과목명	무선이동통신	담당교수	조성래 교수님		
학과	소프트웨어학과	학번	20186889	이름	권용한
제 목: 무선 애드혹 네트워크 라우팅 프로토콜의 성능 비교 분석					

1. 개 요

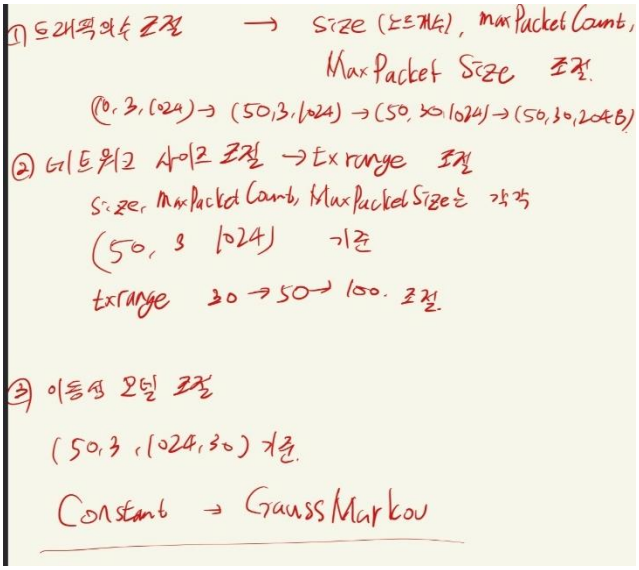
- 무선 애드혹 네트워크 라우팅 프로토콜(AODV, DSDV)의 성능을 비교 분석합니다.
- 트래픽, 네트워크 사이즈, 이동성 여부 등의 환경을 변화시켜가며 프로토콜의 성능을 측정 및 분석합니다.

2. 관련 이론

- AODV, DSR, DSDV의 동작원리와 각 프로토콜의 장단점 비교
- DSDV – 벨만_포드 알고리즘에 기반한 라우팅 프로토콜입니다.
  Table Driven protocol로, 선제적으로 라우팅 테이블을 가지게 됩니다.
  시퀀스 번호를 사용해 링크가 끊겼을 때, 제대로 된 링크 정보를 loop에 빠지지 않고 업데이트 할 수 있습니다.
- DSR – On Demand 방식의 라우팅 방식으로, 패킷 헤더에 모든 경로를 포함해, 헤더가 커질 수 있습니다.
  소스 라우팅을 통해 소스에서 RREP를 보냅니다.
- AODV – DSR과 마찬가지로, On Demand 방식의 라우팅 프로토콜입니다.
  DSR과 유사한 동작방식을 가지지만, 중간 노드에서 처리방식이 다릅니다. 모든 경로를 기록하는 것이 아닌, 목적지 까지 가는 경로에서 넥스트홉이 어느 노드인지에 대한 정보만을 헤더에 포함시킵니다.
  AODV는 소스라우팅이 필요하지 않고, RREQ와 RREP를 통해 목적지와 넥스트홉에 대한 정보를 가진 Table을 생성합니다.

3. 본 문

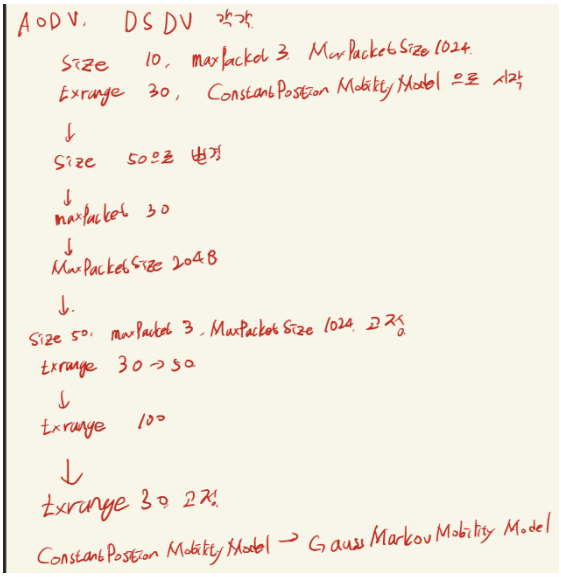
- 트래픽, 네트워크 사이즈, 이동성 여부 등을 변화시키기 위해, 해당 환경에 영향을 미칠 것으로 판단되는 변수를 확인, 이를 조정해 가면서 결과값을 확인 및 분석하였습니다.



우선 트래픽의 경우, 노드의 수, 패킷의 수와 사이즈에 영향을 받는다고 생각해, default값 10, 3, 1024 에서 값을 증가 시키며 결과값을 확인하였습니다.

그리고 다음으로 네트워크의 사이즈는 전달 범위가 중요하다 생각해 txrange를 조절해 확인하였으며, 이때 트래픽의 수는 고정하기 위해 50, 3, 1024를 기준으로 작성하였습니다.

마지막으로 이동성 모델의 경우 이동하지 않는 ConstantPositionMobilityModel과, 노드의 분포가 가우스 분포를 따르는 GaussMarkovMobilityModel을 채택하여 실험을 진행하였습니다. 이때 다른 변수는 50, 3, 1024, 30로 설정해 실험을 진행하였습니다.



- 출력되는 값을 엑셀에 저장합니다.

-

```
ofstream writeFile;
writeFile.open("result.csv",std::ios_base::out | std::ios_base::app);
writeFile<<txPacketsum<<','<<rxPacketsum<<','<<lostPacketsum<<','<<((rxBytessum * 8.0) / simTime)/1024<<','<<(((txPacketsum - lostPacketsum) * 100) /txPacketsum
writeFile.close();
```

- 기존 주어진 AODV 소스코드를 활용해 주요 부분은 동일하게 작성하였습니다.
- 인터넷스택에 원하는 프로토콜을 넣어 실험을 진행하였습니다.

```
AodvHelper aodv;
DsrHelper dsr;
DsdvHelper dsdv;
DsrMainHelper dsrMain;
// you can configure AODV attributes here using aodv.Set(name, value)
InternetStackHelper stack;
//  stack.SetRoutingHelper (aodv); // has effect on the next Install ()
stack.SetRoutingHelper (dsdv); // has effect on the next Install ()
```

AODV와 DSDV는 실험을 진행하였지만, DSR의 경우 FlowMonitor로 측정이 어려웠습니다.

우선 다른 방법을 찾고자 하였지만, 방법을 제대로 파악하지 못하여 DSR 방식은 제대로 측정하지 못하고 실험을 종료하였습니다…

- 노드의 수를 해당 변수를 통해 조절하였으며,

```
// size (10),   //노드 개수
size (50),    //노드 개수
```

- Packet의 수와 size 또한 조절하였습니다.

```
uint32_t MaxPacketSize = 1024;
//  uint32_t MaxPacketSize = 2048;

Time interPacketInterval = Seconds (0.01);
uint32_t maxPacketCount = 3; //트래픽
//  uint32_t maxPacketCount = 30; //트래픽
double interval_start = 2.0, interval_end = interval_start + interval;
```

- Txrange변수를 통해 네트워크 사이즈를 조정하였고,

```
double txrange = 30;  //네트워크 사이즈?
// double txrange = 50;  //네트워크 사이즈?
// double txrange = 100;  //네트워크 사이즈?
```

- 마지막으로 MobilityModel을 조정하였습니다.

```
MobilityHelper mobilityS;
mobilityS.SetPositionAllocator(positionAllocS);
//  mobilityS.SetMobilityModel("ns3::ConstantPositionMobilityModel"); //whatever it is
mobilityS.SetMobilityModel("ns3::GaussMarkovMobilityModel"); //whatever it is

mobilityS.Install(nodes);
```

- 트래픽의 경우, 사용자의 수(노드의 수)와 보낸 패킷의 크기에 비례한다고 생각해 노드의 수(size)와 패킷의 크기(maxPacket, MaxPacketSize)를 조정해 실험을 진행했습니다.
- 네트워크 사이즈의 경우, 네트워크가 전달될 수 있는 범위의 크기로 생각, txrange 를 조정해 변경할 수 있다고 이해하였으며, 해당 값을 30, 50, 10 으로 조절해 실험을 진행했습니다.
- 이동성은 대표적인 경우를 가정해 이동하지 않는 ConstantPosition 과 랜덤하게 이동하는 경우 GaussMarkov 로 설정해 실험을 진행했습니다.

4. 실험 결과

- 실험 시나리오에 대한 설명
  - Ns-3.37 버전에서 실험을 진행하였습니다.
  - 트래픽, 네트워크 사이즈, 이동성에 따른 성능을 측정하였으며,
  - Mobility Model, Traffic Pattern, Packet Size, Propagation Loss Model 등등 실험 결과에 영향을 주는 파라미터에 대해서 기술
- 노드의 수(size)

  AODV

Size	Total Tx Packets	Throughput	Packets Delivery Ratio	Average E2E Delay
10	19	0.8675	0.52	0.109802
50	604	5.42781	0.74	0.135467

  DSDV

Size	Total Tx Packets	Throughput	Packets Delivery Ratio	Average E2E Delay
10	12	0.493125	0.25	0.006741
50	72	3.45187	0.29	0.505334

- maxPacket

  AODV

maxPacket	Total Tx Packets	Throughput	Packets Delivery Ratio	Average E2E Delay
3	604	5.42781	0.74	0.135467
30	1117	3.53375	0.29	0.178663

  DSDV

maxPacket	Total Tx Packets	Throughput	Packets Delivery Ratio	Average E2E Delay
3	72	3.45187	0.29	0.505334
30	720	19.8894	0.16	0.142441

- MaxPacketSize

  AODV

MaxPacketSize	Total Tx Packets	Throughput	Packets Delivery Ratio	Average E2E Delay
1024	1117	3.53375	0.29	0.178663
2048	1406	18.3203	0.41	0.27779

  DSDV

MaxPacketSize	Total Tx Packets	Throughput	Packets Delivery Ratio	Average E2E Delay
1024	720	19.8894	0.16	0.142441
2048	720	32.1131	0.13	0.286083

- Txrange

AODV

Txrange	Total Tx Packets	Throughput	Packets Delivery Ratio	Average E2E Delay
30	604	5.42781	0.74	0.135467
50	4970	31.0978	0.76	0.112895
100	2188	17.5059	0.75	0.224309

DSDV

Txrange	Total Tx Packets	Throughput	Packets Delivery Ratio	Average E2E Delay
30	72	3.45187	0.29	0.505334
50	72	10.0269	0.84	1.35886
100	72	9.36937	0.79	1.73731

- MobilityModel

AODV

MobilityModel	Total Tx Packets	Throughput	Packets Delivery Ratio	Average E2E Delay
ConstantPosition	604	5.42781	0.74	0.135467
GaussMarkov	726	6.69813	0.76	0.251435

DSDV

MobilityModel	Total Tx Packets	Throughput	Packets Delivery Ratio	Average E2E Delay
ConstantPosition	72	3.45187	0.29	0.505334
GaussMarkov	72	4.93125	0.41	2.28813

5. 고찰

- AODV – OnDemand 방식이기 때문에 destination에 해당하는 table이 존재하지 않으면, 해당 destination으로 가는 넥스트홉을 찾기 위해 RREP, RREQ를 보내고 받게 됩니다.
- 트래픽 – 노드의 수와 패킷의 수, 길이가 증가한다면 트래픽이 증가할 것으로 생각하고 실험을 진행했습니다.

  노드의 수가 증가하면 평균 딜레이와 처리량 및 패킷 전달 비율은 증가함을 확인하였습니다.

  패킷 사이즈와 패킷의 수의 경우엔 처리량과 평균 딜레이는 증가하였지만, 전달비율의 경우 패킷의 수가 증가했을 때 감소했지만, 패킷의 크기를 증가했을 때는 증가함을 확인했습니다.
- 네트워크 사이즈 – txrange가 커짐에 따라 네트워크 사이즈가 커진다고 생각했습니다.
- txrange가 커질수록 딜레이는 커졌지만, 처리량 및 패킷 전달 비율의 경우엔 증가 했다가 다시 감소하였습니다.
- 이동성 – 이동성이 부여됐을 경우, 더 많은 패킷을 보냈으며, 처리량이 증가했음을 확인할 수 있습니다. 이동성이 부여된 경우 처리량 및 딜레이가 더 증가했으며 전달 비율은 증가했지만, 이는 둘 다 높은 성능을 보여준다 판단하였습니다.

- DSDV – Table Driven protocol로, 선제적으로 Table을 생성하게 됩니다.
- 트래픽 – 노드의 수와 패킷의 수, 길이가 증가한다면 트래픽이 증가할 것으로 생각하고 실험을 진행했습니다.

  AODV 방식과 마찬가지로, 노드의 수가 증가하면 평균 딜레이, 처리량 및 패킷 전달 비율이 증가함을 확인할 수 있었습니다.

  패킷의 크기와 수가 증가함에 따라 평균처리량은 증가, 종단간 딜레이 또한 증가하였으며, 패킷전달비율은 감소하였습니다.
- 네트워크 사이즈 – txrange가 커짐에 따라 네트워크 사이즈가 커진다고 생각했습니다.

  AODV와 마찬가지로 txrange가 커질수록 딜레이는 커졌지만, 처리량 및 패킷 전달 비율의 경우엔 증가 했다가 다시 감소하였습니다.
- 이동성 – 보낸 패킷의 수가 동일하지만, 이동성에 따라 다른 결과를 확인할 수 있었습니다.

  이동성이 부여된다면 처리량과 딜레이가 증가했지만, 전달비율 또한 증가했음을 확인하였습니다.

- 처리량은 받은 패킷의 크기에 비례하며, 때문에 전달비율이 증가하면 함께 증가하는 양상을 띄며, 패킷사이즈와 패킷의 수에 비례하는 결과를 보여줍니다.

  노드가 증가하였을 경우엔 AODV와 DSDV 모두 딜레이가 조금 증가하지만, 성공 비율 및 처리량이 증가하였는데, 노드가 너무 적으면 연결이 끊긴 지점이 있다고 생각되며, 때문에 해당 지점으로 패킷을 보내지 못해 성능이 떨어지는 것이 아닐까 하고 생각하였습니다.

  다만 패킷의 수와 사이즈가 증가할 때는 성능이 더 떨어졌는데, 패킷이 처리되기 전에 다른 패킷이 전달되는 등의 상황이 발생해 병목현상이 발생해서 성능이 떨어지는 것으로 생각됩니다.
- 딜레이는 노드의 수가 커지면서 AODV가 DSDV보다 더 우수한 성능을 보여주는 것을 확인하였습

니다. 다만 노드가 10일땐 DSDV가 더 우수한 성능을 보여주었습니다.

- 네트워크의 범위가 커지면 DSDV가 AODV보다 더 우수한 성능을 보여준다고 생각 되었는데, 다만 딜레이가 DSDV에서 더 컸습니다.

  다만 AODV는 On Demand 방식이기 때문에, 의미있는 패킷보다 RREQ, RREP가 더 많은 비율을 차지할 수 있다고 생각됩니다.

  전체적으로 수치적으로는 AODV가 DSDV보다 딜레이가 적고 성공 비율이 더 높기 때문에 AODV가 더 우수한 성능을 보여준다고 생각됩니다.

- 의문점

  AODV에서 패킷의 크기가 증가한 경우엔 전달 비율이 증가하였는데, 트래픽이 증가해 병목현상이 일어나 성능이 감소한다고 생각했는데, 어째서 이때 전달 비율이 증가하게 되었는가.

  AODB에서 Txrange가 50에서 100으로 더 증가했음에도 보내는 패킷의 수가 감소하였는가.

6. 결론

- 실험을 성공적으로 진행하기 위해서 네트워크의 사이즈가 일정하다면 일정 수준의 노드가 필요하다고 생각됩니다. 노드의 수가 너무 적다면 네트워크의 사이즈가 작아서 제대로 연결이 되지 못해 실험 결과가 좋지 못하게 나온다고 생각됩니다.

  트래픽의 수가 증가한다면 네트워크의 범위가 충분하다는 가정하에, 노드가 처리할 수 있는 수준까지는 처리량 및 전달 비율이 증가할 것으로 판단되지만, 노드가 처리할 수 있는 범위를 넘게 된다면 병목현상이 발생해 전달되지 못한 패킷은 drop out되어 성능이 더 떨어지게 되는 것으로 관찰되었습니다.

  앞서 설명했듯, 성능을 높이기 위해선 일정 수준 이상의 네트워크 사이즈가 구성되어야 하며, 일정 사이즈를 넘는다면 좋은 성능을 보여주는 것으로 확인됩니다.

  위에서 언급한 의문 사항 중, AODV에서 네트워크 사이즈가 증가했음에도 처리량이 감소하는 것은 아마 끊긴 지점이 없어져서 보내는 패킷의 량이 감소하는 것이 아닐까 하고 판단됩니다.