과목명	무선이동통신	담당교수	조성래 교수님		
학과	소프트웨어학부	학번	20204898	이름	박소은
제 목: 이동 속도에 따른 DSDV, DSR, ADOV 프로토콜 성능 분석					

개요

프로젝트의 목표

이동 속도에 따라 DSDV, DSR, ADOV 프로토콜 성능이 어떻게 달라지는지 분석한다.

진행한 프로젝트에 대한 요약

이동 속도가 사람이 걷는 속도(평균 1.4m/s)일 때와, 이동 수단을 타고 이동하는 속도(28m/s)에 따라 DSDV, DSR, ADOV 프로토콜을 사용하여 성능을 분석하였다.

관련 이론

AODV, DSR, DSDV의 동작 원리와 각 프로토콜의 장단점 비교

- AODV: Ad hoc On-demand Distance Vector

필요한 경우에만 경로를 찾는다. 노드 간의 거리를 계산하여 목적지까지 최단 경로를 찾고, 거리 정보 또는 노드 상태가 변경되면 즉시 경로를 수정한다. 경로 변경에 빠르게 대응하기 때문에 경로 변경이 많은 모바일 노드에서 잘 작동된다. 이에 따른 단점으로, 노드가 많이 이동하는 경우 라우팅 테이블이 계속 업데이트되어야 하므로 오버헤드가 커진다.

- DSR: Dynamic Source Routing

Request packet을 전송할 때, 경로가 필요해지면 경로를 탐색한다. 데이터를 보내는 노드가 경로를 찾아서 패킷에 경로 정보를 입력한다. 경로를 알고 있는 경우에는 데이터 패킷에 경로를 포함시키고, 모르는 경우에는 경로 탐색 프로세스를 통해 주변 노드들로 경로를 찾는다. 경로 변경이 자주 발생하는 상황에서 사용된다. 초기에 경로를 탐색하기 위한 시간이 발생하며, 경로를 저장하기 위한 패킷의 저장공간이 필요하다. 특히 네트워크의 크기가 큰 경우, 저장공간이 더 많이 필요하다.

- DSDV: Destination-Sequenced Distance-Vector

모든 노드가 라우팅 테이블을 유지하여, 목적지까지의 거리와 순서를 알고 있다. 노드는 서로 라우팅 테이블에 관한 정보들을 교환하며 최신 정보로 업데이트한다. 안정적인 경로가 필요할 때 효율적으로 작동하는 프로토콜이다. 그러나 변화가 많은 네트워크에서는 오버헤드가 커지고, 라우팅 정보를 저장할 공간이 많이 필요하다는 단점이 있다.

본문

NS-3 시뮬레이터 구현 절차

1. P2pNodes, csmaNodes 등 노드 생성
2. YansWifi 설정
3. SSID 설정
4. Mobility 설정: mobilityType이 0인 경우 28m/s, 1인 경우 1.4m/s로 움직이도록 설정
5. InternetStack을 프로토콜에 맞게 설정
6. IP 설정 후 시뮬레이션 시작
7. 시뮬레이션 결과에 대해 FlowMonitor를 활용하여 성능 측정

전체 소스 코드가 아닌 핵심 소스 코드 위주로 설명

- Mobility setting

```
if (mobilityType == 0) {
    // Node moves fast
    NS_LOG_INFO("Node moves fast: 24m/s (Car)");
    mobility.SetMobilityModel("ns3::RandomWalk2dMobilityModel",
        "Bounds", RectangleValue (Rectangle (-50, 50, -50, 50)),
        "Speed", StringValue("ns3::ConstantRandomVariable[Constant=28]"));
} else if (mobilityType == 1) {
    // Node moves slow
    NS_LOG_INFO("Node moves slow: 1.4m/s (Human)");
    mobility.SetMobilityModel("ns3::RandomWalk2dMobilityModel",
        "Bounds", RectangleValue (Rectangle (-50, 50, -50, 50)),
        "Speed", StringValue("ns3::ConstantRandomVariable[Constant=1.4]"));
}
mobility.Install(wifiStaNodes);

// AP는 움직이면 안 되므로 Constant poisition을 가진다
mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
mobility.Install (wifiApNode);
```

mobilityType을 사용자에게 입력 받아 0인 경우 자동차 정도의 속도인 28m/s, 1을 입력받은 경우 사람이 걷는 속도인 1.4m/s로 설정한다.

AP 노드는 움직이면 안 되므로 ConstantPositionMobility 모델을 사용하고, 움직이는 노드는 RandomWalk2dMobility 모델을 사용한다.

- Protocol 설정

```
// Internet stack, protocol
InternetStackHelper stack;

if (protocol == 0) {
    AodvHelper aodv;
    stack.SetRoutingHelper(aodv);
    NS_LOG_INFO("AODV protocol");
} else if (protocol == 1) {
    DsrMainHelper dsrMain;
    DsrHelper dsr;
    dsrMain.Install(dsr, p2pNodes);
    dsrMain.Install(dsr, csmaNodes);
    if (apDevices.GetN() > 0) {
        dsrMain.Install(dsr, wifiApNode);
        dsrMain.Install(dsr, wifiStaNodes);
    }
    NS_LOG_INFO("DSR protocol");
} else if (protocol == 2) {
    DsdvHelper dsdv;
    stack.SetRoutingHelper(dsdv);
    NS_LOG_INFO("DSDV protocol");
```

AodvHelper, DsMainHelper, DsdvHelper를 사용하여 각 프로토콜에 맞게 설정한다.

- FlowMonitor를 활용한 성능 측정

```
for(std::map<FlowId, FlowMonitor::FlowStats>::const_iterator i = stats.begin(); i != stats.end(); i++) {
    Ipv4FlowClassifier::FiveTuple t = classifier->FindFlow(i->first);
    NS_LOG_INFO("Flow " << i->first << " (" << t.sourceAddress << "->" << t.destinationAddress << ")\n");
    NS_LOG_INFO("  Lost Packets: " << i->second.lostPackets << "\n");
    NS_LOG_INFO("  Delay Sum: " << i->second.delaySum.GetSeconds() << " s\n");
    NS_LOG_INFO("  Default Delay: " << i->second.delaySum.GetSeconds()/i->second.rxPackets << " s\n");
    NS_LOG_INFO("  throughput : " << i->second.rxBytes * 8.0 / (i->second.timeLastRxPacket.GetSeconds()
            - i->second.timeFirstTxPacket.GetSeconds())/1000000 << " Mbps" << std::endl);

    // std::cout << "  Tx Bytes: " << i->second.txBytes << "\n";
    // std::cout << "  Rx Bytes: " << i->second.rxBytes << "\n";

    if (endTime < i->second.timeLastRxPacket.GetSeconds()) {
        endTime = i->second.timeLastRxPacket.GetSeconds();
    }
}
```

각 Flow마다 Lost packet 수, Delay Sum, Throughput을 구하여 로그로 출력한다.

하나의 Flow는 한 노드에서 다른 노드로 패킷이 전달되는 과정이다.

실험 결과

실험 시나리오에 대한 설명

- Mobility Model

  이동하는 노드의 경우 RandomWalk2dMobilityModel을 사용하였다.

  Bounds parameter값은 공통적으로 RectangleValue (Rectangle (-500, 500, -500, 500))로 설정하였고, 사용자가 입력한 mobilityType 변수의 값에 따라 Speed parameter값을 구분하였다.

  mobilityType이 0인 경우 자동차 평균 이동속도 정도인 28m/s로 설정하였고, 1인 경우 사람이 걸어다니는 평균 속도인 1.4m/s로 설정하였다.

  AP 노드는 ConstantPositionMobilityModel 모델을 사용하여 움직이지 않게 설정하였다.

- Packet size: 공통적으로 100개의 패킷을 보내도록 설정하였다.

실험 결과 설명

- 코드 실행 결과

```
soeun@soeun-VirtualBox:~/workspace/ns-3-allinone/ns-3.33$ NS_LOG="TermProject"
./waf --run "scratch/term_project"
Waf: Entering directory `/home/soeun/workspace/ns-3-allinone/ns-3.33/build'
[2727/2787] Compiling scratch/term_project.cc
[2748/2787] Linking build/scratch/term_project
Waf: Leaving directory `/home/soeun/workspace/ns-3-allinone/ns-3.33/build'
Build commands will be stored in build/compile_commands.json
'build' finished successfully (16.271s)
+0.000000000s -1 TermProject:main(): [INFO ] Node moves fast: 24m/s (Car)
+0.000000000s -1 TermProject:main(): [INFO ] AODV protocol
+0.000000000s -1 TermProject:main(): [INFO ] Starting Simulation
/NodeList/7/$ns3::MobilityModel/CourseChange x = 10, y = 0
/NodeList/7/$ns3::MobilityModel/CourseChange x = 10.7695, y = -0.638654
/NodeList/7/$ns3::MobilityModel/CourseChange x = 11.6392, y = -0.145047
/NodeList/7/$ns3::MobilityModel/CourseChange x = 12.632, y = -0.0254987
/NodeList/7/$ns3::MobilityModel/CourseChange x = 11.6363, y = 0.0671053
/NodeList/7/$ns3::MobilityModel/CourseChange x = 10.758, y = 0.545161
/NodeList/7/$ns3::MobilityModel/CourseChange x = 9.9445, y = -0.0364386
/NodeList/7/$ns3::MobilityModel/CourseChange x = 10.3303, y = 0.886132
/NodeList/7/$ns3::MobilityModel/CourseChange x = 9.44532, y = 0.420558
/NodeList/7/$ns3::MobilityModel/CourseChange x = 10.2426, y = -0.183
/NodeList/7/$ns3::MobilityModel/CourseChange x = 9.25125, y = -0.313963
/NodeList/7/$ns3::MobilityModel/CourseChange x = 8.25855, y = -0.434593
/NodeList/7/$ns3::MobilityModel/CourseChange x = 9.05376, y = -1.04094
/NodeList/7/$ns3::MobilityModel/CourseChange x = 8.35122, y = -0.329293
/NodeList/7/$ns3::MobilityModel/CourseChange x = 8.1272, y = -1.30388
/NodeList/7/$ns3::MobilityModel/CourseChange x = 9.05006, y = -0.918745
/NodeList/7/$ns3::MobilityModel/CourseChange x = 8.98678, y = -1.91674
/NodeList/7/$ns3::MobilityModel/CourseChange x = 8.03238, y = -1.61823
/NodeList/7/$ns3::MobilityModel/CourseChange x = 8.56852, y = -0.774105
/NodeList/7/$ns3::MobilityModel/CourseChange x = 7.59529, y = -1.00395
/NodeList/7/$ns3::MobilityModel/CourseChange x = 8.56751, y = -1.23803
```

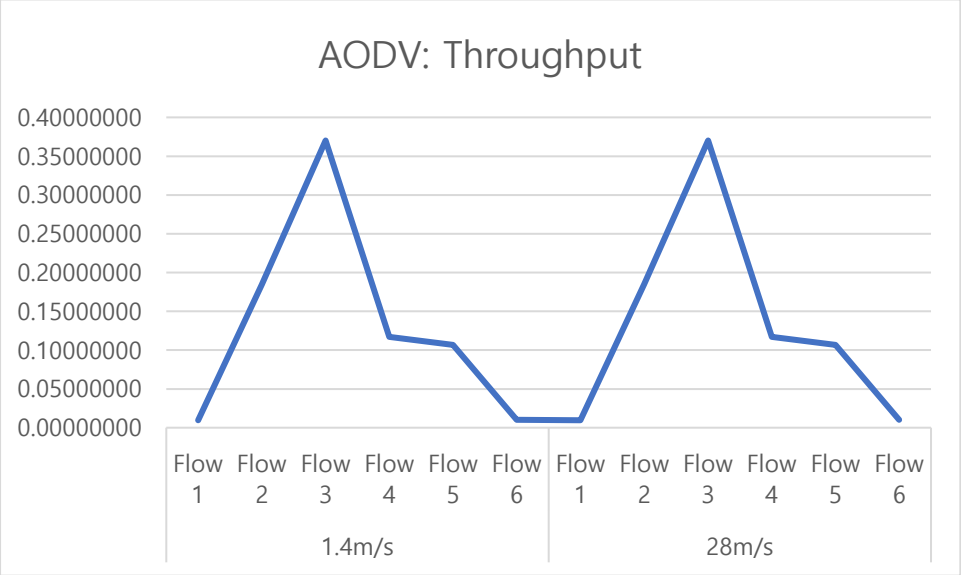
```
+20.000000000s -1 TermProject:main(): [INFO ]   Default Delay: 0.0010372 s
+20.000000000s -1 TermProject:main(): [INFO ]   throughput : 0.370228 Mbps
+20.000000000s -1 TermProject:main(): [INFO ] Flow 4 (10.1.1.1->10.1.1.2)
+20.000000000s -1 TermProject:main(): [INFO ]   Lost Packets: 0
+20.000000000s -1 TermProject:main(): [INFO ]   Delay Sum: 0.0020512 s
+20.000000000s -1 TermProject:main(): [INFO ]   Default Delay: 0.0020512 s
+20.000000000s -1 TermProject:main(): [INFO ]   throughput : 0.117005 Mbps
+20.000000000s -1 TermProject:main(): [INFO ] Flow 5 (10.1.3.4->10.1.3.3)
+20.000000000s -1 TermProject:main(): [INFO ]   Lost Packets: 0
+20.000000000s -1 TermProject:main(): [INFO ]   Delay Sum: 0.00359418 s
+20.000000000s -1 TermProject:main(): [INFO ]   Default Delay: 0.00359418 s
+20.000000000s -1 TermProject:main(): [INFO ]   throughput : 0.106839 Mbps
+20.000000000s -1 TermProject:main(): [INFO ] Flow 6 (10.1.2.4->10.1.3.3)
+20.000000000s -1 TermProject:main(): [INFO ]   Lost Packets: 0
+20.000000000s -1 TermProject:main(): [INFO ]   Delay Sum: 0.0418574 s
+20.000000000s -1 TermProject:main(): [INFO ]   Default Delay: 0.00523217 s
+20.000000000s -1 TermProject:main(): [INFO ]   throughput : 0.00998532 Mbps
```

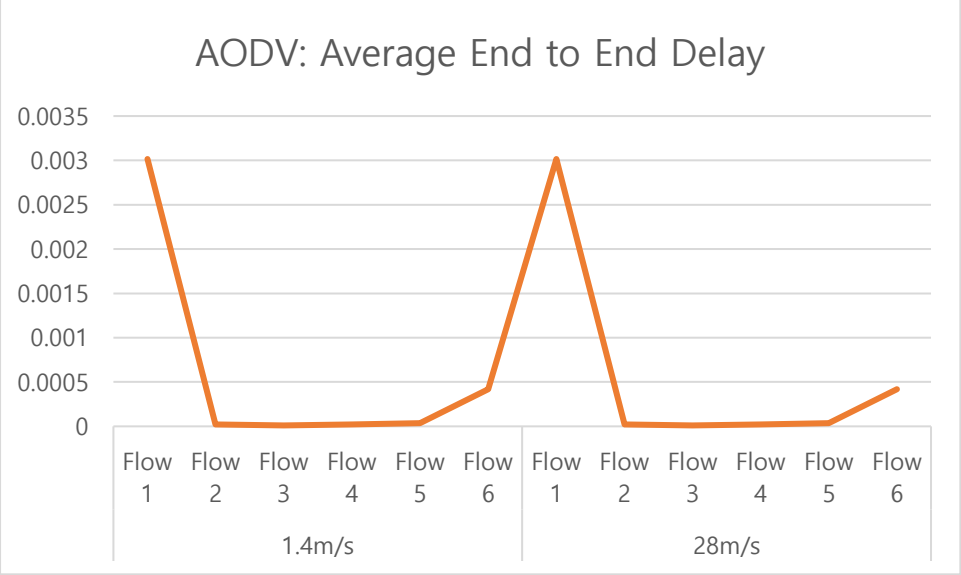
이동하는 모델의 위치와, 각 flow의 성능이 출력된다.

		AODV
1.4m/s	Average Throughput(Mbps)	0.133047502
	Average End to End Delay(s)	0.000586886
	Packet Delivery Ratio(%)	100
28m/s	Average Throughput(Mbps)	0.133047335
	Average End to End Delay(s)	0.000586887
	Packet Delivery Ratio(%)	100

			Throughput(Mbps)	Delay Sum(s)	Average End to End Delay
AODV	1.4m/s	Flow 1	0.00961169	0.301512	0.00301512
		Flow 2	0.184615	0.00208	0.0000208
		Flow 3	0.370228	0.0010372	0.000010372
		Flow 4	0.117005	0.0020512	0.000020512

		Flow 5	0.10684	0.00359416	3.59416E-05
		Flow 6	0.00998532	0.0418572	0.000418572
	28m/s	Flow 1	0.00961169	0.301512	0.00301512
		Flow 2	0.184615	0.00208	0.0000208
		Flow 3	0.370228	0.0010372	0.000010372
		Flow 4	0.117005	0.0020512	0.000020512
		Flow 5	0.106839	0.00359418	3.59418E-05
		Flow 6	0.00998532	0.0418574	0.000418574





AODV의 경우 속도가 높아졌을 때 Throughput이 아주 미세하게 줄었으며, 평균 종단간 지연도 미세하게 늘어났다. 패킷 전달 비율은 모두 100%로 나타났다.

고찰

실험 결과가 어떤 의미를 주는지 설명

AODV 프로토콜의 경우에는 노드의 상태가 변경되면 즉시 경로를 수정하기 때문에 노드가 빠르게 많이 이동하는 경우에는, 라우팅 테이블을 갱신하는 오버헤드가 발생한다.

따라서 속도가 빠를 때는 처리량이 낮아지고, 평균 종단간 지연이 늘어난다.

각 알고리즘의 적합한 환경(네트워크, 조건)

AODV: 경로가 잘 변하지 않고, 작은 네트워크에서 효율적이다.

DSR: 경로가 자주 변하는 환경에서도 다른 프로토콜과 비교해서 빠르게 처리 가능하다. 하지만 경로가 길어지는 경우 오버헤드가 증가할 수 있다.

DSDV: 경로가 자주 변하는 환경에서는 높은 오버헤드가 발생할 수 있다. 따라서 안정적이고 예측 가능한 네트워크 환경이 적합하다.

의문점

AODV는 작동이 잘 되었지만, DSDV와 DSR은 오류가 발생하여 실험하지 못하였다.

결론

주요 실험 결과들 재요약

AODV의 경우 속도가 높아졌을 때 Throughput이 아주 미세하게 줄었으며, 평균 종단간 지연도 미세하게 늘어나는 것을 확인할 수 있었다.