**CSMA/CD**

경제학과

2012313610

이도현

1. **Implementation**

먼저 3가지 파일로 구성되어 있습니다. 처음에 제시해주신 2개의 node.py, medium.py, 그리고 자동으로 두 파일을 동작시키기 위한 init.py가 있습니다. init.py를 실행해주시면 자동으로 프로그램이 작동합니다. 데이터 수집도 구현하였으니나, 외부 라이브러리 pandas뿐만 아니라 hdf5 관련 패키지 설치가 필요하여 관련 부분은 주석처리 하였습니다.

맥 환경에서 구현하였으며, init.py를 켜면 바로 구동이 가능합니다.

**-Init.py**

init.py는 멀티프로세싱으로 구현하였습니다. 한개의 미디엄과 다수의 노드 프로세스를 만들어 자동으로 동작하게 하였습니다. 또한 매체의 전송 상태, 그리고 충돌 여부를 확인하기 위해 공유 이벤트 변수 e, c를 만들었습니다. e는 BUSY, IDLE 여부를, c는 충돌 여부를 알려줍니다. 이를 각 노드와 미디엄에 전달하여 미디엄은 계속해서 두 변수의 상태를 바꾸고 노드들은 두 변수의 상태를 통해 미디엄의 상태를 확인합니다. 패킷 생성은 마지막 while이 달성합니다. 처음 제시해주신 노드가 stdin을 통해 패킷 생성 명령을 받길래 이 구조를 유지하려고 파이프를 이용했습니다. 각 노드에 파이프를 stdin을 넣고 저 while 문이 랜덤으로 정해진 노드에 패킷 전송 명령을 내립니다. 이때 time.sleep()의 시간을 조절하여 패킷 부하량을 결정할 수 있습니다.

**-node.py**

다음으로 미디엄과 노드 간의 통신인데, 기본 구현은 유사하나 CSMA/CD 구현 과정에서 수정이 있습니다. 먼저 기존과 같이 미디엄이 패킷을 수신 시 모든 노드로 패킷을 보내지 않고, 정해진 목적지에 노드에만 패킷을 전달합니다. 이를 위해 각 노드는 먼저 미디엄과 연결 후 일정시간을 기다리고 미디엄으로부터 전체 노드 주소를 받습니다. 이 주소 목록을 가지고 랜덤하고 보낼 상대를 정하고, 상대를 정한 이후 패킷에 목적지를 씁니다. 이때 패킷은 리스트로 구현됐는데 첫번째 요소는 출발지 주소, 두번째 요소는 도착지 주소, 세번째는 데이터의 종류를 나타내는 요소, 마지막은 데이터를 담습니다. 패킷을 이 리스트를 직렬화하여 전송합니다. 이때 전송 시에 패킷을 받지 않는 문제를 해결하려고 전송 담당부분을 쓰레드로 만들어 한 쓰레드는 수신을, 한 쓰레드는 송신을 맡도록 하였습니다. 물론 패킷을 송신 시에는 패킷 수신부의 sysin의 입력을 막아 새로운 패킷 생성을 제한합니다. 송신 쓰레드는 threadManage에서 관리하는데 이 구현은 쉽습니다. 쓰레드 매니저는 sense\_send\_detect함수를 호출하는데 이 함수는 미디엄의 상태를 확인하고 미디엄이 idle 시에 데이터를 전송, 그리고 전송 이후 TOTAL DELAY만큼 충돌 체크를 한 이후 이상 없을 시에 True를, 이상이 있을 시에 False를 전달합니다. 이때 이 불리언 값을 쓰레드 매니저가 받아 성공 시에 쓰레드를 종료하고 새로운 패킷을 만들도록 합니다. 실패 시에는 랜덤 크기의 타임 슬롯만큼 기다린 이후 다시 sense\_send\_detect를 수행합니다.

**-medium.py**

미디엄 또한 위의 구조에 맞게 변형하였지만 크게 많은 것이 변하지는 않았습니다. 가장 처음 시작할 때 노드들과 연결 후 현재 네트워크에 연결된 노드들의 정보를 각 노드에게 보냅니다. 패킷 포워드의 경우 주소에 맞게 포워드를 해주게 바꾸었습니다. 특징적인 부분은 e, c인데 그 전에 설명드렸다싶이 e는 BUSY, IDLE을 c는 충돌 정보를 알려줍니다. 그리고 충돌 지점의 조건문에서 기존의 코드처럼 thread를 사용하여 스테이트 변화의 유예를 두었습니다. 한가지 중요한 부분은 t1, t2 cancel()부분인데 이는 충돌이 중복될 시에 쓰레드 호출이 여러번 이루어져 원치않는 상태가 되는 것을 방지합니다. 또한 change\_status의 쓰레드 타이머도 중요한데 이때 timer값 PDELAY/3은 미디엄의 상태가 바뀌더라도 노드들이 바로 그 사실을 아는 것을 방지합니다.

1. **Performance Evaluation**

먼저 각 인자의 값을 어찌하는지에 따라 충돌 정도가 많이 달라집니다. 특히나 문제에 요구 사항에서 정해진 인자가 한정되어있는데 이 인자들 이외의 변수의 수치에 따라 충돌 대응 정도가 달라집니다. 예를 들어 문제에 제시되지 않은 미디엄이 전송하는 한다는 사실을 알 때까지의 딜레이 크기가 어떻냐에 따라 충돌 빈도가 확연히 차이나게 되는데 만약 딜레이가 총 지연시간((전송시간 + 전파시간)\*2)의 절반일 경우 각 노드들은 꽤나 오랜 시간이 지난 이후에나 미디엄이 전송 상태임을 알 수 있습니다. 이때 네트워크에 얼마나 많은 패킷 부하가 걸리는지에 따라 충돌 정도가 다르지만, 제 구현은 발생 빈도로 패킷 부하를 실현시켰기에 패킷 생성 딜레이가 총 지연시간보다 많이 작을 경우 대부분 패킷 전송에 실패한 다는 것을 알 수 있습니다.

1. **traffic**

일단 미디엄 전달 지연을 총 전달 지연의 1/4인 0.075로 두고 진행하겠습니다. 그 이유는 문제에서 주어진 백오프 윈도우 최대 개수가 상대적으로 작기에 충돌 발생 시 충돌 관리가 어렵기에 미디엄 상태 전달 지연을 낮춰 충돌 빈도를 줄이기 위해서입니다.

노드 수 : 5

패킷 크기 : 10000비트

대역폭 : 100000bps

전파 지연 : 0.01초

총 지연 시간 : 0.3초

미디엄 상태 전달 지연 : 0.075초

패킷 발생 지연(빈도) : 0.2초

위의 조건에서 15초간 패킷을 주고 받은 결과는 아래와 같습니다.

Node 2

deliver\_time K success

0 0.301428 3 1

1 0.305716 3 1

2 0.301130 3 1

3 0.300274 3 1

4 0.301874 3 1

5 0.300286 3 1

6 0.300674 3 1

7 0.313936 3 1

8 0.300414 3 1

9 0.301293 3 1

10 0.300533 3 1

11 1.500676 8 1

12 0.442952 3 1

13 0.301174 3 1

14 0.406319 3 1

Node 1

deliver\_time K success

0 0.300678 3 1

1 0.311194 3 1

2 0.309379 3 1

3 0.300367 3 1

4 0.310794 3 1

5 0.300445 3 1

6 0.300446 3 1

7 0.300303 3 1

8 0.301084 3 1

9 0.300434 3 1

10 0.323877 3 1

11 0.300981 3 1

12 0.398261 3 1

13 0.300511 3 1

14 0.300357 3 1

15 0.300415 3 1

16 0.300490 3 1

Node 0

deliver\_time K success

0 0.303208 3 1

1 0.300294 3 1

2 0.300503 3 1

3 0.303412 3 1

4 0.300925 3 1

5 0.375696 3 1

6 0.300310 3 1

7 0.300400 3 1

8 0.301628 3 1

9 0.950014 6 1

10 0.427455 3 1

Node 4

deliver\_time K success

0 0.300652 3 1

1 0.378367 3 1

2 0.345275 3 1

3 0.369725 3 1

4 0.322755 3 1

5 0.300376 3 1

6 0.319389 3 1

7 0.300371 3 1

8 0.326189 3 1

9 0.301563 3 1

10 0.300352 3 1

11 0.300494 3 1

12 1.425344 7 1

13 0.395324 3 1

14 0.379966 3 1

15 0.430735 3 1

Node 3

deliver\_time K success

0 0.300542 3 1

1 0.301883 3 1

2 0.301258 3 1

3 0.302719 3 1

4 0.301231 3 1

5 0.300320 3 1

6 0.300346 3 1

7 0.300746 3 1

8 0.348348 3 1

9 0.300389 3 1

10 0.300413 3 1

11 0.300327 3 1

12 0.300306 3 1

13 2.627374 8 1

14 0.361975 3 1

15 0.300485 3 1

여기서 각 테이블은 각 노드의 패킷 전송 데이터를 나타내는데 deliver\_time은 총 전송에 걸린 시간, K는 몇번의 backoff를 하였는지, success는 성공 여부를 1일 때는 성공, 0일 떄는 실패로 나타냅니다.이 자료에서 보면 대부분 경우 충돌없이 K가 3으로 잘 전송되었지만 있다라 충돌이 일어나도 대부분 충돌 제어가 잘 이루어진 것을 확인할 수 있습니다.

이 같은 상태에서 패킷 발생 지연(빈도)을 조정하여 그 여파를 분석해보면 먼저 패킷 발생 지연이 0.2 이상까지는 패킷 손실이 거의 없을 뿐더러, 충돌도 거의 없이 전송이 진행된다는 확인 할 수 있었습니다.

하지만 이 패킷 발생 지연을 낮춘다면, 다시 말해 네트워크에 패킷 부하를 키울 시에는 충돌 제어가 어렵다는 것을 아래의 데이터에서 확인할 수 있습니다.

* **패킷 발생 빈도 증가**

노드 수 : 5

패킷 크기 : 10000비트

대역폭 : 100000bps

전파 지연 : 0.01초

총 지연 시간 : 0.3초

미디엄 상태 전달 지연 : 0.075초

패킷 발생 지연(빈도) : 0.15초

|  |  |  |
| --- | --- | --- |
| Node 0  deliver\_time K success  0 0.300699 3 1  1 0.300404 3 1  2 0.381827 3 1  3 0.300364 3 1  4 0.300391 3 1  5 0.300343 3 1  6 1.198565 7 1  7 2.102620 9 0  8 2.383113 9 0  9 2.908786 9 0  10 2.371822 9 0  11 1.878157 9 0  12 1.951610 9 0  13 2.085238 9 0  14 1.645669 7 1 | Node 1  deliver\_time K success  0 0.300650 3 1  1 0.300356 3 1  2 0.300323 3 1  3 0.351800 3 1  4 0.348322 3 1  5 0.355893 3 1  6 0.300275 3 1  7 0.321396 3 1  8 2.332384 8 1  9 2.543893 8 1  10 1.473750 7 1  11 2.268869 9 0  12 1.675745 9 0  13 3.465417 9 0  14 2.318390 9 0  15 2.619787 8 1 | Node 2  deliver\_time K success  0 0.300615 3 1  1 0.305162 3 1  2 0.505035 4 1  3 0.393588 3 1  4 0.314754 3 1  5 0.348748 3 1  6 0.350272 3 1  7 1.726418 7 1  8 1.061989 9 0  9 1.582312 9 0  10 1.570534 9 0  11 2.721666 9 0  12 1.131986 9 0  13 2.240850 9 0  14 1.775653 9 0  15 2.405957 9 0  16 0.877801 6 1  17 0.302080 3 1 |
| Node 3  deliver\_time K success  0 0.303790 3 1  1 0.300416 3 1  2 0.907546 6 1  3 0.369087 3 1  4 0.353267 3 1  5 0.354163 3 1  6 0.381951 3 1  7 0.300382 3 1  8 1.572870 9 0  9 2.229128 9 0  10 1.134639 6 1  11 2.571427 9 0  12 0.749706 5 1  13 2.185689 9 0  14 1.325147 9 0  15 0.594622 4 1  16 0.927821 9 0  17 0.751727 5 1  18 1.546397 9 0  19 0.755927 5 1  20 2.912230 8 1  24 0.302484 3 1 | Node 4  deliver\_time K success  0 0.300862 3 1  1 0.300449 3 1  2 0.300403 3 1  3 0.300344 3 1  4 0.775721 6 1  5 0.402502 3 1  6 0.412631 3 1  7 0.300553 3 1  8 0.301573 3 1  9 0.301037 3 1  10 1.320636 9 0  11 2.640995 9 0  12 1.080927 6 1  13 1.712317 9 0  14 2.514005 9 0  15 1.962451 9 0  16 1.618924 9 0  17 2.585709 9 0  18 3.099588 8 1  19 0.302435 3 1 |  |

이처럼 패킷을 0.15초 간격으로 발생시키니 0.2초 때에 비해 실패율이 확연히 높아졌을 뿐더러, 성공한 패킷의 전송 시간 또한 증가한 것을 확인할 수 있습니다.

* **패킷 발생 빈도 증가**

노드 수 : 5

패킷 크기 : 10000비트

대역폭 : 100000bps

전파 지연 : 0.01초

총 지연 시간 : 0.3초

미디엄 상태 전달 지연 : 0.075초

패킷 발생 지연(빈도) : 0.1초

|  |  |  |
| --- | --- | --- |
| Node 0  deliver\_time K success  0 2.728914 9 0  1 1.265948 6 1  2 1.533156 9 0  3 1.214728 6 1  4 0.317486 3 1  5 0.330130 3 1  6 2.907394 9 0  7 1.561079 9 0  8 1.741931 9 0  9 1.311543 9 0  10 2.746601 8 1  11 1.953805 9 0  12 2.108602 9 0  13 1.447464 9 0  14 2.291528 9 0  15 3.669319 9 0  16 3.006076 9 0  17 1.377752 7 1  18 1.769361 9 0  19 1.343840 9 0  20 2.397621 9 0  21 1.466643 9 0  22 1.993685 8 1  23 0.709905 5 1  24 1.638442 9 0  25 0.622949 5 1  26 0.311048 3 1 | Node 1  deliver\_time K success  0 0.326501 3 1  1 0.302751 3 1  2 2.331582 8 1  3 0.300673 3 1  4 2.240804 9 0  5 2.615579 9 0  6 2.288623 8 1  7 2.329375 9 0  8 1.526219 8 1  9 2.270171 9 0  10 2.079126 9 0  11 2.187271 9 0  12 2.335041 9 0  13 2.104123 9 0  14 2.422464 9 0  15 1.712569 9 0  16 2.955794 9 0  17 2.021747 9 0  18 3.107419 9 0  19 1.771098 8 1  20 2.094780 9 0  21 2.558001 9 0  22 1.620232 9 0 | Node 2  deliver\_time K success  0 0.339974 3 1  1 0.307692 3 1  2 2.362403 8 1  3 1.785673 9 0  4 1.335377 9 0  5 1.063920 6 1  6 0.323069 3 1  7 0.303803 3 1  8 2.456087 9 0  9 1.783365 7 1  10 1.068678 9 0  11 2.039312 9 0  12 1.764312 9 0  13 2.695332 9 0  14 2.921496 9 0  15 1.975015 9 0  16 0.443822 3 1  17 2.188237 9 0  18 2.496675 9 0  19 1.536174 9 0  20 3.029110 9 0  21 0.976086 5 1  22 1.758157 9 0  23 1.326814 9 0  24 0.670780 4 1  25 1.553177 9 0  26 3.219096 9 0  27 2.987144 8 1  28 0.404235 3 1 |
| Node 3  deliver\_time K success  0 0.302915 3 1  1 2.624459 9 0  2 1.840257 9 0  3 2.859223 9 0  4 1.364003 9 0  5 0.442216 3 1  6 0.310711 3 1  7 2.120031 8 1  8 1.948564 9 0  9 1.451699 9 0  10 0.959992 9 0  11 1.760176 9 0  12 1.331310 9 0  13 2.447741 9 0  14 3.040254 9 0  15 1.862031 9 0  16 2.276639 9 0  17 1.950297 9 0  18 1.646817 9 0  19 2.766108 9 0  20 1.680248 9 0  21 2.390669 9 0  22 1.959951 9 0  23 1.528761 9 0  24 0.513369 3 1  25 3.185701 8 1 | Node 4  deliver\_time K success  0 2.398654 8 1  1 1.823186 9 0  2 2.852910 9 0  3 2.325082 9 0  4 2.330302 9 0  5 1.853802 9 0  6 2.473356 9 0  7 2.711810 8 1  8 1.570884 9 0  9 2.338267 9 0  10 2.987378 9 0  11 0.432963 3 1  12 1.716676 9 0  13 0.427484 3 1  14 1.758280 9 0  15 2.359429 9 0  16 2.759296 9 0  17 3.147884 9 0  18 2.558687 9 0  19 2.690504 9 0  20 1.763647 8 1  21 0.300555 3 1 |  |

패킷을 더 빠른 간격으로 생성, 예를 들어 0.1 간격으로 생성 시에는 대부분 실패하거나, 성공하더라도 패킷 전송 시간이 매우 큰 것을 확인할 수 있습니다.

1. **Node number**

그렇다면 노드의 숫자에 따라서는 어떤 결과가 나타날까요?

먼저 저번 시뮬레이션에서 낮은 패킷 전송률을 보인 0.15 패킷 발생 지연 때 노드 수를 낮추어보면 다음과 같습니다.

* **노드 수 감소**

노드 수 : 3

패킷 크기 : 10000비트

대역폭 : 100000bps

전파 지연 : 0.01초

총 지연 시간 : 0.3초

미디엄 상태 전달 지연 : 0.075초

패킷 발생 지연(빈도) : 0.15초

|  |  |  |
| --- | --- | --- |
| Node 0  deliver\_time K success  0 0.300652 3 1  1 0.300387 3 1  2 0.300348 3 1  3 0.300289 3 1  4 0.300273 3 1  5 0.300474 3 1  6 1.210602 7 1  7 0.414816 3 1  8 0.371447 3 1  9 0.366121 3 1  10 0.301274 3 1  11 0.790498 6 1  12 0.367143 3 1  13 0.730817 5 1  14 0.367881 3 1  15 0.364557 3 1  16 1.150647 7 1  17 0.308233 3 1  18 0.301427 3 1  19 1.028191 6 1  20 0.377622 3 1  21 0.381450 3 1  22 0.352211 3 1  23 0.361070 3 1  24 0.370416 3 1  25 0.364121 3 1  26 0.363939 3 1  27 0.367241 3 1  28 0.381357 3 1  29 0.389445 3 1  30 0.378304 3 1  31 0.387736 3 1 | Node 1  deliver\_time K success  0 0.300634 3 1  1 0.305157 3 1  2 0.302209 3 1  3 1.359040 7 1  4 0.388562 3 1  5 0.376690 3 1  6 0.352900 3 1  7 0.306615 3 1  8 0.835254 6 1  9 0.406170 3 1  10 0.306502 3 1  11 0.917192 6 1  12 0.361643 3 1  13 0.358992 3 1  14 0.308922 3 1  15 2.129923 9 0  16 0.699456 5 1  17 0.347770 3 1  18 0.380504 3 1  19 0.362927 3 1  20 0.374700 3 1  21 0.375585 3 1  22 0.383827 3 1  23 0.362791 3 1  24 0.349944 3 1  25 0.387453 3 1  26 0.361937 3 1  27 0.380168 3 1  28 0.385333 3 1  29 0.382870 3 1  30 0.300394 3 1  31 0.301327 3 1  32 0.300448 3 1  33 0.300238 3 1  34 0.300294 3 1 | Node 2  deliver\_time K success  0 0.300518 3 1  1 0.300386 3 1  2 0.300340 3 1  3 0.929650 6 1  4 0.404885 3 1  5 0.392292 3 1  6 0.365184 3 1  7 1.018071 6 1  8 0.398286 3 1  9 0.304719 3 1  10 0.552536 6 1  11 0.369855 3 1  12 0.358030 3 1  13 0.378897 3 1  14 0.305017 3 1  15 1.516260 8 1  16 0.357195 3 1  17 0.937733 6 1  18 0.369906 3 1  19 0.351672 3 1  20 0.376189 3 1  21 0.362971 3 1  22 0.378915 3 1  23 0.367197 3 1  24 0.385524 3 1  25 0.356264 3 1  26 0.373533 3 1  27 0.371023 3 1  28 0.397146 3 1  29 0.394076 3 1  30 0.354942 3 1  31 0.305160 3 1 |

이 처럼 노드 수가 줄을 시에는 네트워크에 충돌 가능성이 줄어들어, 많은 양의 패킷에도 불구하고 전송이 잘 이루어지는 것을 확인할 수 있습니다.

* **노드 수 증가**

노드 수 : 9

패킷 크기 : 10000비트

대역폭 : 100000bps

전파 지연 : 0.01초

총 지연 시간 : 0.3초

미디엄 상태 전달 지연 : 0.075초

패킷 발생 지연(빈도) : 0.15초

|  |  |  |
| --- | --- | --- |
| Node 0  deliver\_time K success  0 3.082654 9 0  1 2.186824 9 0  2 1.624770 9 0  3 3.239922 9 0  4 2.222940 9 0  5 2.898272 9 0  6 2.133159 9 0  7 3.226391 8 1  8 0.360136 3 1 | Node 1  deliver\_time K success  0 0.311137 3 1  1 1.713480 9 0  2 3.020208 9 0  3 1.988816 9 0  4 2.536212 9 0  5 2.746010 9 0 | Node 2  deliver\_time K success  0 0.380269 3 1  1 2.119645 9 0  2 1.771740 9 0  3 1.232232 9 0  4 2.630910 9 0  5 2.929194 9 0  6 2.117447 9 0  7 1.191097 9 0  8 0.451983 3 1  9 2.375974 9 0  10 1.064209 9 0  11 0.688788 6 1 |
| Node 3  deliver\_time K success  0 2.222253 9 0  1 2.067012 9 0  2 2.429894 9 0  3 1.798143 9 0  4 2.243272 9 0  5 2.058882 9 0  6 3.037613 9 0  7 2.283906 9 0  8 1.421524 9 0  9 2.428405 9 0 | Node 4  deliver\_time K success  0 0.300530 3 1  1 1.450951 9 0  2 1.508386 9 0  3 1.650359 9 0  4 2.911367 9 0  5 1.518454 9 0  6 2.422682 8 1  7 2.111409 9 0  8 3.148548 9 0 | Node 5  deliver\_time K success  0 0.301327 3 1  1 0.300799 3 1  2 1.168314 9 0  3 0.892526 9 0  4 2.070931 9 0  5 1.658921 9 0  6 1.918943 9 0  7 2.491144 9 0  8 2.147458 9 0  9 3.533508 9 0  10 3.264009 9 0  11 1.451319 9 0  12 0.957502 6 1 |
| Node 6  deliver\_time K success  0 0.301433 3 1  1 0.300325 3 1  2 0.300418 3 1  3 0.345423 3 1  4 0.545399 4 1  5 1.889593 9 0  6 2.480131 9 0  7 2.275604 9 0  8 2.487638 9 0  9 2.797191 9 0  10 1.739143 9 0 | Node 7  deliver\_time K success  0 0.300879 3 1  1 0.415507 3 1  2 0.300400 3 1  3 1.704801 9 0  4 2.449641 9 0  5 2.864163 9 0  6 2.748736 9 0  7 2.799496 9 0  8 2.451409 9 0  9 1.780114 9 0  10 2.582258 8 1  11 0.869608 7 1 | Node 8  deliver\_time K success  0 0.301664 3 1  1 0.300396 3 1  2 1.117752 9 0  3 0.916484 9 0  4 2.664451 9 0  5 1.979629 9 0  6 2.010052 9 0  7 1.599130 9 0  8 2.666273 9 0  9 1.947278 9 0  10 2.773562 9 0  11 1.566154 9 0  12 2.307270 9 0  13 0.668414 5 1 |

이와 반대로 노드 수를 9개로 증가시에는 너무나 잦은 충돌이 일어나서 초반 부를 제외한 대부분의 경우 전송에 실패한다는 사실을 알 수 있습니다.

1. **Packet size**

마지막으로 패킷의 크기에 따른 성능 차이를 알아보겠습니다. 지금의 구현상 패킷의 크기가 증가하면, 전송시간이 증가하고, 총 지연이 증가하게 됩니다.

먼저 위의 패킷 발생 지연 0.15초, 노드 수 5개의 경우 패킷 크기를 전의 5배인 5000으로 키울 시 결과는 다음과 같습니다.

* **패킷 크기 증가**

노드 수 : 5

패킷 크기 : 50000비트

대역폭 : 100000bps

전파 지연 : 0.01초

총 지연 시간 : 0.3초

미디엄 상태 전달 지연 : 0.075초

|  |  |  |
| --- | --- | --- |
| 패킷 발생 지연(빈도) : 0.15초 Node 0  deliver\_time K success  0 1.102685 3 1  1 1.101549 3 1  2 1.103914 3 1  3 3.800610 6 1  4 1.114317 3 1  5 1.100396 3 1  6 1.114722 3 1  7 1.102847 3 1  8 1.111064 3 1  9 1.121231 3 1  10 1.103512 3 1  11 1.102697 3 1  12 1.102720 3 1  13 1.107084 3 1  14 1.106880 3 1  15 1.116802 3 1  16 1.105802 3 1  17 1.102145 3 1  18 1.100844 3 1  19 1.100451 3 1  20 1.114896 3 1 | Node 1  deliver\_time K success  0 1.102583 3 1  1 1.100470 3 1  2 1.102028 3 1  3 1.110126 3 1  4 4.197761 7 1  5 1.110818 3 1  6 1.107023 3 1  7 1.109342 3 1  8 1.110154 3 1  9 1.114919 3 1  10 1.112656 3 1  11 1.106394 3 1  12 1.130678 3 1  13 1.118918 3 1  14 1.102125 3 1  15 1.105620 3 1  16 1.106729 3 1  17 1.105703 3 1  18 1.101289 3 1  19 1.105908 3 1  20 1.100843 3 1  21 1.100598 3 1  22 1.100508 3 1  23 1.100365 3 1  24 1.100419 3 1  25 1.100295 3 1 | Node 2  deliver\_time K success  0 1.102429 3 1  1 1.101817 3 1  2 1.176562 3 1  3 3.228824 6 1  4 1.101250 3 1  5 1.151810 3 1  6 1.114093 3 1  7 1.104857 3 1  8 1.101489 3 1  9 1.116321 3 1  10 1.160074 3 1  11 1.100322 3 1 |
| Node 3  deliver\_time K success  0 1.101535 3 1  1 1.101498 3 1  2 1.100671 3 1  3 1.101004 3 1  4 3.951207 7 1  5 1.103635 3 1  6 1.118625 3 1  7 1.119475 3 1  8 1.104093 3 1  9 1.107918 3 1  10 1.104300 3 1  11 1.113089 3 1  12 1.145445 3 1  13 1.106297 3 1  14 1.102903 3 1  15 1.104284 3 1  16 1.117630 3 1  17 1.106784 3 1  18 1.102347 3 1  19 1.100612 3 1  20 1.100325 3 1  21 1.100637 3 1  22 1.100389 3 1 | Node 4  deliver\_time K success  0 1.123560 3 1  1 5.457097 8 1  2 1.113765 3 1  3 1.103244 3 1  4 1.104091 3 1  5 1.112842 3 1  6 1.103130 3 1  7 1.110994 3 1  8 1.108131 3 1  9 1.105107 3 1  10 1.100510 3 1  11 1.103799 3 1  12 1.110079 3 1  13 1.101380 3 1  14 1.100452 3 1  15 1.112634 3 1  16 1.104182 3 1 |  |

이와 같이 패킷의 크기가 증가했다고 하여 패킷 전달율이 떨어지지는 않습니다. 반면 전송시간은 확연히 증가한다는 것을 확인할 수 있습니다. 결국 전송 시간이 증가하면 타임 슬롯 크기와 같은 수치들에 비례하게 증가하기에 패킷 전송률과는 관계가 없는 것으로 생각됩니다.

* **패킷 크기 감소**

노드 수 : 5

패킷 크기 : 5000비트

대역폭 : 100000bps

전파 지연 : 0.01초

총 지연 시간 : 0.3초

미디엄 상태 전달 지연 : 0.075초

패킷 발생 지연(빈도) : 0.15초

|  |  |  |
| --- | --- | --- |
| Node 0  deliver\_time K success  0 0.100802 3 1  1 0.100812 3 1  2 0.100912 3 1  3 0.100349 3 1  4 0.100408 3 1  5 0.100321 3 1  6 0.100409 3 1  7 0.100300 3 1  8 0.100544 3 1  9 0.100362 3 1  10 0.100611 3 1  11 0.100318 3 1  12 0.100589 3 1  13 0.100552 3 1  14 0.100309 3 1  15 0.100683 3 1  16 0.100398 3 1  17 0.100604 3 1  18 0.100395 3 1  19 0.100501 3 1  20 0.100528 3 1 | Node 1  deliver\_time K success  0 0.100653 3 1  1 0.100362 3 1  2 0.100439 3 1  3 0.100351 3 1  4 0.100298 3 1  5 0.100524 3 1  6 0.100637 3 1  7 0.100627 3 1  8 0.100570 3 1  9 0.100465 3 1  10 0.100521 3 1  11 0.101372 3 1  12 0.100566 3 1  13 0.100378 3 1  14 0.100564 3 1  15 0.100429 3 1 | Node 2  deliver\_time K success  0 0.101441 3 1  1 0.100746 3 1  2 0.100358 3 1  3 0.100492 3 1  4 0.100534 3 1  5 0.100539 3 1  6 0.100480 3 1  7 0.100308 3 1  8 0.100437 3 1  9 0.100649 3 1  10 0.100512 3 1  11 0.100523 3 1  12 0.100382 3 1  13 0.100658 3 1  14 0.100594 3 1  15 0.100361 3 1  16 0.100331 3 1  17 0.100795 3 1  18 0.100664 3 1  19 0.100827 3 1  20 0.100606 3 1  21 0.100559 3 1  22 0.100331 3 1  23 0.100538 3 1  24 0.101726 3 1 |
| Node 3  deliver\_time K success  0 0.100845 3 1  1 0.100306 3 1  2 0.100396 3 1  3 0.100457 3 1  4 0.100545 3 1  5 0.100358 3 1  6 0.100519 3 1  7 0.100304 3 1  8 0.100620 3 1  9 0.100430 3 1  10 0.100449 3 1  11 0.100727 3 1  12 0.100328 3 1  13 0.100444 3 1  14 0.100500 3 1  15 0.100605 3 1  16 0.100642 3 1  17 0.100451 3 1  18 0.100305 3 1 | Node 4  deliver\_time K success  0 0.100592 3 1  1 0.100609 3 1  2 0.100358 3 1  3 0.100569 3 1  4 0.100581 3 1  5 0.101632 3 1  6 0.100297 3 1  7 0.100614 3 1  8 0.100633 3 1  9 0.100377 3 1  10 0.100324 3 1  11 0.100590 3 1  12 0.100474 3 1  13 0.100559 3 1  14 0.100432 3 1  15 0.100501 3 1  16 0.100735 3 1  17 0.100515 3 1 |  |

이 경우는 패킷 크기를 그 전의 반으로 줄인 것인데 여기서 또한 전송시간만 바뀔 뿐, 패킷 전달률에는 큰 영향이 없는 것이 확인된다.

1. **Conclusion**

이번 CSMA/CB 구현 실험을 통해 많은 것을 배우게 되었는데 먼저 CSMA/CB이 실제 돌아가는 로직을 알 수 있었다. 더 중요한 부분은 단지 작동 로직에서 더 나아가 CSMA/CD의 특징들을 익힐 수 있었다.먼저 CSMA/CD 네트워크 사용자가 증가하거나, 부하가 증가할 시에 네트워크가 현재의 충돌 제어 방식으로는 한계가 있음을 알 수 있었다. 반면 패킷의 크기는 현재의 구현에서는 패킷 전달율에 큰 영향을 미치지 않는 다는 사실을 알 수 있었다.