

The array can be expressed graphically like this:

states[N][M]:

user1: 111111000111111$\cdots$0000000001111110000000 user2: 001111110000000$\cdots$0001111110000000000111

$\vdots$ $\vdots$

userN: $\underbrace{000000011111100\cdots 1111110000001111110000}_{\text{M short time intervals}}$

Since the frame length(frameLen) is assumed to be constant, the number of consetutive 1s remains the same(except the frames at the end of each row). The above example always has 4 consetutive 1s which represents a 6-slot length frame.

At the beginnig of each simulation, the array need to be initialized based on a given probability p which is the probability that a user generates a frame during a certain short time interval Δt. Once the first state is initialized to be 1, its consetutive (frameLen - 1) states should all be set to 1. This initialization process is conducted by the function *generate_frame()*.

Then at each short time interval, the states of each user are checked and using function *check_collision*, three status(*COLLISION*, *NO_COLLISION* and *IDLE*) are marked.

If a collision is detected for a user, it pushs all its 1s and 0s backwards for a random time using function *wait_for_random_time()*. To be noticed, the random time should be between 0 and the maximum simulate time and it is set to *SIMULATE_TIME*/2 in the codes.

Finally, the driven function *pure_aloha_simulate()* is called in *main*() to simulate and output the throughput *S*.

Results

Several global parameters are set as below:
USER_NUM = 10
SIMULATE_TIME = 1000
FRAME_LEN = 4

Then the propability p are set differently to archive various G, and for each G, the simulation is run 1000 times to get an average S. The data are recorded in the following table.

p	0.01	0.02	0.03	0.04	0.05	0.10	0.15	0.20	0.30
successNum									

The number of attempts per frame time G is calculated as $G = p * USER_NUM$ and throughput $S$is calculated as $S = successNum/250$. The data are recorded in the following table.

G	0.1	0.2	0.3	0.4	0.5	1.0	1.5	2.0	3.0
S									

Analysis

From the data above, the maximum value of S is approximately when G is . Unfortunately, the result doesn't match well with the theory with $S_{max} = 0.184$ at $G = 0.5$. This may come from some inevitable drawbacks of the simulation model.

For example, the maximum random wait time is set to be *SIMULATE_TIME/2* and this may be a too long time so that when many users wait out of the simulate bound, the remaining users can almost send all of their frames.

Slotted ALOHA

Theories

Slotted ALOHA is an advanced version of pure ALOHA. It divides time into descrete slots that each frame can only be sent at the beginning of each time slot. Moreover, slotted ALOHA requires global time synchronization so that the same slot boundaries can be agreed by all the users.
The vulnerable time for slotted ALOHA is half of the one for pure ALOHA. So the probability of zero frames during the G vulnerable time is

$$P(0)|_{G=G} = e^{-G}$$

Then the throughput is given by

$$S = Ge^{-G}$$

for which the maximum occurs at $G = 1$ with $S = 1/e \approx 0.368$ and the $S - G$ graph is shown on Figure 1.

Assumptions

1. Each frame can only be sent at the beginning of each time slot.
2. All of the rest are same as pure ALOHA.

Simulation

Results

Table for p and $successNum$ is shown below.
Table for G and S is shown below.

Analysis

Similarly, the results of slotted ALOHA also doesn't match well with the theory with $S_{max} = 0.368$ at $G = 1.0$

However, fortunately, compared with data of pure ALOHA, the value of S of slotted ALOHA is always larger regardless of the value of G. This phenomenon shows that slotted ALOHA has higher efficiency that pure ALOHA.

CSMA

CSMA/CD

Theories

At a given moment, user A has a packet to send. Before starting transmission, it checks whether the medium is idle or not. If the medium is busy, the user waits a random time and then tries again. If the medium is idle, the user starts to transmit, and keeps sensing the medium. Once a collision signal is detected, rather than finishing sending the packet, the station stops sending immediately and waits for the next attempt.

For choosing the random time, the protocol uses the Binary Exponential Backoff Algorithm. According to the algorithm, time is divided into discrete slots whose length is equal to the worst-case round-trip propagation time on the ether. After the first collision, the station waits either 0 or 1 slot time before another try. After the second collision, the station chooses either 0, 1, 2 or 3 slots randomly and waits that number of slot time. Thus, by the same rule, after kth collision, the number of time slots to wait is chosen randomly from the interval 0 to $2^k - 1$.

Theoretically, the maximum throughput is given by:

$$\rho_{max} = \frac{X}{X + t_{prop} + 2et_{prop}} = \frac{1}{1 + (2e + 1)a}$$

where $a = \dfrac{t_{prop}}{X} = \dfrac{Rd}{vL}$
(R: transmission rate, L: frame length, v: light speed in medium, d: diameter of system, X: frame time).

Assumptions

1. The propagation time is assumed to be the smallest time unit in the entire design and set to be 1 time slot.
2. The "medium" is set to be a station with 1 time slot delay to all other stations.
3. The duration of the simulation is in the unit of time slots.
4. The states of all users during the whole process are saved in a large array and the size of it is the number of users times the duration.

Simulation

The Simplified Procedure of the Program:
1) The program user sets the number of stations, the probability of a station having a frame to send and the duration of the program.

2) The program forms a new array and fills it with numbers other than 0 and 1.

3) Using the function "setFrame()" to make the array initialized with 0s and 1s. If the frame time is set to be 4, then every 4 consecutive 1s at a position of the array means a certain station at that time will have a frame to send.

4) The program set a "for" loop that repeats the value of duration times. In each loop, it contains another "for" loop, which repeats the number of stations times. In each small loop, the program checks the state of one station.

(i) If the station has something to send and the medium is idle, then the station begins to send and the data it is sending will be received by the medium in the next larger loop.

(ii) If the station has something to send and the medium is busy but has no collisions, it first checks whether it sent data in the last two loops. If true, it keeps sending. If false, it waits a random number of slots based on the Binary Exponential Backoff Algorithm. The act Waiting is realized by postponing the value in the array by a given slots. Values, which are moved out of bound, are just discarded and the newly - produced empty slots are filled by 0s.

(iii) If the station has something to send and the medium is busy and has collisions, it does the same backoff strategy as in the previous part.

(iv) In all other states, the station does nothing.

(v) After each small loop, the program checks the state of the medium and report the change by changing values of several parameters. These parameters will be checked at the first of next large loop in order to simulate the one-slot delay.

(vi) Another array with comparatively small size will be used to record the number of successively transmitted frames of each station.

Results

Analysis

From the figures of Sample 1 and Sample 4, which show the difference between experimental and theoretical results, it can be inferred by both curves that the highest transmission efficiency happens when $n = 1/p$, and $G = 1$. However, there are some obvious errors between two results. One of them is that the simulation cannot reach as high efficiency as in theory. The most important reason may be that in the process of calculating the theoretical result, it doesnt take the influence of random waiting time into account because the theoretical result is just based on the probability of just one attempt per frame time. So with different scope of random waiting slots, the result may differ a lot.

Another big difference is that according to the theory, the efficiency at the point $p = 1$ should be zero. But in the simulation, the value is a positive integer. It is quite reasonable to see that because since the number of slots ranges from 0 to a number larger than half of the entire duration. If some of stations just wait a few slots while others wait much more time, then these stations would have enough time to finish their transmission. But this kind of transmission will cause a very large delay, which means that it is not suitable for many heavy-traffic users.

Between the two figures, we can see that the one with smaller A (propagation time divided by frame time) has better performance, which is the same as illustrated in the textbook.

In the second simulation, which compares the program both with and without BEBA. From the figure, it is clear that the one with BEBA performs much better than the one without BEBA. And instead of having bad performance, the one with BEBA has a better efficiency when p increases. Thus as a conclusion of the result of simulation, it is better to choose BEBA as the algorithm of setting random waiting time when the traffic is very heavy.

For the third simulation, it tells that when user numbers are larger, the performance will be better. This result is expected because when calculating the maximum efficiency, an approximation is made at $(1-\frac{1}{n})^{n-1}$ when n is large enough. However, one problem emerges. In the theory, the probability of a transmission success is $P_s = np(1-p)^{n-1}$, and the maximum value happens when $n = \frac{1}{p}$. Thus the peak value of S should happens at $p = \frac{1}{n}$, but in the figure, the two peak values of different curves happen at the same point. This error may be caused by the scale of simulation or some little errors in the design of simulation.

Comparing to ALOHA and Slotted ALOHA, CSMA has greatly improved the efficiency of transmission. However, a user who is sending a frame needs time, which is twice of the propagation time, to capture the medium. During this small time interval, other users who happen to have packets to send may regard the medium as idle and begins to transit and thus destroys both packets. And with the increasing of propagation time caused by longer distance or lower transmission rate, this kind of fault will have larger effect and reduces the efficiency rapidly.

In order to minimize the influence of problem mentioned above, CSMA/CD (collision detection) is invented. It tells users whether their packets are having collisions with other packets or not. If true, both the users stop transmission to save time and wait for another chance.