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$00000000011111110000000

user2: 001111110000000$\cdots$00011111110000000000111

$\vdots$ $\qquad\qquad\qquad$ $\vdots$

userN: $\underbrace{000000011111100\cdots1111110000001111110000}_{\text{M short time intervals}}$

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 and the figure is shown on Figure 1.

p	0	0.01	0.02	0.03	0.04	0.05	0.10	0.15	0.20	0.30
successNum	0	45	52	52	53	56	61	67	66	74

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 and the figure is shown on Figure 2.

G	0	0.1	0.2	0.3	0.4	0.5	1.0	1.5	2.0	3.0
S	0	0.183	0.209	0.211	0.215	0.224	0.247	0.269	0.268	0.300

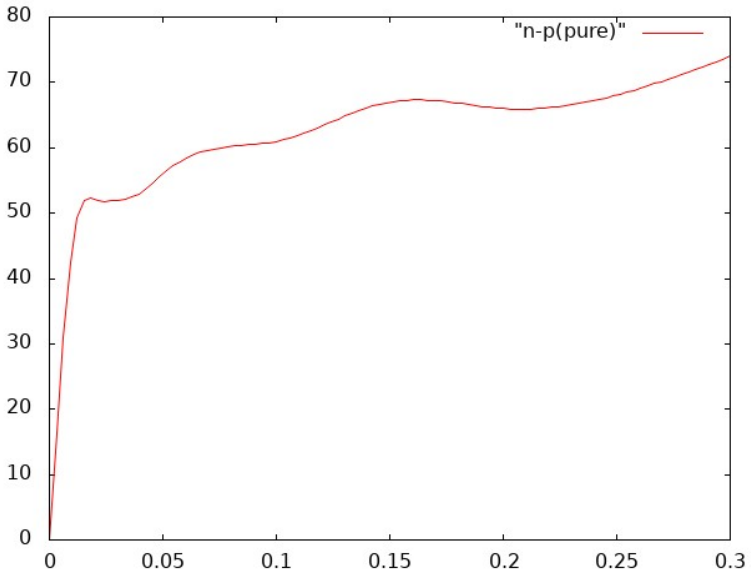

Figure 1: successNum-p

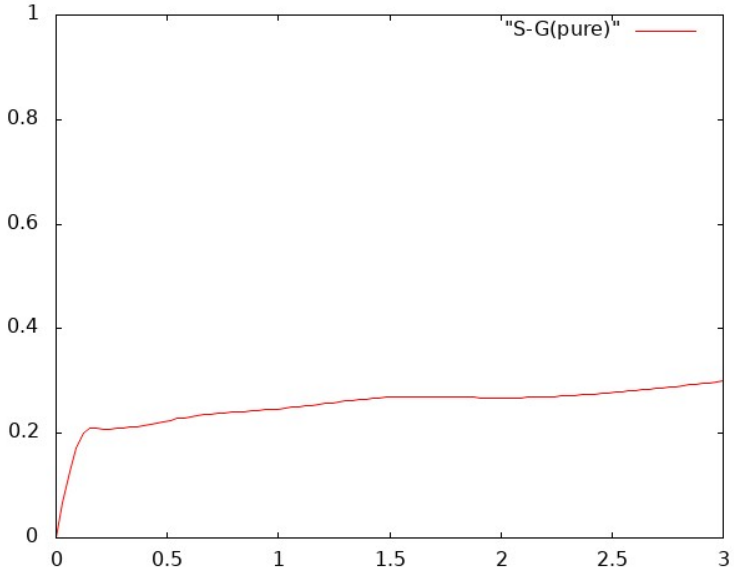

Figure 2: S-G

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.

p	0	0.01	0.02	0.03	0.04	0.05	0.10	0.15	0.20	0.30
Num	0	65	85	94	97	92	80	73	69	65

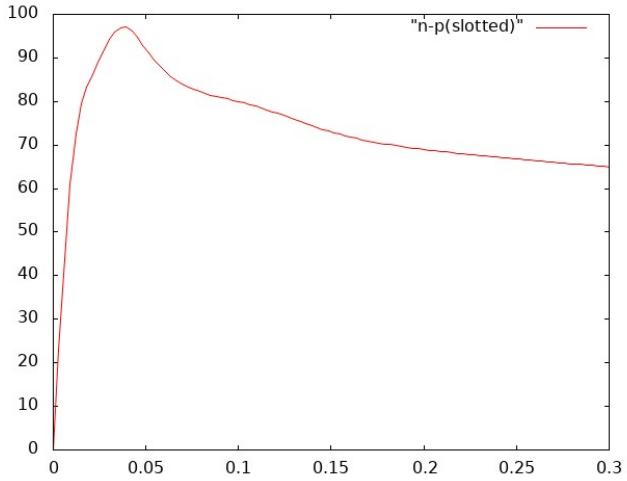

Figure 3:

G	0	0.1	0.2	0.3	0.4	0.5	1.0	1.5	2.0	3.0
S	0	0.262	0.341	0.377	0.391	0.369	0.323	0.292	0.280	0.263

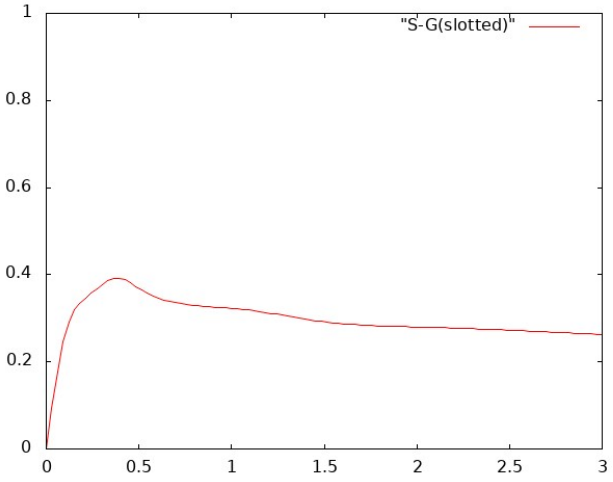

Figure 4:

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

Theories

Carrier Sense Multiple Access (CSMA) is a probabilistic Media Access Control (MAC) protocol in which a node verifies the absence of other traffic before transmitting on a shared transmission medium, such as an electrical bus, or a band of the electromagnetic spectrum.
"Carrier Sense" describes the fact that a transmitter uses feedback from a receiver that detects a carrier wave before trying to send. That is, it tries to detect the presence of an encoded signal from another station before attempting to transmit. If a carrier is sensed, the station waits for the transmission in progress to finish before initiating its own transmission. In other words, CSMA is based on the principle "sense before transmit" or "listen before talk".
"Multiple Access" describes the fact that multiple stations send and receive on the medium. Transmissions by one node are generally received by all other stations using the medium.
CSMA protocol has three access modes: 1-persistent, non-persistent and p-persistent.
1. 1-persistent
When the sender (station) is ready to transmit data, it checks if the transmission medium is busy. If so, it then senses the medium continually until it becomes idle, and then it transmits the message (a frame). In case of a collision, the sender waits for a random period of time and attempts to transmit again.
2. Non-persistent
Non persistent CSMA is less aggressive compared to P persistent protocol. In this protocol, before sending the data, the station senses the channel and if the channel is idle it starts transmitting the data. But if the channel is busy, the station does not continuously sense it but instead of that it waits for random amount of time and repeats the algorithm. Here the algorithm leads to better channel utilization but also results in longer delay compared to 1 -persistent.

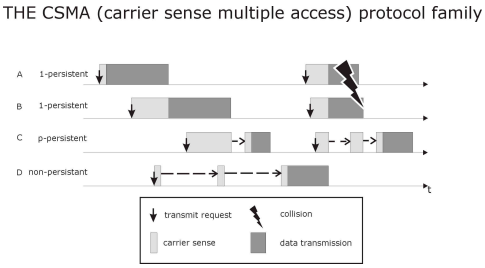

Figure 5: Simple Graphical Description

3. P-persistent
This is a sort of trade-off between 1 and non-persistent CSMA access modes. When the sender is ready to send data, it checks continually if the medium is busy. If the medium becomes idle, the sender transmits a frame with a probability p. If the station chooses not to transmit (the probability of this event is 1-p), the sender waits until the next available time slot and transmits again with the same probability p. This process repeats until the frame is sent or some other sender starts transmitting. In the latter case the sender monitors the channel, and when idle, transmits with a probability p, and so on. p-persistent CSMA is used in CSMA/CA systems including WiFi and other packet radio systems.

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) The length of a frame stays a static constant the whole simulation process which is 3, i.e.
4) The random waiting time is set no larger than 10 and at least 3, which is the length of a frame.
5) The program checks to see whether the medium is idle or busy and decides which station to transmit frames.
i) If the station has something to transmit and the medium is idle without ant collisions, then the station begins to send frames.
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 second. If true, it keeps sending. If false, it waits a random number of seconds.
iii) If the station has something to send and the medium is busy and has collisions, it senses the medium continuously (per second) to send frames.