Data Communication : Simulation Homework Report

공과대학 컴퓨터공학부
2013-11412 오평석

1. Introduction

CSMA는 Carrier sensing 기법을 이용하여 channel이 idle인지 판단한 다음, 서로 경쟁하며 channel을 사용하는 Multiple Access Scheme이다. 현재 IEEE 802.3 MAC 에서 사용하는 scheme인 CSMA/CD는 기존의 CSMA scheme에서 CD(Collision-Detection)과 binary exponential backoff를 도입하여 다양한 load에서의 performance를 개선하였다. 이번 과제에서는 시뮬레이션을 통해 CSMA + uniform backoff, CSMA/CD + uniform backoff, CSMA/CD + binary exponential backoff scheme의 performance 차이를 알아보고 분석해보고자 한다.

2. System description

(1) 개발 환경

```
OS : Ubuntu 14.04.4 LTS (in vmware)
Language : C++
Compiler : g++ 4.8.5
```

다만 과제를 하던 도중 노트북이 고장이 나서 시뮬레이션의 최종 output은 Window 환경의 MSVC를 이용하여 얻었다. 하지만 MSVC의 독자 규격을 이용한 부분은 하나도 없으므로 output은 큰 차이가 없을 것으로 생각한다.

(2) 프로그램 실행 방법

프로그램을 실행하기 위해서는 첨부한 소스를 컴파일하여야 한다. 컴파일 하는 방법은 channel.cpp, node.cpp, simulator.cpp, main.cpp의 object 파일을 생성 후 이를 linking하여 실행 파일을 얻는다. 이 과정을 쉽게 하기 위하여 Makefile을 같이 넣어두었으며, 따라서

```
cseteram@ubuntu:~/Works/CSMASimulator$ make
```

를 통하여 컴파일 가능하다.

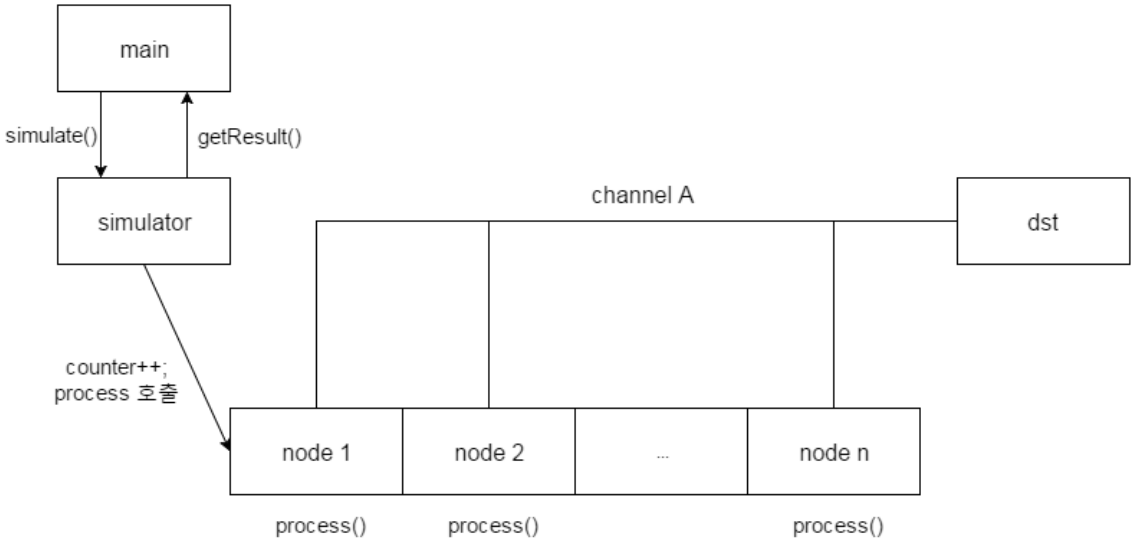
소스 코드는 메일에 첨부하였으며, [Appendix B] 에서도 확인할 수 있다.

(3) 프로그램 구성

프로그램은 크게 3가지 크게 channel, node, simulator로 구성된다. 프로그램의 전체적인 흐름은 다음과 같다.

(1) simulator에서 channel과 node를 만든 다음, channel을 node에 할당해준다.
(2) 그 이후, 현재 카운터를 1 증가시킨 다음 각 노드들의 process 함수를 모두 호출한다.
(3) process 함수가 호출되면, 노드는 현재 상황을 분석하여 알맞는 행동을 한다.

(4) 모든 process 함수가 종료되면, simulator는 collision이 일어났는지 감지한다.
(5) (2) ~ (4)의 과정을 duration 동안 시행한다.



counter는 simulation 시작 후 얼마나 시간이 흘렀는지 나타내는 변수이다. 단위는 us이며, 처음에는 real time으로 구현하려 했으나 이렇게 되면 경우에 따라 process의 호출 주기가 달라져 이로 인해 생기는 오차를 다루기 어려워 보였다. 따라서 loop문을 돌며 counter를 1 증가시키는 방식으로 구현하였으며, 각 node의 process는 1us 마다 호출되는 것으로 약속하였다.

또한 최종적으로 performance를 구하기 위해 필요한 통계량들을 저장하는 클래스가 존재한다. 이렇게 프로그램에는 총 4개의 class가 있는데, 각각의 class에서 가지고 있는 attribute와 member function을 설명하면 다음과 같다. (get, set function은 제외함)

(a) channel class
node들이 데이터를 전송할 때 사용할 channel을 나타내는 클래스이다.

* attributes
state : 현재 채널이 idle인지 busy인지 가리킨다. (idle이면 0, busy이면 1)
nodes : 이 채널을 사용하고 있는 node들을 저장한다. 만약 어떤 시점에서 이 채널에 collision이 발생하였다면, nodes에 저장된 node들이 2개 이상인지 판단하면 된다.

* member functions
add(Node *node) : nodes에 node를 저장한다. 이는 Node가 이 채널을 이용하여 데이터 전송을 시작할 때 호출하는 함수이다.
remove(Node *node) : nodes에서 node를 제거한다. 이는 Node가 데이터 전송을 모두 끝내어 이 채널을 더 이상 사용하지 않을 때 호출하는 함수이다.

(b) node class

데이터를 전송하고자 하는 node를 나타내는 클래스이다. node는 node의 현재 상태, backoff 전략, 행동을 하기 위해 기다려야 하는 counter 등을 보고 현재 상황에서 해야 할 일을 판단하여 행동한다.

* attributes

state : node의 현재 상태를 나타낸다.
strategy : 충돌이 일어났을 때 backoff 전략을 나타낸다. (0이면 uniform, 1이면 binary exp.)
cw : 주어진 CW 값을 저장한다. 만약 backoff 전략이 binary exp. 라면, 이 값은 노드 생성 후 바뀔 수 있다.
leftCounter : node는 현재 상태에서 다음 상태로 전이할 때 어느정도 기다려야 하는 경우도 있다. (예를 들면, packet을 생성하는 중이라면 packet 생성 시간만큼 지나야 전송 준비 상태로 넘어갈 수 있다.) leftCounter는 이러한 값을 저장하는 변수이다.
leftCollisionDetection : node는 데이터 전송 시작 시 collisoin을 판단하는 시간 (50us) 이 존재한다. 이 변수는 collision을 판단하는 시간이 얼마나 남았는지 저장해둔다. 이 값이 0이 되면, node는 자신이 사용하고 있는 channel을 busy로 마킹한다.
stats : 시뮬레이션 시 얻는 통계량을 이 변수에 기록해둔다.
channel : 이 node에 할당된 channel을 나타낸다.
lastGeneratedCount : packet의 생성이 완료되었을 때 counter를 저장해두는 변수이다.

* member functions

generate(void) : node가 packet을 생성할 때 호출하는 함수이다.
process(void) : 1us 마다 호출되는 함수로, node가 지금 상황을 분석하고 적절한 행동을 한 후 다음 상태로 전이한다.

process 함수를 구현하는 일반적인 방법은 상태전이 테이블을 만든 다음 FSM(Finite State Machine)을 구성하는 것이나, 본 시뮬레이션에서 node가 가지는 state와 상태전이 테이블은 크지 않다. 그래서 실제 구현은 다음 상태로 전이할 때 까지 남은 시간을 count한 다음, 그 시간이 0이 되면 현재 상태에서 몇 가지 작업을 한 후 다음 상태로 바꾸는 식으로 구현하였다. 대략적인 코드는 다음과 같다.

```
void Node:process(void)
{
    if (--leftCounter)
        return;

    /* 현재 상태가 패킷 생성 중이라면
     * 전송 준비 상태로 전이하고, 생성이 완료된 시점의 counter를 기록해둔다
     * /

    (이하 생략)
}
```

(c) simulator class

channel과 node를 생성하고, 주어진 scheme에 따라 simulation을 하는 simulator 클래스이다.

main 함수에서는 simulator을 생성한 다음 simulate()를 호출하는데, 이 함수는 counter를 1 증가시킨 후 각 노드의 process를 호출한다. 그리고 channel에서 충돌이 일어났는지 판단한다.

* attributes

scheme : 시뮬레이션을 할 MAC scheme을 나타낸다.
leftSimulationCounter : 시뮬레이션을 할 counter이다. 이 값이 0이 되면, simulation이 종료된다.
nodeCnt : node 갯수
initialCW : cw값. backoff 전략이 uniform이라면 maximum range가 들어가고, binary exp. 라면 처음 cw값인 2가 들어간다.
nodes : simulation에 사용할 node를 저장해둔다.
throughput, meanPacketDelay, transmissionCollisionProb : 우리가 얻고자 하는 output임

* member functions

simulate(void) : simulation을 하고 measure를 계산하여 저장한다.
getResult(void) : throughput, mean packet delay, transmission collsion prob. 를 얻는다.

(d) Statistic class

각종 통계량을 저장하는 클래스이다.

attribute로는 현재 counter, 전송에 성공한 패킷 수, packetDelay의 합, collision이 일어난 전송 수, 전송 횟수 등을 저장한다.

member function으로는 이러한 통계량들을 얻고 갱신한 데 필요한 helper function들을 제공한다.

3. Simulation model

(1) Spec에 명시된 부분

* MAC schemes
  1-persistent CSMA + Uniform distributed backoff within [1, CW]
  1-persistent CSMA + CD + Uniform distributed backoff within [1, CW]
  1-persistent CSMA + CD + Binary exponential backoff

* Uniform distributed backoff within [1, CW]
  CW : a constant

* Binary exponential backoff
  For each collision detection, CW <- 2 * CW
  Value of CW at the first retransmission : 2

* Packet generation at each node
  Time interval : -E[T] * ln x (0 < x < 1)

* Parameter Values
  Packet size : 1024bytes
  Transmission rate : 10Mbps
  Slot time : 50us
  Bit error probability : 0
  Time interval until packet generation E[T] : 10ms
  CW of uniform distributed backoff : CW = 32, 64, 128

* Output
  Throughput : the mean number of transmitted packets per second
  Mean packet delay : the time from a generation time of a packet to the time that the transmission of the packet is completed.
  Transmission collision probability

(2) Spec과 다르거나 Spec에 명시되지 않아 가정한 부분

* node 개수, 시도 횟수
  spec에서는 node 개수가 5, 10, 15, 20, 25로 총 5가지에 대하여 시뮬레이션을 돌리면 되는 것으로 나와 있으나, node 개수가 5개만 있어서는 그래프를 그리기 힘들었다. 그래서 node 개수의 간격을 더 작게 잡아서 1 ~ 25 사이의 node 개수에 대하여 모두 simulation하였다.
  그리고 각 (scheme, N, CW)에 대하여 10번 반복해서 simulation을 돌리고, 평균을 내어 최종 결과를 얻었다.

  최종적으로 simulation을 돌린 횟수는 10 * (25 * 3 * 2 + 25) = 1750번이며, 결과가 모두 나오기까지는 약 80분 정도가 걸렸다.

* Propagation delay 무시
  node간의 거리로 인하여 발생하는 propagation delay는 무시하였다.

* cw값 초기화
  Binary exponential backoff에서 전송에 성공한 경우, 그 노드의 cw값은 2로 초기화된다.

* cw값 상한
  Binary exponential backoff에서 collision이 일어나면 cw값이 2배씩 된다. 이 때 spec에서는 cw의 상한값이 주어져있지 않다. 상한이 없으면 performance에서 손해가 될 것이라고 생각하여 cw의 상한을 1024로 두었다.

* ACK 및 타임 아웃
  CSMA + Uniform distribution backoff scheme에서는, 패킷을 전송하다가 ACK가 돌아오지 않아서 타임 아웃이 나는 경우 collision이라고 판단하였다. 그런데 시뮬레이터 프로그램을 작성하면서 세운 가설은, ACK의 크기나 전송 시간, 그리고 타임 아웃이라고 판단하는 interval은 결과에 dominant하게 영향을 끼치지 않는다는 것이다. 그래서 ACK는 모든 패킷을 전송하면 그 즉시 돌아오며, 데이터를 전송한 노드 입장에서 모든 데이터를 전송했는데 그 타임 슬롯 내에 ACK가 돌아오지 않으면 time out이라고 판단하도록 프로그램을 작성하였다. 굳이 수치화를 한다면, ACK의 크기는 매우 작고, 전송은 매우 빠르게 일어나며, ACK를 전송하면서 일어나는 충돌은 고려하지 않는다는 것이다. 또한, time out이라고 판단하는 interval은 한 타임 슬롯이므로 50us라고 적을 수 있을 것이다.

* 패킷 생성 시, 생성에 걸리는 시간을 매번 새로 계산함
  각 node별로 패킷을 생성할 때, 패킷을 생성할 때 마다 -E(x) * ln(x)를 새로 계산하여 넣어준다. 즉, 같은 노드라도 패킷 i와 패킷 $i+1$의 생성시간은 차이가 있다.

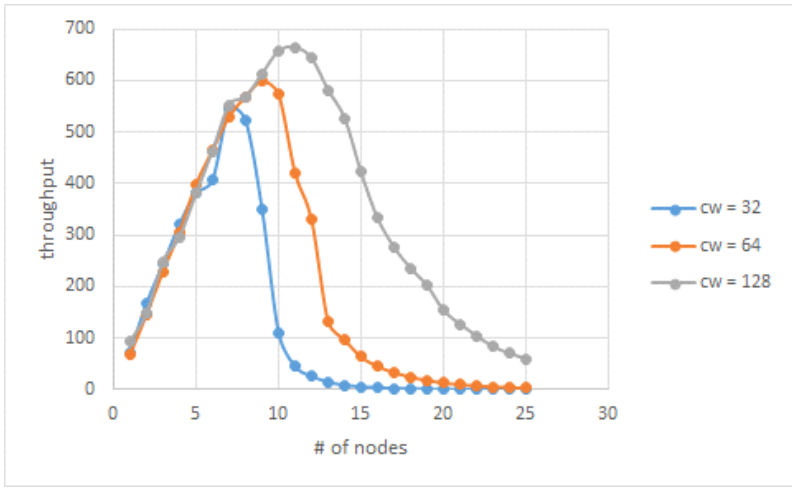
* collision detection time
  어떤 노드에서 전송을 시작할 때 collision을 감지하는 시간이 존재하여, 이 시간동안 다른 노드가 전송을 시도하면 collision으로 판단한다. 이 시간은 슬롯 타임 하나인 50us로 설정하였다.

4. Numerical results with discussion

전체 데이터는 [Appendix A]에서 확인할 수 있다.
다음은 얻은 데이터를 바탕으로 하여 그린 그래프이다.

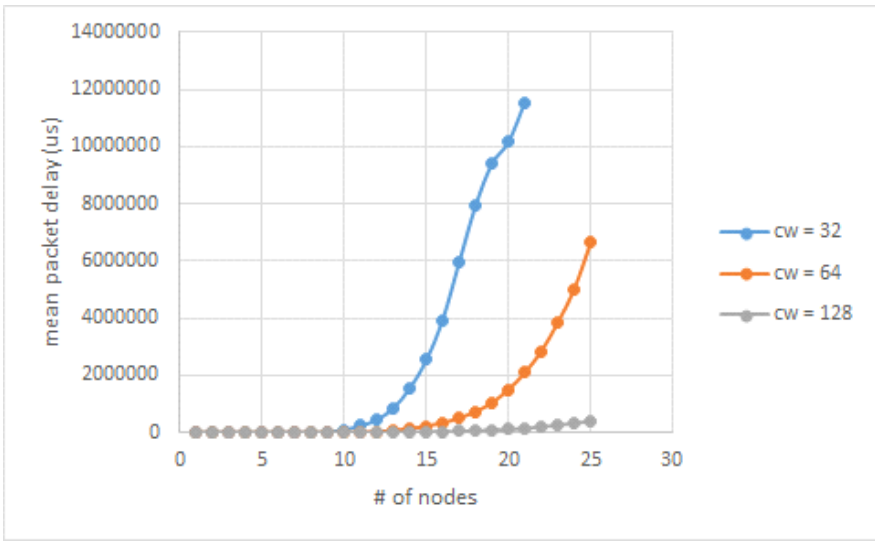
(1) CSMA + Uniform
n에 대한 throughput, mean packet delay, collision probability를 그려보면 다음과 같다.

n에 대한 throughput



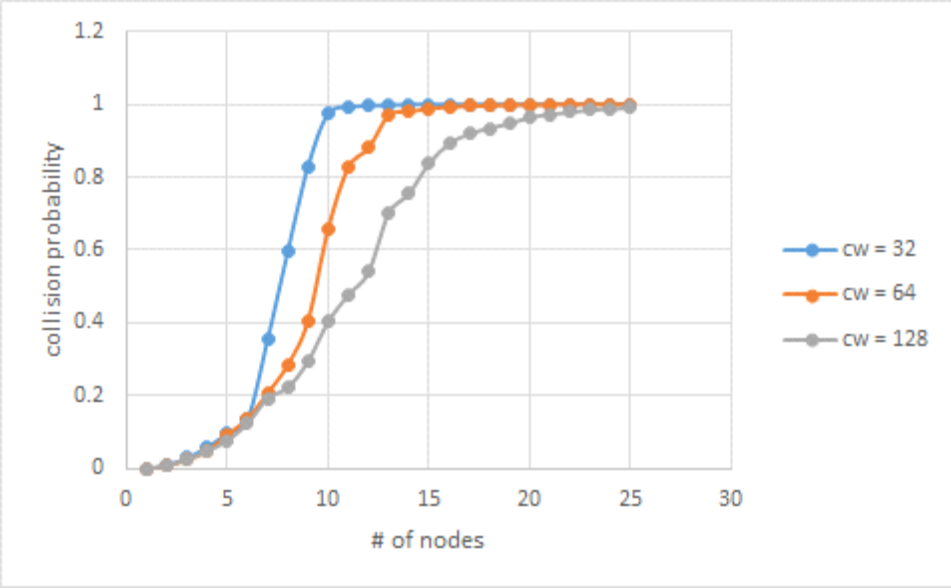
n이 증가하면 node 수가 많아지므로 전송하고자 하는 패킷이 늘어나 throughput이 증가한다. 그러다가 n이 너무 많이 커지게 되면 node 사이의 collision이 늘어나 throughput이 아주 큰 폭으로 감소하게 된다. CW값이 커지면 collision 확률이 줄어들어 조금 더 큰 n에 대하여 저항할 수 있는 기회를 얻게 되며, throughput 값도 더 크고, 극대점을 가지는 n도 더 큰 것을 알 수 있다.

n에 대한 mean packet delay



n이 커지게 되면 collision이 늘어나서 전송을 실패하는 경우가 많아지므로 mean delay packet 역시 증가한다. 이 때 CW = 32, 64의 경우 기하급수적으로 커지는 구간을 볼 수 있는데, 이 때의 throughput이 절망적으로 낮은 것으로 이를 유추할 수 있다. 한편, CW = 128의 경우 throughput이 많이 감소하긴 했지만 그래도 50 이상의 값을 유지하고 있어서, mean packet delay역시 큰 폭으로 증가는 하였지만 CW = 32, 64의 경우보다는 증가하는 정도 매우 작다.

n에 대한 collision probability

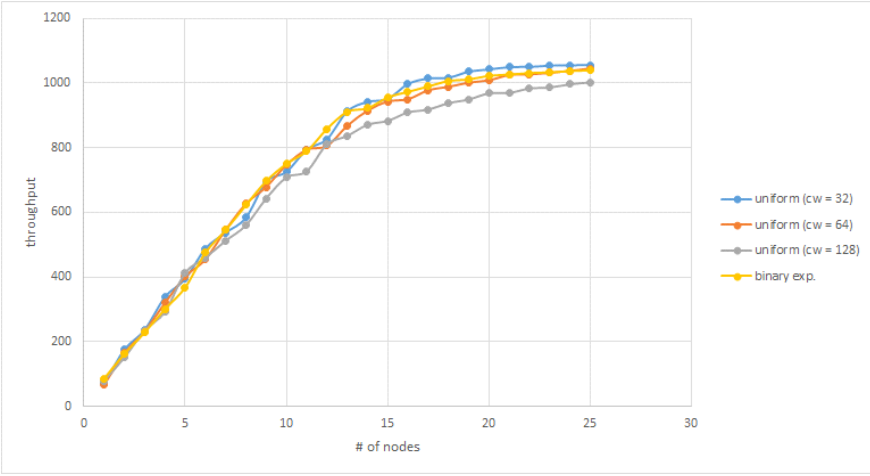


n이 증가할수록 경쟁하는 node 수가 많아지므로 collision probabilty는 증가한다. 이 때 CW값이 크면 경쟁하는 node들 사이의 간격이 넓어지므로 collision이 일어나는 확률 역시 감소함을 알 수 있다.

(2) CSMA + CD + Uniform/Binary exp

n에 대한 throughput, mean packet delay, collision probability를 그려보면 다음과 같다.

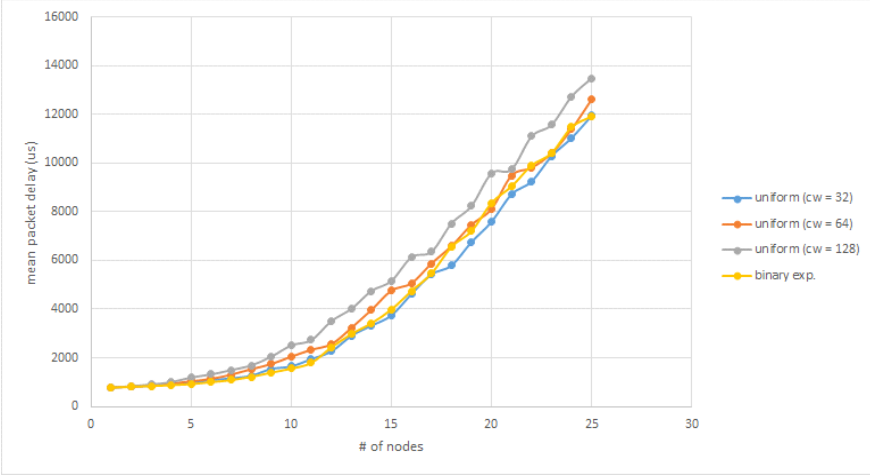
n에 대한 mean packet delay



n이 증가할수록 throughput은 증가하는 경향을 보이다가, n이 너무 커지게 되면 충돌이 자주 일어나 throughput이 증가하지 않고 수렴하는 것을 알 수 있다. 이는 CD를 도입하지 않은 Scheme이 n이 증가하면 throughput이 기하급수적으로 떨어지는 것과 대조적이다.

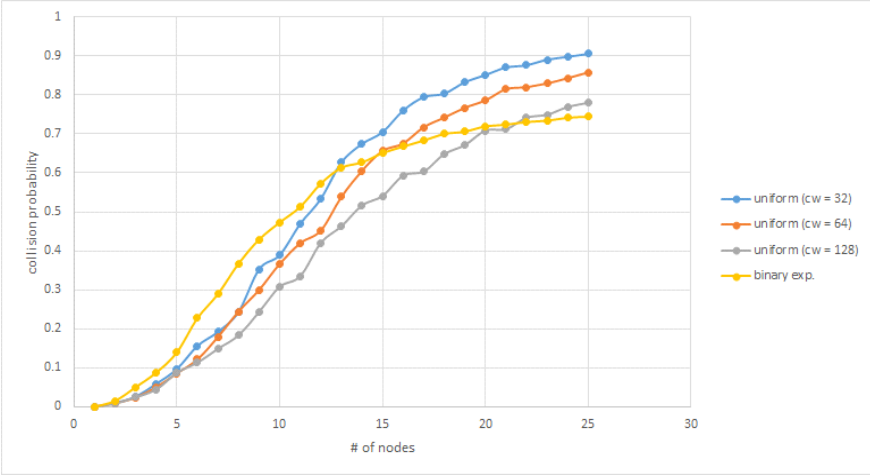
한편 CW가 커지면 충돌 확률이 작아지므로 throughput이 증가할 것으로 예상하였으나 실제 결과는 오히려 약간 감소하였다. 이는 충돌 확률은 줄어들지만 node 사이의 시간 간격은 넓어져서 단위 시간당 보내는 패킷 수가 오히려 감소하였다는 것을 의미한다.

n에 대한 mean packet delay



mean packet delay는 n이 커질수록 증가하는 경향을 보이나, CD를 도입하지 않은 버전과 비교하였을 때 그 폭은 훨씬 작다. 마찬가지로 CW가 커지면 mean packet delay가 작아질 것으로 예상하였지만, 오히려 조금 커졌다. 이유는 throughput의 경우와 같을 것이다.

n에 대한 collision probability



충돌 확률의 경우 n이 커질수록 충돌 확률이 커지는 경향은 같으며, CD를 도입하지 않은 Scheme과 비교하였을 때 같은 n에 대하여 충돌 확률은 조금 더 작았다. (이는 다음 페이지의 그래프에서 확인 가능) 한편 CW가 커지면 node 사이의 경쟁이 완화되므로 충돌 확률 역시 줄어드는 것을 알 수 있었다. binary exp. 의 경우 노드 수가 작을 때에는 충돌 확률이 크지만, 노드 수가 작을 때에는 충돌 확률이 다른 scheme보다 작아지는 것을 확인할 수 있었다.

(3) 종합

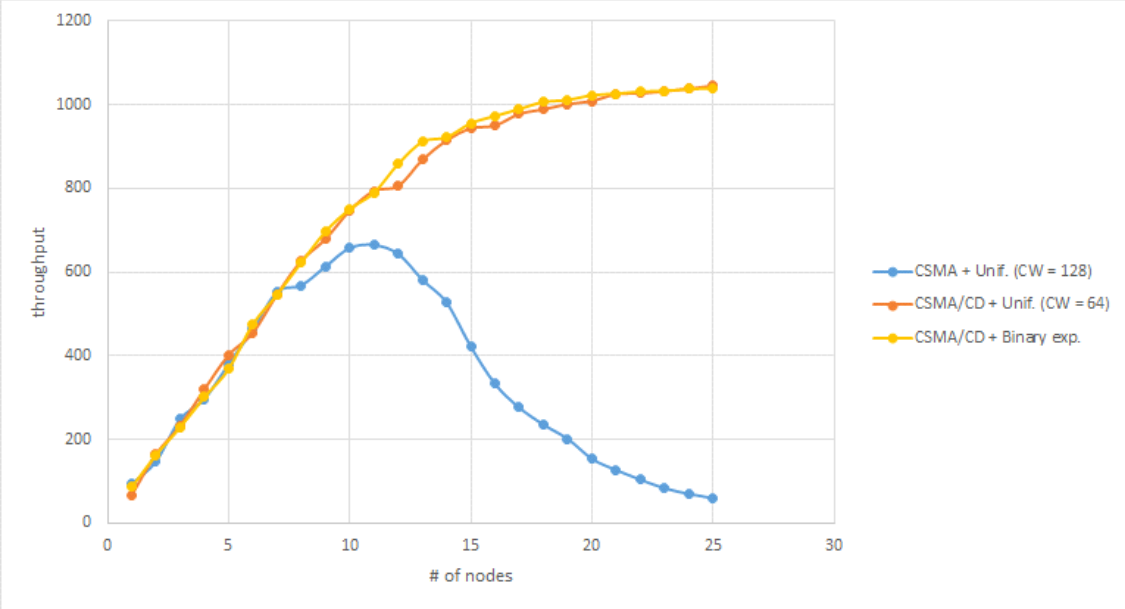
각각의 scheme에서 효율적이라고 생각하는 것들을 꼽아보면 다음과 같다.

CSMA + Uniform (CW = 128)

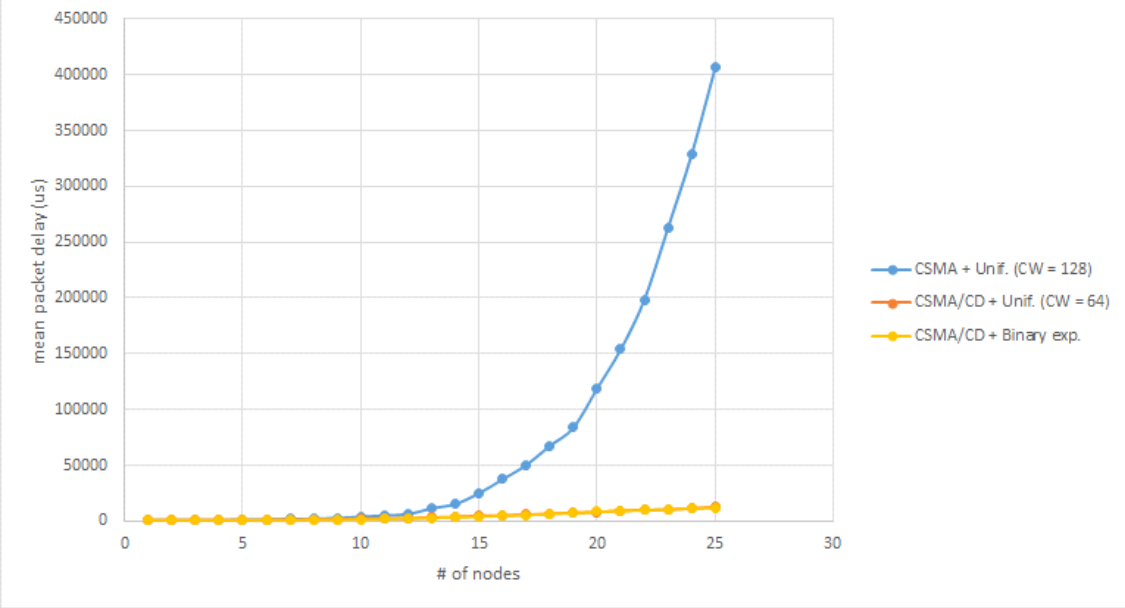
CSMA + CD + Uniform (CW = 64)

CSMA + CD + Binary Exp.

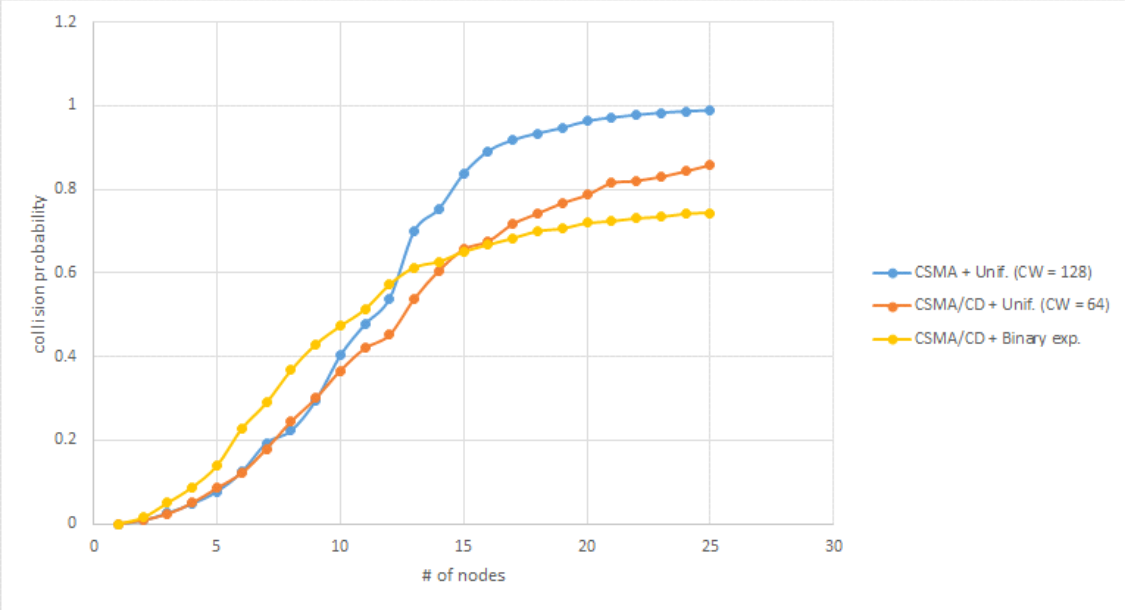
n에 대한 throughput



n에 대한 mean packet delay



n에 대한 collision probability



CD를 도입한 것과 도입하지 않은 것에는 아주 큰 퍼포먼스 차이가 있음을 알 수 있다.

또한, CSMA/CD에서 Unif. 버전과 Binary exp. 버전의 경우 어느 Scheme이 퍼포먼스가 좋다고 말하기는 어렵다. Binary exp. 버전이 큰 n에서 충돌 확률은 작았지만, 작은 n에서는 오히려 높았으며 throughput이나 mean packet delay는 거의 차이를 보이지 않았다.

이는 시뮬레이션을 할 때 n이 한번 정해지면, 그 시뮬레이션을 끝내기 전까지는 노드 개수가 불변인 것 때문이라고 생각한다. Binary exp. 버전의 가장 강력한 강점은 low load나 high load 상관없이 그 때 그 때 CW값을 바꿔가면서, (low일 땐 낮게, high일 땐 높게) 효율적인 퍼포먼스를 보이는 것인데, 이처럼 N값이 고정이 되어버리면 Binary exponential backoff 전략의 장점을 살리기 어렵다. 만약 Binary exponential backoff의 장점을 그래프에서 확인하고 싶었다면 시뮬레이터를 조금 더 고쳐 N이 시뮬레이션 도중 유동적으로 바뀌는, 즉 node가 경쟁에서 빠지거나 다시 참여하는 상황을 추가하면 될 것이다.

5. Conclusion

이상으로 CSMA + uniform backoff, CSMA/CD + uniform backoff, CSMA/CD + binary exponential backoff scheme의 퍼포먼스를 throughput, mean packet delay, collision probability가 node 개수에 따라 어떻게 증가하는지 그래프를 그려 알아보았다. 먼저 CSMA scheme의 경우 CSMA/CD scheme에 비해 node 개수가 증가할 때 성능이 아주 급격하게 나빠졌으며, 심지어 throughput이 한 자리수를 넘지 못하는 경우도 발생하였다. 이는 CSMA scheme은 node 개수가 많은 상황에서는 부적합하다는 것을 알 수 있다.

또한 이를 보완한 CSMA/CD는 node 개수가 증가하여도 throughput이 감소하지 않으며, 수렴하는 성향을 보인다. 이는 노드 개수가 아주 많아져도 최소한의 전송은 일어난다는 것을 알려준다. 한편, uniform distribution backoff의 경우 CW가 증가할수록 collision 확률이 감소하는 것은 맞지만, 그것이 throughput의 증가로 직결하는 것은 아니었다. CW가 증가할수록 node들 사이의 간격이 증가하면서 이것이 단위 시간당 보내는 packet의 개수의 감소로 이어질 수 있

기 때문이다. 따라서 CW의 값은 주어진 네트워크에 따라 최적인 값이 다를 수 있으며, 무조건 작거나 큰 것이 좋다고 말할 수 없다. 다시 말해 주어진 네트워크를 분석하고 이에 적합한 scheme을 설정해야 할 것이다.

그런 면에서 현재 IEEE 802.3 MAC 에서 사용하는 scheme인 CSMA/CD + binary exponential backoff는 아주 적합한 approach와 퍼포먼스를 보여주고 있다. 충돌이 적게 일어난다는 것은 node수가 작으므로, cw값이 작은 것이 퍼포먼스에 유리할 것이다. 충돌이 많이 일어난다면 node수가 많으므로, cw값이 큰 것이 퍼포먼스에 유리할 것이다. 이렇게 low load나 high load에 상관없이 효율적인 퍼포먼스를 보여줄 수 있다는 점에서 매우 매력적인 scheme인 것이다. 한편 이번 시뮬레이션에서는 이 매력적인 scheme의 장점이 뚜렷하게 나타나지 않은 점은 아쉬우며, 이를 보기 위해서는 node의 개수가 유동적으로 변하는 상황에서 시뮬레이션을 해야 할 것으로 생각한다.