BAB IV

HASIL DAN PEMBAHASAN

4.1. Implementasi

Pada tahap implementasi ini akan membahas mengenai tahapan-tahapan yang dilakukan dalam pembuatan simulasi MANET routing protokol OLSR dan DSDV. Setelah melakukan perancang sebelumnya, kemudian selanjutnya membuat *script* program, menjalankan program simulasi, dan mengumpulkan data yang dihasilkan dari simulasi. Tahapan-tahapan tersebut adalah tahapan yang dilakukan sebelum menganalisis data dan membuat kesimpulan.

Dalam pembuatan program simulasi ini dibangun berdasarkan rancangan tahapan-tahapan simulasi dan *flowchart* yang telah dibuat sebelumnya. Dalam NS3 menyediakan modul-modul dan contoh-contoh *script* yang digunakan untuk membantu dalam pembuatan program ini. Pada program simulasi ini memanfaatkan contoh *script* manet routing *compare* yang tersedia di NS3 yang kemudian akan ditambahkan *script* yang dibutuhkan.

Setelah simulasi program dibuat sesuai rancangan, kemudian simulasi dijalankan, yang akan menghasilkan beberapa *output*, yang berisi kinerja dari routing protokol selama simulasi berjalan. *Output* dari simulasi ini adalah berupa file .xml, .csv dan .flowmon, kemudian akan digunakan untuk pengambilan data untuk di analisis.

4.1.1. Script dan Penjelasan Program Simulasi MANET

1. Fungsi Utama

   Fungsi utama berisi inisialisasi kelas untuk eksperimen, pembuatan file *csv* dan perintah menjalankan simulasi.

```
int
main (int argc, char *argv[])
{
  ManetExperiment experiment;
  std::string CSVfileName = experiment.CommandSetup
(argc,argv);
```

```
  std::ofstream out (CSVfileName.c_str ());
  out << "SimulationSecond," <<
  "ReceiveRate," <<
  "PacketsReceived," <<
  "NumberOfSinks," <<
  "RoutingProtocol," <<
  "TransmissionPower" <<
  std::endl;
  out.close ();
  int nSinks = 4;
  double txp = 5;
  experiment.Run (nSinks, txp, CSVfileName);
}
```

2. Inisialisasi Kelas utama *ManetExperiment*

   Kelas ini merupakan kelas utama yang berisi beberapa variabal utama dan fungsi-fungsi yang akan dijalankan saat simulasi.

```
class ManetExperiment
{
public:
  ManetExperiment ();
  void Run (int nSinks, double txp, std::string
CSVfileName);
  std::string CommandSetup (int argc, char **argv);
private:
  Ptr<Socket> SetupPacketReceive (Ipv4Address addr,
Ptr<Node> node);
  void ReceivePacket (Ptr<Socket> socket);
  void CheckThroughput ();
  uint32_t port;
  uint32_t bytesTotal;
  uint32_t packetsReceived;
  std::string m_CSVfileName;
  int m_nSinks;
  std::string m_protocolName;
  double m_txp;
  uint32_t m_protocol;
  uint32_t m_nodesTotal;
  uint32_t m_packetS;
  std::string m_sizeP;
};
ManetExperiment::ManetExperiment ()
  : port (9),
    bytesTotal (0),
    packetsReceived (0),
    m_CSVfileName ("simulasi-manet-olsr-dsdv.csv"),
    m_protocol (1),
    m_nodesTotal (20),
    m_packetS (1)
{}
```

3. *Print Received Packet*

   Print Received Packet adalah untuk mencetak paket data yang dikirim dan diterima oleh sebuah *node* pada saat simulasi berjalan.

```
static inline std::string
PrintReceivedPacket (Ptr<Socket> socket, Ptr<Packet> packet,
Address senderAddress){
  std::ostringstream oss;
  oss << Simulator::Now ().GetSeconds () << " " << socket-
>GetNode ()->GetId ();
  if (InetSocketAddress::IsMatchingType (senderAddress)){
      InetSocketAddress addr =
InetSocketAddress::ConvertFrom (senderAddress);
      oss << " received one packet from " << addr.GetIpv4
();}
  else{
      oss << " received one packet!";}
  return oss.str ();}
void ManetExperiment::ReceivePacket (Ptr<Socket> socket){
  Ptr<Packet> packet;
  Address senderAddress;
  while ((packet = socket->RecvFrom (senderAddress))){
      bytesTotal += packet->GetSize ();
      packetsReceived += 1;
      NS_LOG_UNCOND (PrintReceivedPacket (socket, packet,
senderAddress));}
}
Ptr<Socket>
ManetExperiment::SetupPacketReceive (Ipv4Address addr,
Ptr<Node> node){
  TypeId tid = TypeId::LookupByName
("ns3::UdpSocketFactory");
  Ptr<Socket> sink = Socket::CreateSocket (node, tid);
  InetSocketAddress local = InetSocketAddress (addr, port);
  sink->Bind (local);
  sink->SetRecvCallback (MakeCallback
(&ManetExperiment::ReceivePacket, this));
  return sink;
}
```

4. Perhitungan *Throughput*

   a. Cek dan cetak *received rate* dan waktu simulasi untuk menghitung *throughput*.

   Program akan menghitung berapa besarnya data yang diterima pada masing-masing *node* dalam setiap detiknya selama simulasi berjalan. Kemudian data-data tersebut akan di catat ke dalam file .csv.

```
void ManetExperiment::CheckThroughput (){
  double kbs = (bytesTotal * 8.0) / 1024;
```

```
  bytesTotal = 0;
  std::ofstream out (m_CSVfileName.c_str (), std::ios::app);
  out << (Simulator::Now ()).GetSeconds () << ","
      << kbs << ","
      << packetsReceived << ","
      << m_nSinks << ","
      << m_protocolName << ","
      << m_txp << ""
      << std::endl;
  out.close ();
  packetsReceived = 0;
  Simulator::Schedule (Seconds (1.0),
&ManetExperiment::CheckThroughput, this);
}
```

b. Set *sinks* dan *source node*

Program akan menentukan *node* yang akan mengirim paket (sumber) dan *node* yang akan menerima paket (tujuan). Kemudian juga ditentukan kapan *node* akan memulai mengirim paket dan kapan *node* berhenti mengirim paket.

```
  for (int i = 0; i < nSinks; i++)
    {
      Ptr<Socket> sink = SetupPacketReceive
(adhocInterfaces.GetAddress (i), adhocNodes.Get (i));
      AddressValue remoteAddress (InetSocketAddress
(adhocInterfaces.GetAddress (i), port));
      onoff1.SetAttribute ("Remote", remoteAddress);
      Ptr<UniformRandomVariable> var =
CreateObject<UniformRandomVariable> ();
      ApplicationContainer temp = onoff1.Install
(adhocNodes.Get (i + nSinks));
      temp.Start (Seconds (var->GetValue (50.0,51.0)));
      temp.Stop (Seconds (TotalTime));
    }
```

5. Fungsi *Command Setup*

Fungsi *Command Setup* merupakan fungsi untuk menyimpan nilai parameter yang di masukkan oleh *user* saat menjalankan perintah ./waf menggunakan terminal. Nilai parameter itu di isi oleh *user* sesuai dengan skenario yang ingin dijalankan, seperti jumlah *node* dan ukuran paket data.

```
std::string ManetExperiment::CommandSetup (int argc, char
**argv)
{
  CommandLine cmd;
  cmd.AddValue ("csv", "The name of the CSV output file
name", m_CSVfileName);
```

```
  cmd.AddValue ("protocol", "1=OLSR;2=DSDV", m_protocol);
  cmd.AddValue ("nodes", "Number of Nodes", m_nodesTotal);
  cmd.AddValue ("packet", "Packet Size", m_packetS);
  cmd.Parse (argc, argv);
  return m_CSVfileName;
}
```

6. Pembuatan *Node*

   Program akan membuat *node* terlebih dahulu yang merupakan konsep dari NS3.

```
 NodeContainer adhocNodes;
 adhocNodes.Create (nodesTotal);
```

7. Inisialisasi *Wifi Settings* dan *Wifi Phy channel*

   Wifi settings adalah jenis *wifi* yang digunakan pada simulasi ini. *Wifi* berfungsi sebagai sarana pengirim dan penerima data pada program. Kemudian program membuat *channel* yang berfungsi sebagai penghubung antar *node*.

```
  WifiHelper wifi;
  wifi.SetStandard (WIFI_PHY_STANDARD_80211b);

  YansWifiPhyHelper wifiPhy =  YansWifiPhyHelper::Default
();
  YansWifiChannelHelper wifiChannel;
  wifiChannel.SetPropagationDelay
("ns3::ConstantSpeedPropagationDelayModel");
  wifiChannel.AddPropagationLoss
("ns3::FriisPropagationLossModel");
  wifiPhy.SetChannel (wifiChannel.Create ());
```

8. Transmisi *Power*

   Transmisi *power* adalah penentu besarnya daya yang akan digunakan untuk mengirim paket.

```
 wifiPhy.Set ("TxPowerStart",DoubleValue (txp));
 wifiPhy.Set ("TxPowerEnd", DoubleValue (txp));
```

9. Membuat *device*

   Pembuatan *device* untuk setiap *node* yang berfungsi sebagai penghubung antara *node* dan *channel.*

```
 wifiMac.SetType ("ns3::AdhocWifiMac");
 NetDeviceContainer adhocDevices = wifi.Install (wifiPhy,
```

```
wifiMac,adhocNodes);
```

10. Dimensi wilayah simulasi

    Program akan membuat dimensi wilayah sebagai tempat simulasi berjalan. Dimensi tersebut di representasikan dengan bentuk axis X dan Y. Satuan dari dimensi tersebut adalah meter. Wilayah simulasi tersebut berbentuk persegi atau persegi panjang sesuai ukuran yang dimasukkan pada X dan Y. Wilayah simulasi dapat dilihat secara visual menggunakan *file* .xml yang dijalan pada aplikasi NetAnim.

```
ObjectFactory pos;
  pos.SetTypeId ("ns3::RandomRectanglePositionAllocator");
  pos.Set ("X", StringValue
("ns3::UniformRandomVariable[Min=0.0|Max=500.0]"));
  pos.Set ("Y", StringValue
("ns3::UniformRandomVariable[Min=0.0|Max=500.0]"));
```

11. *Set* posisi awal *node* dan kecepatan *node* menjadi *static*

    Pada skenario penambahan *node*, dibutuhkan posisi awal *node* yang tetap sama pada routing protokol OLSR atau DSDV, maupun pada jumlah *node* 20 atau 40, agar validitas dan konsistensi skenario tetap terjamin saat ingin dibandingkan. Sehingga pada langkah ini di konfigurasi *steamIndex* menjadi 0. Kemudian program juga mengatur pergerakan dan kecepatan *node* agar tetap dan konsisten selama simulasi berjalan.

```
 Ptr<PositionAllocator> taPositionAlloc = pos.Create ()-
>GetObject<PositionAllocator> ();
  streamIndex += taPositionAlloc->AssignStreams
(streamIndex);
  std::stringstream ssSpeed;
  ssSpeed << "ns3::UniformRandomVariable[Min=0.0|Max=" <<
nodeSpeed << "]";
  std::stringstream ssPause;
  ssPause << "ns3::ConstantRandomVariable[Constant=" <<
nodePause << "]";
  mobilityAdhoc.SetMobilityModel
("ns3::RandomWaypointMobilityModel",
     "Speed", StringValue (ssSpeed.str ()),
     "Pause", StringValue (ssPause.str ()),
     "PositionAllocator", PointerValue (taPositionAlloc));
  mobilityAdhoc.SetPositionAllocator (taPositionAlloc);
  mobilityAdhoc.Install (adhocNodes);
  streamIndex += mobilityAdhoc.AssignStreams (adhocNodes,
streamIndex)
```

12. Inisialisasi dan pengaturan routing protokol OLSR dan DSDV

    Program akan melakukan inisialisasi routing protokol, kemudian program akan memilih routing protokol OLSR atau DSDV sesuai dengan perintah yang dimasukkan *user*.

```
OlsrHelper olsr;
DsdvHelper dsdv;
Ipv4ListRoutingHelper list;
InternetStackHelper internet;
switch (m_protocol)
  {
  case 1:
    list.Add (olsr, 100);
    m_protocolName = "OLSR";
    internet.SetRoutingHelper (list);
    internet.Install (adhocNodes);
    break;
  case 2:
    list.Add (dsdv, 100);
    m_protocolName = "DSDV";
    internet.SetRoutingHelper (list);
    internet.Install (adhocNodes);
    break;
  default:
    NS_FATAL_ERROR ("No such protocol:" << m_protocol);
  }
```

13. Set *IP Address*

    Program membuat *range IP Address*, kemudian akan memberikan *IP* pada setiap *node*.

```
Ipv4AddressHelper addressAdhoc;
addressAdhoc.SetBase ("10.1.1.0", "255.255.255.0");
Ipv4InterfaceContainer adhocInterfaces;
adhocInterfaces = addressAdhoc.Assign (adhocDevices);
```

14. Perhitungan pengiriman paket menggunakan *FlowMonitor*

    Program akan melakukan perhitungan pengiriman paket seperti *delay*, jumlah paket dikirim dan diterima, waktu pengiriman, dan paket yang hilang dengan menggunakan *module FlowMonitor* yang ada di NS3. Setelah program melakukan perhitungan, kemudian hasilnya akan dicetak kedalam sebuah file .flowmon.

```
Ptr<FlowMonitor> flowmon;
FlowMonitorHelper flowmonHelper;
```

```
  flowmon = flowmonHelper.InstallAll ();
flowmon->SerializeToXmlFile ((tr_name + ".flowmon").c_str(),
false, false);
```

15. Membuat *AnimationInterface*

    Setelah melakukan perhitungan paket data dan mencatat pergerakan *node*, kemudian program akan mencetak hasil *trace node* dalam bentuk animasi yang akan disimpan kedalam file .xml.

```
AnimationInterface anim ((tr_name + ".xml").c_str());
```

16. Simulasi dimulai dan berhenti

```
 Simulator::Stop (Seconds (TotalTime));
 Simulator::Run ();
. . .
 Simulator::Destroy ();
```

4.1.2. Tahapan-Tahapan Menjalankan Simulasi

Setelah implementasi pembuatan program selesai, kemudian program akan dijalankan dengan perintah-perintah tertentu yang sudah di konfigurasi sebelumnya. Perintah-perintah tersebut akan menyesuaikan dengan skenario simulasi yang akan dijalankan.

1. Sebelum menjalankan program, *user* terlebih dahulu harus masuk ke direktori aplikasi NS3 yang terdapat *file* .waf. Kemudian *user* membuka *terminal* untuk mengetik perintah menjalankan program simulasi, yaitu :

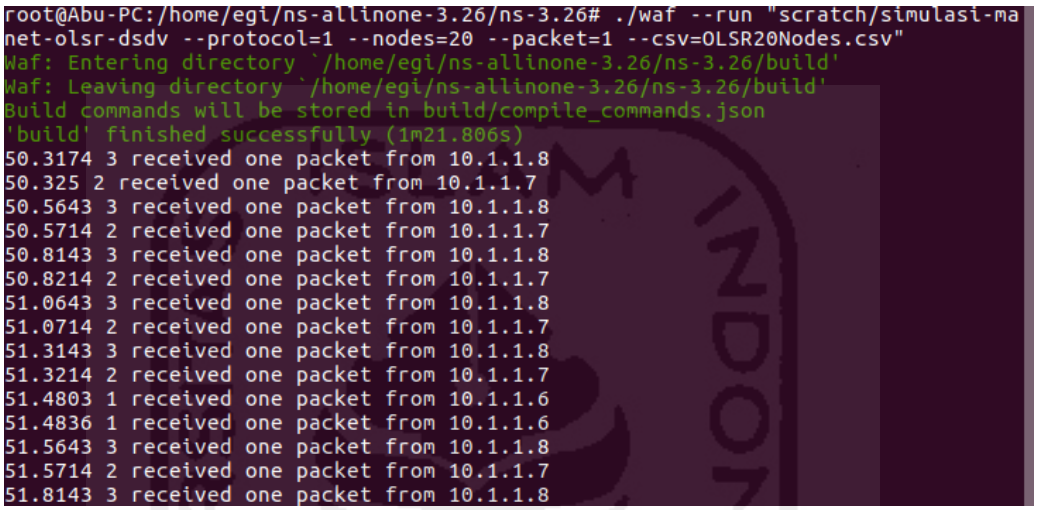
```
./waf --run "manet-routing --protokol=1 --nodes=20 --paket=1
csv=OLSR20Node.csv"
```

Penjelasan:

./waf --run	: perintah *default* NS3 untuk menjalankan program
manet-routing	: nama *file* program simulasi
protokol	: jenis protokol yang dipilih, 1=OLSR dan 2=DSDV
nodes	: jumlah *node*
paket	: ukuran paket data, 1=64 *bytes* dan 2=128 *bytes*

csv : nama *file* .csv.

2. Setelah menjalankan perintah eksekusi program, kemudian NS3 akan melakukan simulasi dengan mem-*build* dan meng-*compile* program dan simulasi berjalan, seperti pada gambar 4.1.

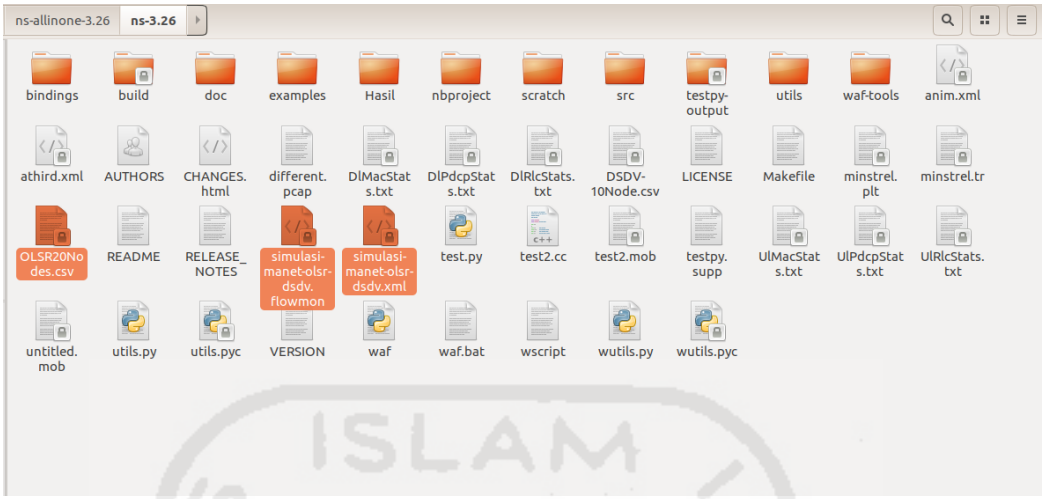


Gambar 4.1. Memulai simulasi

3. Kemudian simulasi berakhir dan NS3 akan berhenti.

4.1.3. Tahapan Pengambilan Data

Setelah melakukan tahapan-tahapan menjalankan simulasi seperti diatas, NS3 akan menghasilkan output 3 buah *file* yaitu .csv, .flowmon dan .xml seperti pada gambar 4.2. Ketiga *file* tersebut akan digunakan untuk menganalisis data yang akan dijelaskan pada langkah selanjutnya.



Gambar 4.2. *Output file*

1. *File* .csv

File ini berisi beberapa informasi, yaitu lamanya waktu simulasi, besar data yang diterima, paket yang diterima, jumlah koneksi, jenis protokol, dan daya transmisi. *File* .csv ini dibutuhkan saat analisis data untuk menghitung nilai throughput. Untuk menghitung nilai throughput perlu diketahui total paket yang diterima dan lamanya waktu pengiriman paket dihitung dari pertama kali paket dikirim sampai dengan paket yang terakhir diterima. Isi dari *file* .csv dapat dilihat pada gambar 4.3 dan 4.4.

SimulationSecond	ReceiveRate	PacketsReceived	NumberOfSinks	RoutingProtocol	TransmissionPower
0	0	0	4	OLSR	5
1	0	0	4	OLSR	5
2	0	0	4	OLSR	5
3	0	0	4	OLSR	5
4	0	0	4	OLSR	5
5	0	0	4	OLSR	5
6	0	0	4	OLSR	5
7	0	0	4	OLSR	5
8	0	0	4	OLSR	5
9	0	0	4	OLSR	5
10	0	0	4	OLSR	5
11	0	0	4	OLSR	5
12	0	0	4	OLSR	5
13	0	0	4	OLSR	5
14	0	0	4	OLSR	5
15	0	0	4	OLSR	5
16	0	0	4	OLSR	5
17	0	0	4	OLSR	5
18	0	0	4	OLSR	5
19	0	0	4	OLSR	5
20	0	0	4	OLSR	5
21	0	0	4	OLSR	5
22	0	0	4	OLSR	5
23	0	0	4	OLSR	5
24	0	0	4	OLSR	5

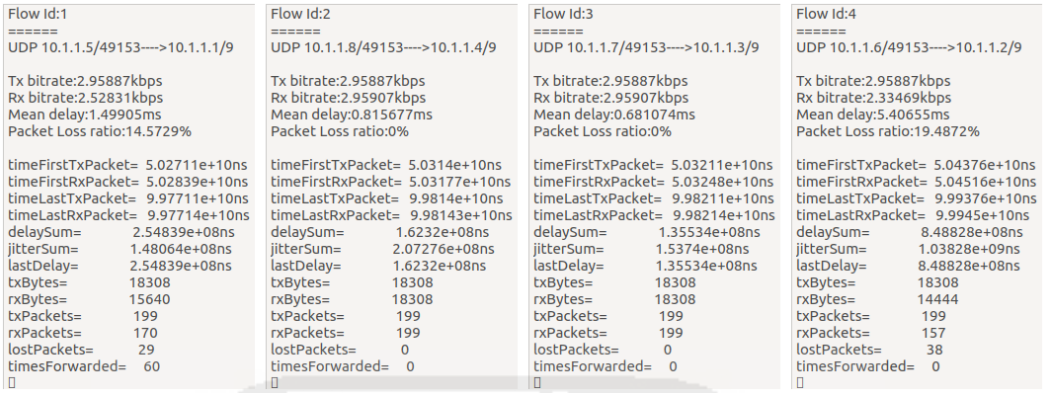
Gambar 4.3. *File* .csv skenario 20 *node* OLSR (1)

74	8	16	4	OLSR	5
75	8	16	4	OLSR	5
76	8	16	4	OLSR	5
77	8	16	4	OLSR	5
78	8	16	4	OLSR	5
79	8	16	4	OLSR	5
80	8	16	4	OLSR	5
81	8	16	4	OLSR	5
82	8	16	4	OLSR	5
83	8	16	4	OLSR	5
84	8	16	4	OLSR	5
85	8	16	4	OLSR	5
86	8	16	4	OLSR	5
87	8	16	4	OLSR	5
88	8	16	4	OLSR	5
89	8	16	4	OLSR	5
90	8	16	4	OLSR	5
91	8	16	4	OLSR	5
92	8	16	4	OLSR	5
93	8	16	4	OLSR	5
94	8	16	4	OLSR	5
95	7.5	15	4	OLSR	5
96	8	16	4	OLSR	5
97	8	16	4	OLSR	5
98	8	16	4	OLSR	5
99	8	16	4	OLSR	5

Gambar 4.4. *File* .csv skenario 20 *node* OLSR (2)

2. *Flie* .flowmon

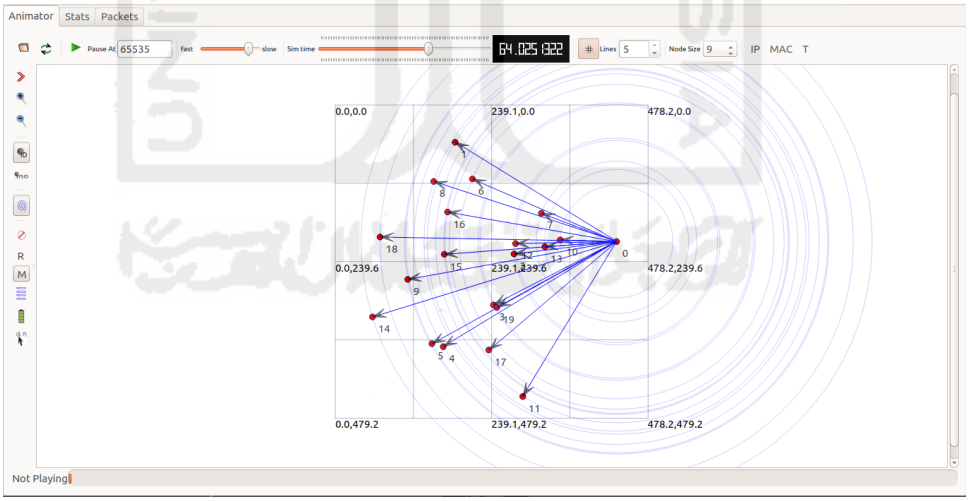
File *FlowMonitor* yang berekstensi .flowmon ini berisi beberapa informasi yang berkaitan dengan nilai-nilai parameter paket data, seperti *delay*, paket yang hilang, paket dikirim dan paket diterima. *File* .flowmon ini dibutuhkan saat analisis data untuk menghitung PDR (*packet delivery ratio*), *packet loss* dan *delay*. Untuk menghitung PDR dan *packet loss* perlu diketahui paket yang dikirim dapat dilihat pada *txPackets* dan paket yang diterima dapat dilihat pada *rxPackets*. Sedangkan untuk nilai *delay*, pada *FlowMonitor* nilai *delay* secara otomatis telah dihitung, hanya saja perlu menghitung rata-rata delay dari empat koneksi tersebut. *File* .flowmon ini dapat dibuka menggunakan aplikasi NetAnim. Isi dari file .flowmon dapat dilihat pada gambar 4.5.



Gambar 4.5. *File* .flowmon skenario ukuran paket 64 *bytes* DSDV

3. *File* .xml

Simulasi yang sudah dijalankan dapat dilihat prosesnya dalam bentuk animasi, dengan cara membuka *file* .xml menggunakan aplikasi NetAnim. *File* .xml memperlihatkan bagaimana simulasi berjalan mulai dari *broadcast*, proses pengiriman paket data dan pergerakan *node-node*. Animasi dapat dilihat pada gambar 4.6.



Gambar 4.6. Animasi simulasi

4.2. Analisis Data

Pada tahapan analisis data ini akan melakukan perbandingan kinerja routing protokol OLSR dan DSDV setelah melakukan perhitungan parameter *QoS*. Analisis

data dilakukan dengan menghitung nilai-nilai yang terdapat di *file* yang dihasilkan dari simulasi, yaitu *file* .csv dan .flowmon.

Pertama, untuk menghitung *throughput* dengan membuka file .csv, kemudian hitung total dari *recevie rate* dan selisih waktu pertama kali paket dikirim hingga terakhir kali paket diterima. Nilai *throughput* didapat setelah membagi total *recevie rate* dengan waktu pengiriman paket.

Selanjutnya, untuk menghitung PDR, *packet loss* dan *delay* dengan membuka file .flowmon. Untuk menghitung PDR, membagi *rxPackets* dengan *txPackets*, kemudian dirubah dalam bentuk persen dengan cara dikalikan dengan 100%. Untuk menghitung *packet loss*, *txPackets* dikurang *rxPackets* kemudian hasilnya dibagi *txPackets*, kemudian dirubah dalam bentuk persen dengan cara dikalikan dengan 100%. Untuk menghitung *delay* dengan melihat *Mean delay* kemudian dijumlah *Mean delay* di setiap koneksi dan dibagi dengan jumlah koneksi.

4.2.1. Hasil Simulasi Untuk Skenario Penambahan *Node*

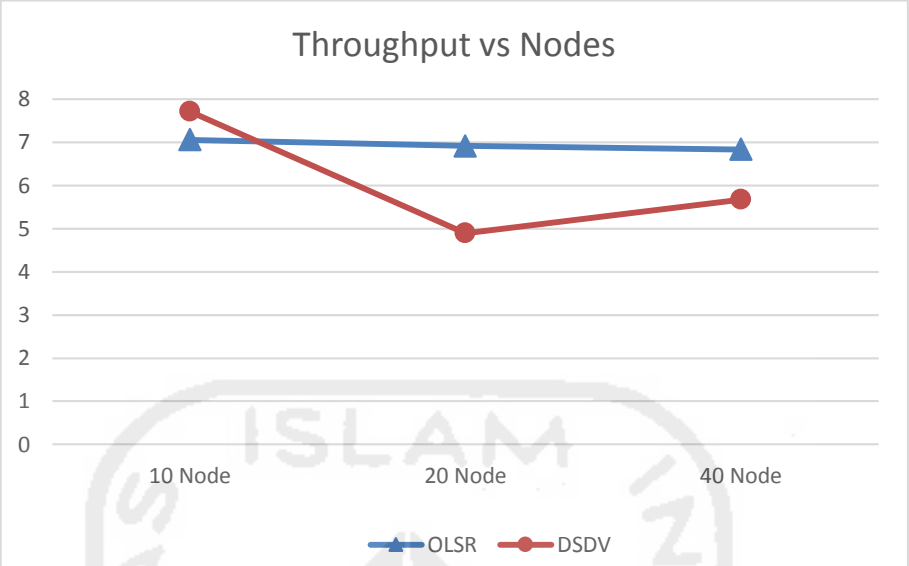
Setelah hasil simulasi skenario penambahan *node* selesai kemudian akan dianalisis yang akan dibagi dalam 4 bagian sesuai dengan parameter *QoS* yang diuji.

4.2.1.1. *Throughput*

Hasil dari perhitungan rata-rata *throughput* pada perbandingan performa routing protokol OLSR dan DSDV yang didapat dari simulasi untuk skenario penambahan *node* yang telah dijalankan, hasil tersebut dapat di lihat pada tabel 4.1 dan gambar 4.7. nilai rata-rata *throughput* ini menggunakan satuan Kbps (*Kilo bit per second*).

Tabel 4.1. Nilai rata-rata *throughput* pada skenario penambahan *node*

Protokol	10 Node	20 Node	40 Node	AVG
OLSR	7,061224489	6,918367346	6,836734693	6,938775510
DSDV	7,714285714	4,897959183	5,673469387	6,095238095



Gambar 4.7. Grafik *throughput* vs *nodes*

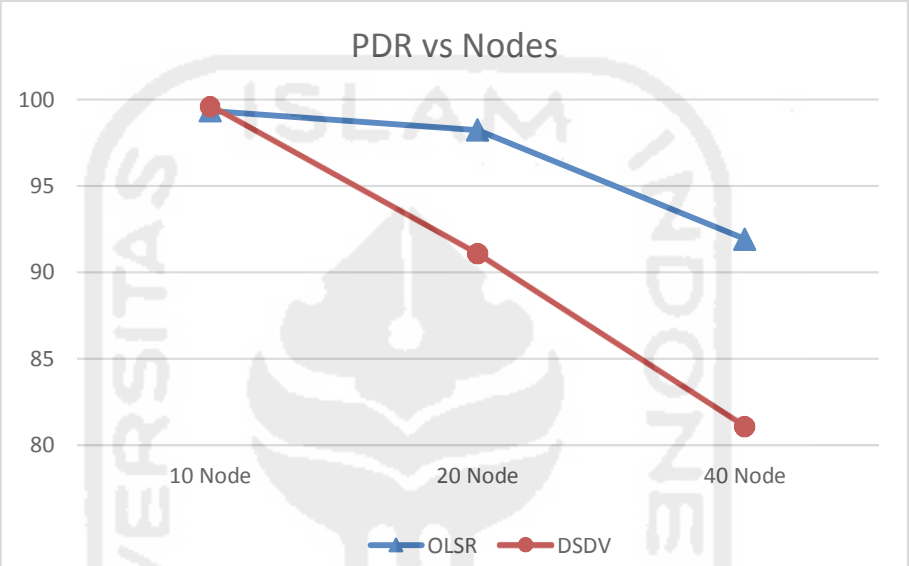
Berdasarkan grafik pada gambar 4.7, dapat dilihat routing protokol OLSR lebih baik dibandingkan dengan DSDV dari nilai performa rata-rata throughput terhadap penambahan node. OLSR menunjukkan nilai rata-rata 6,938775510 sedangkan DSDV menunjukkan nilai rata-rata 6,095238095. Hal ini dikarenakan DSDV menyimpan seluruh informasi ke semua node dan mem-*broadcast* secara berkala untuk memperbarui tabel routing. *Broadcast* yang dilakukan DSDV ini mengakibatkan penggunaan *bandwidth* yang berlebih, yang seharusnya dapat digunakan untuk mengirim paket data. Sedangkan OLSR menggunakan metode MPR dalam memperbarui tabel routing. Jadi dalam OLSR hanya *node* MPR saja yang mengirimkan pesan *broadcast*. Hal tersebut dapat mengurangi penggunaan *bandwidth* yang berlebihan.

4.2.1.2. *Packet Delivery Ratio*

Hasil dari perhitungan rata-rata PDR pada perbandingan performa routing protokol OLSR dan DSDV yang didapat dari simulasi untuk skenario penambahan *node* yang telah dijalankan, hasil tersebut dapat di lihat pada tabel 4.2 dan gambar 4.8. nilai rata-rata PDR ini menggunakan satuan persentase.

Tabel 4.2. Nilai rata-rata PDR pada skenario penambahan *node*

Protokol	10 Node	20 Node	40 Node	AVG
OLSR	99,36%	98,24%	91,92%	96,51%
DSDV	99,62%	91,08%	81,08%	90,59%



Gambar 4.8. Grafik PDR vs *nodes*

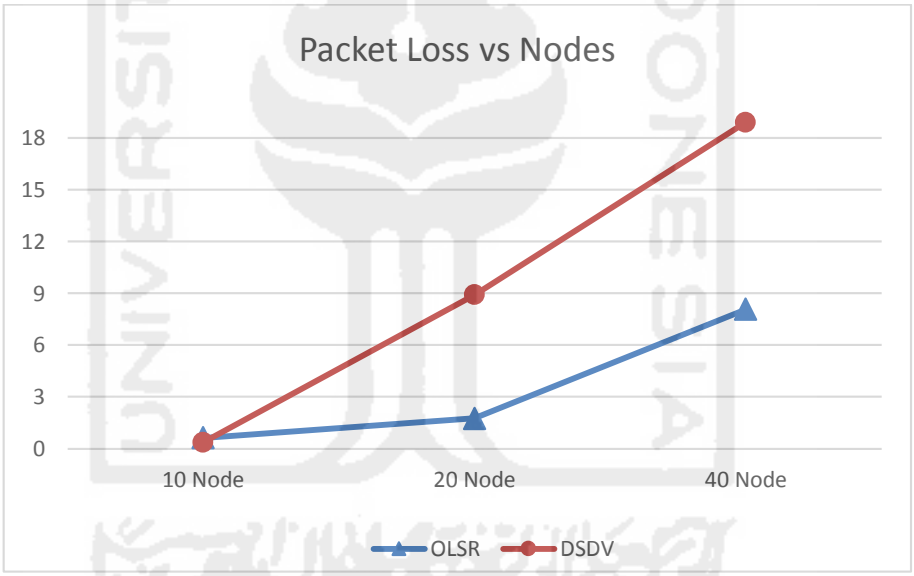
Berdasarkan grafik pada gambar 4.8, dapat dilihat routing protokol OLSR lebih baik dibandingkan dengan DSDV dari nilai performa rata-rata PDR terhadap penambahan node. OLSR menunjukkan nilai rata-rata 96,51% sedangkan DSDV menunjukkan nilai rata-rata 90,59%. Hal ini disebabkan kepadatan *node* yang menyebabkan terbentuknya banyak *hop* dan juga memungkinkan terjadinya perubahan jalur. Pada OLSR hal tersebut dapat teratasi karena memiliki sistem MPR. Sedangkan pada DSDV yang harus memperbarui tabel routing secara berkala. Hal tersebut menyebabkan ketika mengirim paket dan kehilangan jalur maka pengiriman paket harus menunggu hingga proses pembaruan routing tabel selesai untuk menentukan jalur pengiriman.

4.2.1.3. *Packet Loss*

Hasil dari perhitungan rata-rata *packet loss* pada perbandingan performa routing protokol OLSR dan DSDV yang didapat dari simulasi untuk skenario penambahan *node* yang telah dijalankan, hasil tersebut dapat di lihat pada tabel 4.3 dan gambar 4.9. nilai rata-rata *packet loss* ini menggunakan satuan persentase.

Tabel 4.3. Nilai rata-rata *packet loss* pada skenario penambahan *node*

Protokol	10 Node	20 Node	40 Node	AVG
OLSR	0,63%	1,75%	8,07%	3,48%
DSDV	0,37%	8,91%	18,91%	9,40%



Gambar 4.9. Grafik *packet loss* vs *nodes*

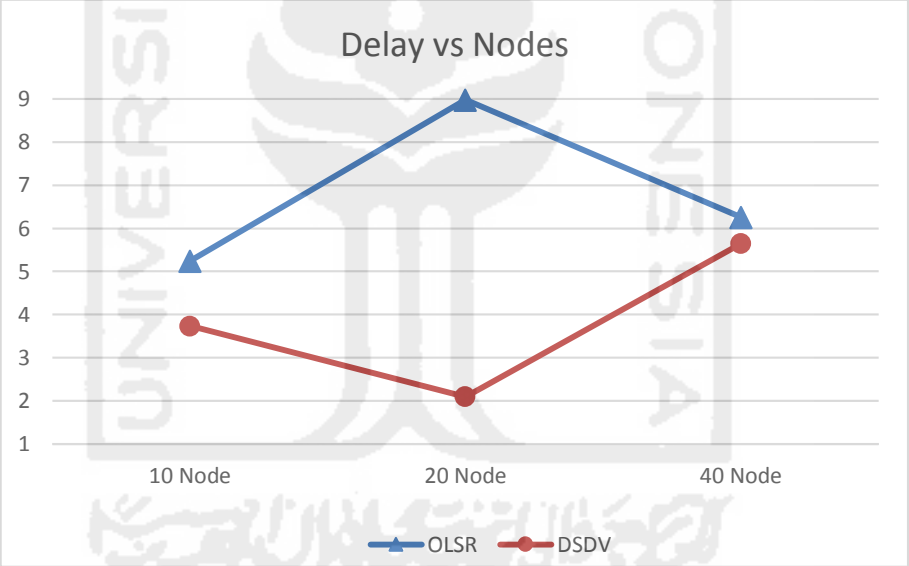
Berdasarkan grafik pada gambar 4.9, dapat dilihat routing protokol OLSR lebih baik dibandingkan dengan DSDV dari nilai performa rata-rata *packet loss* terhadap penambahan *node*. OLSR menunjukkan nilai rata-rata 3,48% sedangkan DSDV menunjukkan nilai rata-rata 9,40%. Sama halnya dengan PDR, hal ini disebabkan kepadatan *node* yang besar, memungkinkan hilangnya paket data saat pengiriman.

4.2.1.4. *Delay*

Hasil dari perhitungan rata-rata *delay* pada perbandingan performa routing protokol OLSR dan DSDV yang didapat dari simulasi untuk skenario penambahan *node* yang telah dijalankan, hasil tersebut dapat di lihat pada tabel 4.4 dan gambar 4.10. nilai rata-rata *delay* ini menggunakan satuan *milliseconds*.

Tabel 4.4. Nilai rata-rata *delay* pada skenario penambahan *node*

Protokol	10 Node	20 Node	40 Node	AVG
OLSR	5,2439447	8,9745405	6,2553335	6,8246062
DSDV	3,7331015	2,1005877	5,6479125	3,8272005



Gambar 4.10. Grafik *delay* vs *nodes*

Berdasarkan grafik pada gambar 4.10, dapat dilihat routing protokol DSDV lebih baik dibandingkan dengan OLSR dari nilai performa rata-rata *delay* terhadap penambahan *node*. OLSR menunjukkan nilai rata-rata 6,8246062 sedangkan DSDV menunjukkan nilai rata-rata 3,8272005. Hal ini disebabkan sifat DSDV yang selalu melakukan pembaruan tabel routing secara berkala. Sehingga DSDV dapat dengan cepat menentukan jalur pengiriman, karena sudah terdapat pada tabel routing.

Melihat performa routing protokol berdasarkan nilai *QoS* terhadap skenario penambahan *node*. Diketahui bahwa routing protokol OLSR lebih baik pada nilai rata-rata *throughput*, PDR dan *packet loss* dibandingkan dengan DSDV. Sedangkan routing protokol DSDV lebih baik pada nilai rata-rata *delay* dibandingkan dengan OLSR. Diketahui juga bahwa nilai *throughput*, PDR dan *packet loss* DSDV pada skenario penambahan *node* dengan jumlah *node* yang sedikit, DSDV lebih baik dibandingkan OLSR. DSDV lebih cocok digunakan untuk kepadatan jaringan berskala kecil (Dabungke, Wahidah, & Mulyana, 2009). Sedangkan OLSR cocok digunakan untuk kepadatan jaringan berskala menengah ke atas. Hal tersebut disebabkan sistem MPR yang dimiliki OLSR, yang membuat proses pengiriman paket data lebih cepat.

4.2.2. Hasil Simulasi Untuk Skenario Penambahan Ukuran Paket Data

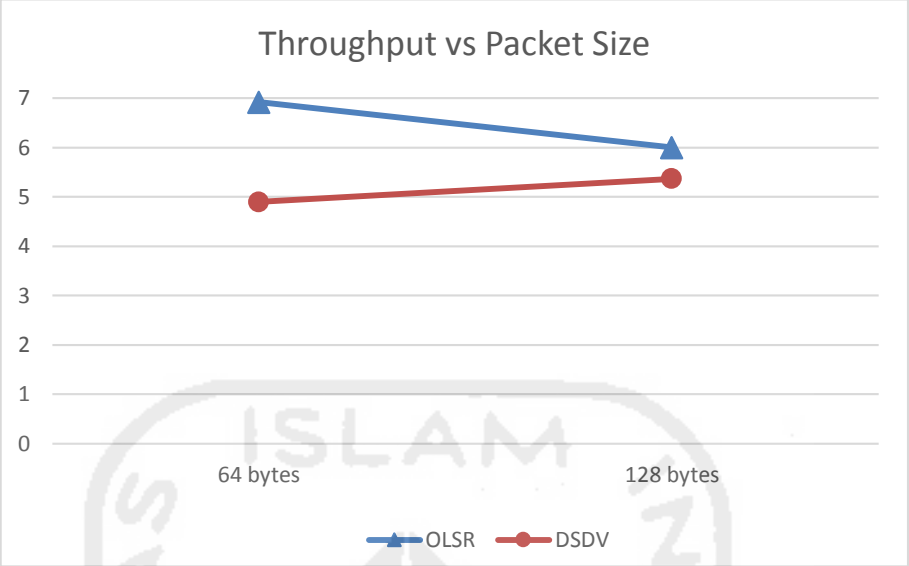
Setelah hasil simulasi skenario penambahan ukuran paket data selesai kemudian akan dianalisis yang akan dibagi dalam 4 bagian sesuai dengan parameter *QoS* yang diuji.

4.2.2.1. *Throughput*

Hasil dari perhitungan rata-rata *throughput* pada perbandingan performa routing protokol OLSR dan DSDV yang didapat dari simulasi untuk skenario penambahan ukuran paket data yang telah dijalankan, hasil tersebut dapat di lihat pada tabel 4.5 dan gambar 4.11. nilai rata-rata *throughput* ini menggunakan satuan Kbps (*Kilo bit per second*).

Tabel 4.5. Nilai rata-rata *throughput* pada skenario penambahan ukuran paket

Protokol	64 bytes	128 bytes	AVG
OLSR	6,918367346	6	6,459183673
DSDV	4,897959183	5,367346938	5,132653061



Gambar 4.11. Grafik *throughput* vs *packet size*

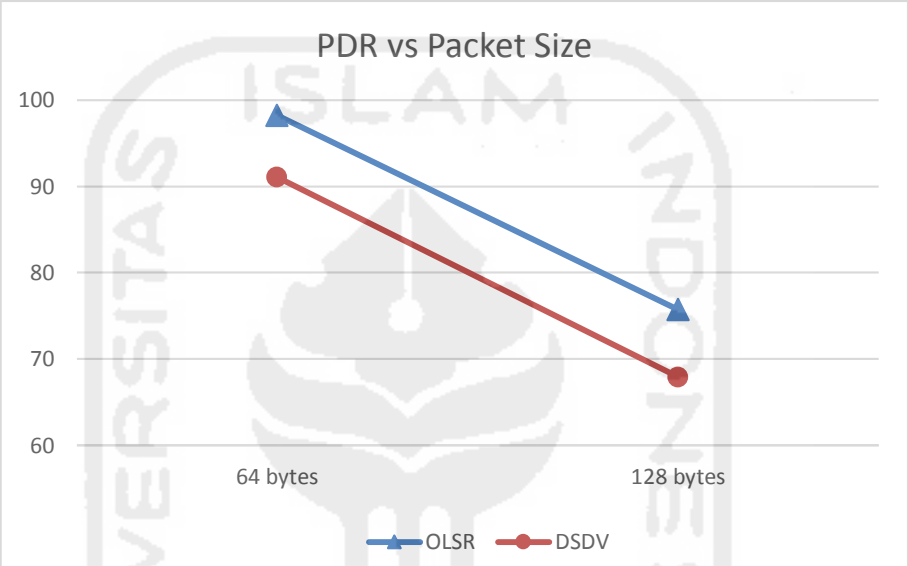
Berdasarkan grafik pada gambar 4.11, dapat dilihat routing protokol OLSR lebih baik dibandingkan dengan DSDV dari nilai performa rata-rata *throughput* terhadap penambahan ukuran paket data. OLSR menunjukkan nilai rata-rata 6,459183673 sedangkan DSDV menunjukkan nilai rata-rata 5,132653061. Hal ini dikarenakan DSDV menyimpan seluruh informasi ke semua *node* dan mem-*broadcast* secara berkala untuk memperbarui tabel routing. *Broadcast* yang dilakukan DSDV ini mengakibatkan penggunaan *bandwidth* yang berlebih, yang seharusnya dapat digunakan untuk mengirim paket data, terlebih lagi ukuran paket yang bertambah. Sedangkan OLSR menggunakan metode MPR dalam memperbarui tabel routing. Jadi dalam OLSR hanya node MPR saja yang mengirimkan pesan *broadcast*. Hal tersebut dapat mengurangi penggunaan *bandwidth* yang berlebihan.

4.2.2.2. *Packet Delivery Ratio*

Hasil dari perhitungan rata-rata PDR pada perbandingan performa routing protokol OLSR dan DSDV yang didapat dari simulasi untuk skenario penambahan ukuran paket data yang telah dijalankan, hasil tersebut dapat di lihat pada tabel 4.6 dan gambar 4.12. Nilai rata-rata PDR ini menggunakan satuan persentase.

Tabel 4.6. Nilai rata-rata PDR pada skenario penambahan ukuran paket

Protokol	64 bytes	128 bytes	AVG
OLSR	98,24%	75,75%	86,99%
DSDV	91,08%	67,92%	79,50%



Gambar 4.12. Grafik PDR vs *packet size*

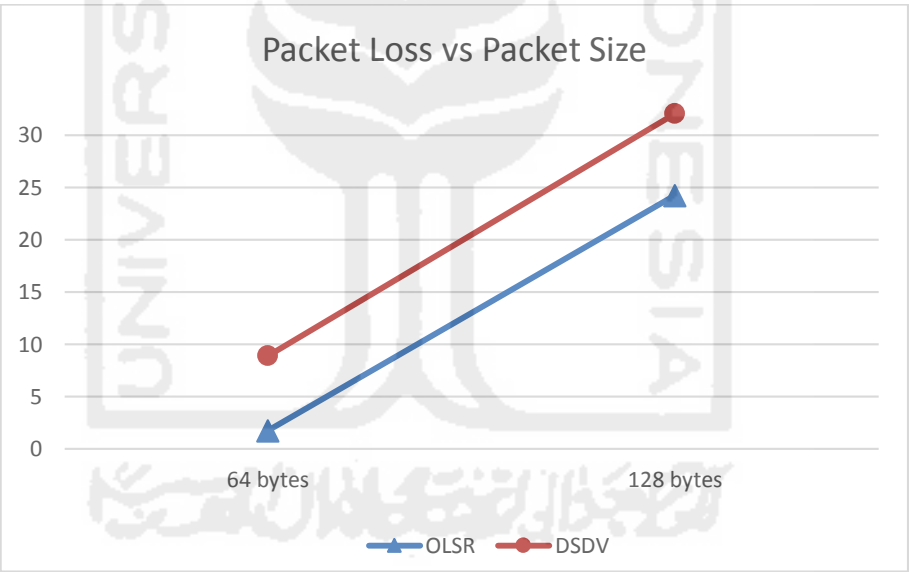
Berdasarkan grafik pada gambar 4.12, dapat dilihat routing protokol OLSR lebih baik dibandingkan dengan DSDV dari nilai performa rata-rata PDR terhadap penambahan ukuran paket data. OLSR menunjukkan nilai rata-rata 86,99% sedangkan DSDV menunjukkan nilai rata-rata 79,50%. Hal ini disebabkan ukuran paket data yang besar berpengaruh pada pengiriman paket. Pada OLSR hal tersebut dapat teratasi karena memiliki sistem MPR sehingga beban paket hanya akan disebar pada *node* MPR saja. Sedangkan pada DSDV yang harus memperbarui tabel routing secara berkala. Hal tersebut menyebabkan ketika mengirim paket dan kehilangan jalur maka pengiriman paket harus menunggu hingga proses pembaruan routing tabel selesai untuk menentukan jalur pengiriman.

4.2.2.3. *Packet Loss*

Hasil dari perhitungan rata-rata *packet loss* pada perbandingan performa routing protokol OLSR dan DSDV yang didapat dari simulasi untuk skenario penambahan ukuran paket data yang telah dijalankan, hasil tersebut dapat di lihat pada tabel 4.7 dan gambar 4.13. nilai rata-rata *packet loss* ini menggunakan satuan persentase.

Tabel 4.7. Nilai rata-rata *packet loss* pada skenario penambahan ukuran paket

Protokol	64 bytes	128 bytes	AVG
OLSR	1,75%	24,24%	13,07%
DSDV	8,91%	32,07%	20,49%



Gambar 4.13. Grafik *packet loss* vs *packet size*

Berdasarkan grafik pada gambar 4.13, dapat dilihat routing protokol OLSR lebih baik dibandingkan dengan DSDV dari nilai performa rata-rata *packet loss* terhadap penambahan ukuran paket data. OLSR menunjukkan nilai rata-rata 13,07% sedangkan DSDV menunjukkan nilai rata-rata 20,49%. Sama halnya dengan PDR, hal ini disebabkan ukuran paket yang besar, memungkinkan

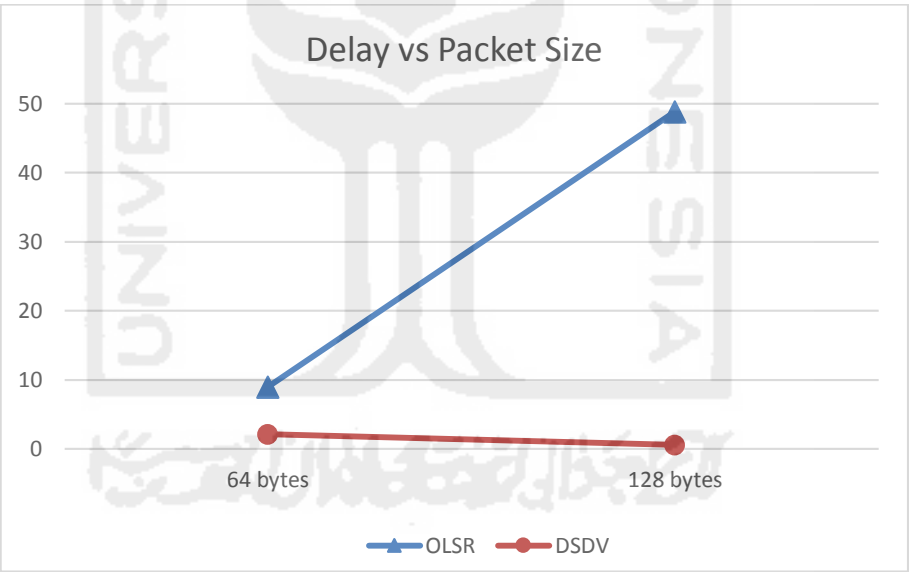
hilangnya paket data saat pengiriman.

4.2.2.4. *Delay*

Hasil dari perhitungan rata-rata *delay* pada perbandingan performa routing protokol OLSR dan DSDV yang didapat dari simulasi untuk skenario penambahan node yang telah dijalankan, hasil tersebut dapat di lihat pada tabel 4.8 dan gambar 4.14. nilai rata-rata *delay* ini menggunakan satuan *milliseconds*.

Tabel 4.8. Nilai rata-rata *delay* pada skenario penambahan ukuran paket

Protokol	64 bytes	128 bytes	AVG
OLSR	8,974540	48,840675	28,907607
DSDV	2,100585	0,555727	1,328156



Gambar 4.14. Grafik *delay* vs *packet size*

Berdasarkan grafik pada gambar 4.14, dapat dilihat routing protokol DSDV lebih baik dibandingkan dengan OLSR dari nilai performa rata-rata *delay* terhadap penambahan ukuran paket data. OLSR menunjukkan nilai rata-rata 28,907607 sedangkan DSDV menunjukkan nilai rata-rata 3,8272005. Hal ini disebabkan sifat DSDV yang selalu melakukan pembaruan tabel routing secara berkala. Sehingga

DSDV dapat dengan cepat menentukan jalur pengiriman, karena sudah terdapat pada tabel routing.

Melihat performa routing protokol berdasarkan nilai *QoS* terhadap skenario penambahan ukuran paket data. Diketahui bahwa routing protokol OLSR lebih baik pada nilai rata-rata *throughput*, PDR dan *packet loss* dibandingkan dengan DSDV. Sedangkan routing protokol DSDV lebih baik pada nilai rata-rata *delay* dibandingkan dengan OLSR.