PERCOBAAN 4

SIMULASI MANET (Mobile Adhoc Network)

4.1 Tujuan :

Setelah melaksanakan praktikum ini mahasiswa diharapkan mampu :

- Mendesain dan memprogram jaringan MANET menggunakan NS-3.
- Memahami dan membandingkan hasil trace output jaringan MANET pada NS-3 menggunakan gnuplot.

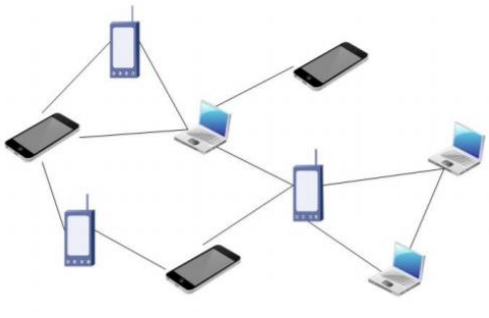
4.2 Peralatan :

- 1 PC dilengkapi dengan OS Ubuntu 18.04
- Software NS-3 versi 3.29
- Software Tracemetrics

4.3 Teori :

Mobile Ad Hoc Network (MANET) adalah sebuah jaringan tanpa kabel yang terdiri atas mobile node yang bergerak secara acak. Node-node dalamjaringan ini berfungsi juga sebagai router yang bertanggung jawab untuk mencari dan menangani rute ke setiap node didalam jaringan. Node bergerak bebas secara acak, dengan demikian topologi di jaringan mungkin dapat berubah dengan cepat dan tidak dapat diprediksi. Untuk mengatasi pergerakan ini diperlukan suatu protokol routing yang digunakan untuk menentukan rute antar node agar setiap node dapat berkomunikasi dan bertukar informasi.

Setiap node dilengkapi dengan transmitter dan receiver wireless menggunakan antena atau sejenisnya yang bersifat omnidirectional (broadcast), highly directional (point to point), memungkinkan untuk diarahkan, atau dikombinasi dari beberapa hal tersebut. Omnidirectional maksudnya adalah gelombang radio dipancarkan ke segala arah oleh perangkat transmitter wireless. Sedangkan highly directional adalah gelombang yang dipancarkan ke satu arah tertentu. Pada gambar 1 menunjukkan contoh jaringan MANET sederhana.



Gambar 1. Mobile Ad Hoc Network

4.4 Prosedur Percobaan :

1. Buka direktori *ns-allinone-3.25/ns-3.25/scratch* melalui terminal.
2. Copy-kan file manet-routing-compare.cc ke scratch

   # cp ns-allinone-3.29/ns-3.29/examples/routing/manet-routing-compare.cc ns-allinone-3.29/ns-3.29/scratch
3. Tuliskan perintah gedit dengan nama file *manet-routing-compare.cc* seperti berikut.

   gedit manet-routing-compare.cc
4. Tuliskan script di bawah ini, dimana script ini merupakan simulasi sederhana Wifi empat node.

```
#include <fstream>
#include <iostream>
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/mobility-module.h"
#include "ns3/aodv-module.h"
#include "ns3/olsr-module.h"
#include "ns3/dsdv-module.h"
#include "ns3/dsr-module.h"
#include "ns3/applications-module.h"
#include "ns3/yans-wifi-helper.h"

using namespace ns3;
using namespace dsr;

NS_LOG_COMPONENT_DEFINE ("manet-routing-compare");

class RoutingExperiment
{
public:
  RoutingExperiment ();
  void Run (int nSinks, double txp, std::string CSVfileName);
  //static void SetMACParam (ns3::NetDeviceContainer & devices,
  //                         int slotDistance);
  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;
  bool m_traceMobility;
  uint32_t m_protocol;
};

RoutingExperiment::RoutingExperiment ()
  : port (9),
    bytesTotal (0),
    packetsReceived (0),
    m_CSVfileName ("manet-routing.output.csv"),
    m_traceMobility (false),
    m_protocol (2) // AODV
{
}

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
RoutingExperiment::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));
    }
}

void
RoutingExperiment::CheckThroughput ()
```

```
{
 double kbs = (bytesTotal * 8.0) / 1000;
 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), &RoutingExperiment::CheckThroughput, this);
}

Ptr<Socket>
RoutingExperiment::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 (&RoutingExperiment::ReceivePacket, this));

 return sink;
}

std::string
RoutingExperiment::CommandSetup (int argc, char **argv)
{
 CommandLine cmd;
 cmd.AddValue ("CSVfileName", "The name of the CSV output file name", m_CSVfileName);
 cmd.AddValue ("traceMobility", "Enable mobility tracing", m_traceMobility);
 cmd.AddValue ("protocol", "1=OLSR;2=AODV;3=DSDV;4=DSR", m_protocol);
 cmd.Parse (argc, argv);
 return m_CSVfileName;
}

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

 //blank out the last output file and write the column headers
 std::ofstream out (CSVfileName.c_str ());
 out << "SimulationSecond," <<
 "ReceiveRate," <<
 "PacketsReceived," <<
 "NumberOfSinks," <<
 "RoutingProtocol," <<
 "TransmissionPower" <<
 std::endl;
 out.close ();

 int nSinks = 10;
 double txp = 7.5;
```

```
  experiment.Run (nSinks, txp, CSVfileName);
}

void
RoutingExperiment::Run (int nSinks, double txp, std::string CSVfileName)
{
  Packet::EnablePrinting ();
  m_nSinks = nSinks;
  m_txp = txp;
  m_CSVfileName = CSVfileName;

  int nWifis = 50;

  double TotalTime = 200.0;
  std::string rate ("2048bps");
  std::string phyMode ("DsssRate11Mbps");
  std::string tr_name ("manet-routing-compare");
  int nodeSpeed = 20; //in m/s
  int nodePause = 0; //in s
  m_protocolName = "protocol";

  Config::SetDefault  ("ns3::OnOffApplication::PacketSize",StringValue ("64"));
  Config::SetDefault ("ns3::OnOffApplication::DataRate",  StringValue (rate));

  //Set Non-unicastMode rate to unicast mode
  Config::SetDefault ("ns3::WifiRemoteStationManager::NonUnicastMode",StringValue
(phyMode));

  NodeContainer adhocNodes;
  adhocNodes.Create (nWifis);

  // setting up wifi phy and channel using helpers
  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 ());

  // Add a mac and disable rate control
  WifiMacHelper wifiMac;
  wifi.SetRemoteStationManager ("ns3::ConstantRateWifiManager",
                                "DataMode",StringValue (phyMode),
                                "ControlMode",StringValue (phyMode));

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

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

  MobilityHelper mobilityAdhoc;
  int64_t streamIndex = 0; // used to get consistent mobility across scenarios

  ObjectFactory pos;
  pos.SetTypeId ("ns3::RandomRectanglePositionAllocator");
  pos.Set ("X", StringValue ("ns3::UniformRandomVariable[Min=0.0|Max=300.0]"));
  pos.Set ("Y", StringValue ("ns3::UniformRandomVariable[Min=0.0|Max=1500.0]"));
```

```
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);
NS_UNUSED (streamIndex); // From this point, streamIndex is unused

AodvHelper aodv;
OlsrHelper olsr;
DsdvHelper dsdv;
DsrHelper dsr;
DsrMainHelper dsrMain;
Ipv4ListRoutingHelper list;
InternetStackHelper internet;

switch (m_protocol)
 {
 case 1:
   list.Add (olsr, 100);
   m_protocolName = "OLSR";
   break;
 case 2:
   list.Add (aodv, 100);
   m_protocolName = "AODV";
   break;
 case 3:
   list.Add (dsdv, 100);
   m_protocolName = "DSDV";
   break;
 case 4:
   m_protocolName = "DSR";
   break;
 default:
   NS_FATAL_ERROR ("No such protocol:" << m_protocol);
 }

if (m_protocol < 4)
 {
   internet.SetRoutingHelper (list);
   internet.Install (adhocNodes);
 }
else if (m_protocol == 4)
 {
   internet.Install (adhocNodes);
   dsrMain.Install (dsr, adhocNodes);
 }

NS_LOG_INFO ("assigning ip address");

Ipv4AddressHelper addressAdhoc;
addressAdhoc.SetBase ("10.1.1.0", "255.255.255.0");
```

```
  Ipv4InterfaceContainer adhocInterfaces;
  adhocInterfaces = addressAdhoc.Assign (adhocDevices);

  OnOffHelper onoff1 ("ns3::UdpSocketFactory",Address ());
  onoff1.SetAttribute ("OnTime", StringValue ("ns3::ConstantRandomVariable[Constant=1.0]"));
  onoff1.SetAttribute ("OffTime", StringValue ("ns3::ConstantRandomVariable[Constant=0.0]"));

  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 (100.0,101.0)));
     temp.Stop (Seconds (TotalTime));
   }

  std::stringstream ss;
  ss << nWifis;
  std::string nodes = ss.str ();

  std::stringstream ss2;
  ss2 << nodeSpeed;
  std::string sNodeSpeed = ss2.str ();

  std::stringstream ss3;
  ss3 << nodePause;
  std::string sNodePause = ss3.str ();

  std::stringstream ss4;
  ss4 << rate;
  std::string sRate = ss4.str ();

  //NS_LOG_INFO ("Configure Tracing.");
  //tr_name = tr_name + "_" + m_protocolName +"_" + nodes + "nodes_" + sNodeSpeed +
"speed_" + sNodePause + "pause_" + sRate + "rate";

  //AsciiTraceHelper ascii;
  //Ptr<OutputStreamWrapper> osw = ascii.CreateFileStream ( (tr_name + ".tr").c_str());
  //wifiPhy.EnableAsciiAll (osw);
  AsciiTraceHelper ascii;
  MobilityHelper::EnableAsciiAll (ascii.CreateFileStream (tr_name + ".mob"));

  //Ptr<FlowMonitor> flowmon;
  //FlowMonitorHelper flowmonHelper;
  //flowmon = flowmonHelper.InstallAll ();

  NS_LOG_INFO ("Run Simulation.");

  CheckThroughput ();

  Simulator::Stop (Seconds (TotalTime));
  Simulator::Run ();

  //flowmon->SerializeToXmlFile ((tr_name + ".flowmon").c_str(), false, false);

  Simulator::Destroy ();
}
```

5. Buka direktori *ns-allinone-3.2/ns-3.29* dan jalankan program tersebut menggunakan *waf* dengan menuliskan perintah seperti di bawah ini.

```
root@zenhadi-VirtualBox:~/ns-allinone-3.29/ns-3.29# ./waf --run scratch/manet-ro
uting-compare
Waf: Entering directory `/home/zenhadi/ns-allinone-3.29/ns-3.29/build'
[2576/2633] Compiling scratch/manet-routing-compare.cc
[2593/2633] Linking build/scratch/manet-routing-compare
Waf: Leaving directory `/home/zenhadi/ns-allinone-3.29/ns-3.29/build'
Build commands will be stored in build/compile_commands.json
'build' finished successfully (32.421s)
100.348 4 received one packet from 10.1.1.15
100.596 4 received one packet from 10.1.1.15
100.846 4 received one packet from 10.1.1.15
100.872 3 received one packet from 10.1.1.14
100.873 3 received one packet from 10.1.1.14
101.06 3 received one packet from 10.1.1.14
101.079 9 received one packet from 10.1.1.20
101.082 9 received one packet from 10.1.1.20
101.09 5 received one packet from 10.1.1.16
```

   Start simulasi untuk pengiriman data pada detik ke 100.

6. Untuk melakukan analisis, aktifkan “Flow Monitor” pada kode program tsb dengan cara menghilangkan komentar:

```
Ptr<FlowMonitor> flowmon;
FlowMonitorHelper flowmonHelper;
flowmon = flowmonHelper.InstallAll ();
```

   Tambahkan di header file:

```
#include “ns3/flow-monitor-helper.h”
```

7. Jalankan kembali seperti pada perintah no. 5.
8. Terdapat 3 file output yaitu:
   a. *manet-routing-compare.flowmon*
   b. *manet-routing-compare.mob*
   c. *manet-routing.output.csv*
9. Untuk melakukan analisi terhadap file flowmon, buatlah script berikut ini:

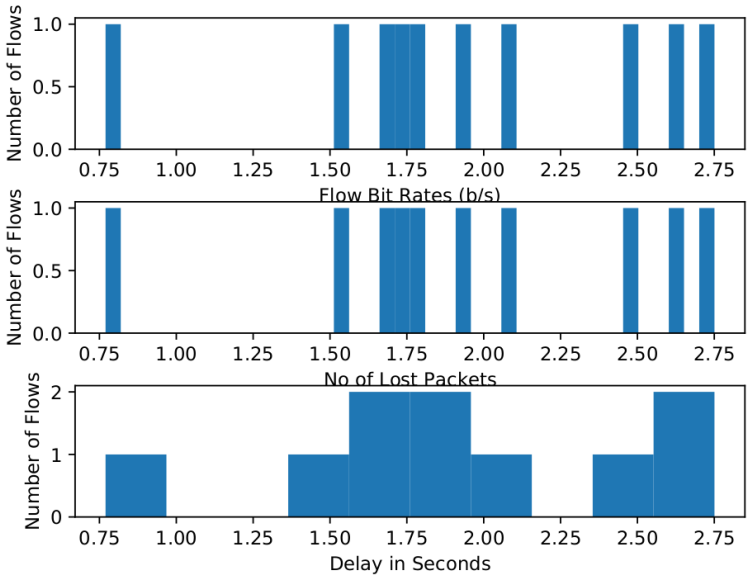
```
from xml.etree import ElementTree as ET
import sys
import matplotlib.pyplot as pylab
et=ET.parse(sys.argv[1])
bitrates=[]
losses=[]
delays=[]
for flow in et.findall("FlowStats/Flow"):
        for tpl in et.findall("Ipv4FlowClassifier/Flow"):
                if tpl.get('flowId')==flow.get('flowId'):
                        break
        if tpl.get('destinationPort')=='654':
                continue
        losses.append(int(flow.get('lostPackets')))

        rxPackets=int(flow.get('rxPackets'))
        if rxPackets==0:
                bitrates.append(0)  #no packet sent
        else:
                t0=float(flow.get('timeFirstRxPacket')[:-2])
                t1=float(flow.get("timeLastRxPacket")[:-2])
                duration=(t1-t0)*1e-9
                bitrates.append(8*long(flow.get("rxBytes"))/duration*1e-3) #into kbps
                delays.append(float(flow.get('delaySum')[:-2])*1e-9/rxPackets)

pylab.subplot(311)
pylab.hist(bitrates,bins=40) #bins = no of total binary
pylab.xlabel("Flow Bit Rates (b/s)")
pylab.ylabel("Number of Flows")

pylab.subplot(312)
pylab.hist(bitrates,bins=40)
pylab.xlabel("No of Lost Packets")
pylab.ylabel("Number of Flows")

pylab.subplot(313)
pylab.hist(bitrates,bins=10)
pylab.xlabel("Delay in Seconds")
pylab.ylabel("Number of Flows")

pylab.subplots_adjust(hspace=0.4)
pylab.savefig("results.pdf")
```

10. Simpan file tersebut: *flow.py* dan jalankan dengan python.
11. Lakukan instalasi program berikut agar bisa menjalankan script tersebut:

```
$apt install python-pip
$pip install matplotlib
$apt-get install python-tk
```

$python flow.py manet-routing-compare.flowmon

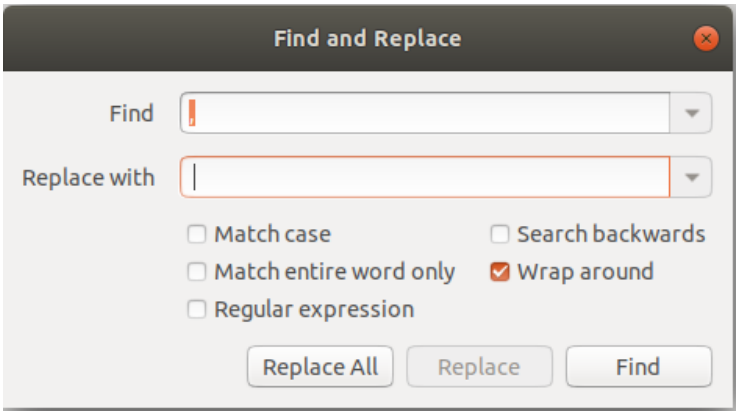
12. Dapatkan hasil seperti berikut ini:



13. Untuk melakukan analisis terhadapa file csv: *manet-routing.output.csv*

```
SimulationSecond,ReceiveRate,PacketsReceived,NumberOfSinks,RoutingProtocol,TransmissionPower
0,0,0,10,AODV,7.5
1,0,0,10,AODV,7.5
2,0,0,10,AODV,7.5
3,0,0,10,AODV,7.5
4,0,0,10,AODV,7.5
```

14. Edit file dengan menghapus baris pertama dan menghilangkan tanda ","



15. Simpan file datas menjadi: *AODV.csv*, maka hasilnya akan seperti berikut ini:

```
0 0 0 10 AODV 7.5
1 0 0 10 AODV 7.5
2 0 0 10 AODV 7.5
3 0 0 10 AODV 7.5
4 0 0 10 AODV 7.5
5 0 0 10 AODV 7.5
```

16. Buatlah script file: *gnuplot.code*

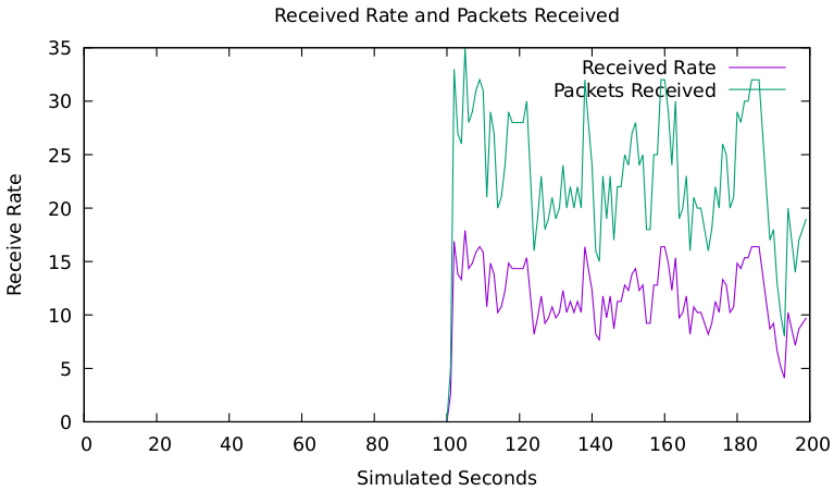
```
set terminal pdf
set output "Packetrate.pdf"
set title "Received Rate and Packets Received"
set xlabel "Simulated Seconds"
set ylabel "Receive Rate"
plot "AODV.csv" using 1:2 with lines title "Received Rate", "AODV.csv" using 1:3 with lines
title "Packets Received"
```

17. Jalankan scrip diatas, install terlebih dahulu gnuplot jika belum tersedia:

    # apt install gnuplot

    # gnuplot gnuplot.code

18. Maka hasilnya akan sebagai berikut:



4.5 Analisa

- Rubahlah dengan menggunakan protokol yang berbeda, misal OLSR
- Amatilah perubahan yang terjadi.