PERCOBAAN 5

ANALISA QoS PADA MANET (Mobile Adhoc Network)

5.1 Tujuan :

Setelah melaksanakan praktikum ini mahasiswa diharapkan mampu :

- Memahami berbagai jenis mobility model
- Memahami pergerakan node di ns-3

5.2 Peralatan :

- 1 PC dilengkapi dengan OS Ubuntu 22.04
- Software NS-3 versi 3.36.1
- Software bonnmotion

5.3 Teori :

Model mobilitas adalah salah satu parameter penting perlu diperhatikan saat ingin membangun sebuah simulasi dalam lingkungan MANET (Rohankar, 2012). Model mobilitas adalah sebuah teknik dalam simulasi yang menentukan pola pergerakan node, bagaimana letak posisi node, percepatan dan perubahan kecepatan node setiap waktu. Maka dari itu, pemilihan model mobilitas dapat menjadi pengaruh besar terhadap performa dari routing protokol. Berikut adalah model mobilitas:

1. Random Waypoint Mobility Model

   Node pada model mobilitas Random Waypoint disebarkan secara acak pada simulasi. Masing-masing node bergerak secara independen dan terpisah dari node yang lain. Pertama yang dilakukan adalah setiap node memilih target destinasi secara acak kemudian memilih kecepatan pergerakan. Kecepatan perpindahan dipilih secara acak sesuai dengan interval. Setelah sampai di destinasi, node kemudian berhenti untuk waktu yang sudah ditentukan (pause time) sebelum kemudian mulai melakukan prosedur yang sama lagi.

2. Random Walk Mobility Model

   Node pada model mobilitas Random Walk bergerak dari lokasi awal menuju lokasi yang baru dengan menentukan kecepatan dan arah yang akan dituju secara acak.

Selain kecepatan yang sudah ditentukan dengan interval, arah destinasi juga sudah ditentukan dengan interval. Destinasi baru akan ditentukan setelah node telah bergerak sesuai waktu yang sudah ditentukan atau jarak yang sudah dilalui. Secara sederhana, model mobilitas Random Walk adalah model mobilitas Random Waypoint tanpa adanya pause time.

3. Random Direction Mobility Model

   Model mobilitas Random Direction dikembangkan dengan tujuan untuk mengurangi masalah density wave yang terjadi pada model mobilitas Random Waypoint. Density wave adalah penumpukan node yang terjadi pada satu sisi field simulasi. Pada Random Direction, node memilih arah pergerakan secara acak kemudian bergerak menuju arah tersebut. Berbeda dengan Random Waypoint, node baru akan berhenti dengan waktu yang telah ditentukan (pause time) saat mencapai batas field simulasi. Setelah itu, node akan memilih arah baru dan bergerak sehingga node dapat tersebar secara merata.

5.4 Prosedur Percobaan

1. Pada simulasi MANET terdapat 3 file output yaitu:
   a. *manet-routing-compare.flowmon*
   b. *manet-routing-compare.mob*
   c. *manet-routing.output.csv*

   Letak 3 file tersebut pada folder:
   /home/zenhadi/Downloads/ns-allinone-3.36.1/ns-3.36.1

2. Analisa QoS yaitu throughput dengan file python dari file: *manet-routing.output.csv*

manet-routing.output.csv

```
SimulationSecond,ReceiveRat
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
6,0,0,10,AODV,7.5
7,0,0,10,AODV,7.5
8,0,0,10,AODV,7.5
9,0,0,10,AODV,7.5
10,0,0,10,AODV,7.5
11,0,0,10,AODV,7.5
12,0,0,10,AODV,7.5
13,0,0,10,AODV,7.5
14,0,0,10,AODV,7.5
15,0,0,10,AODV,7.5
16,0,0,10,AODV,7.5
17,0,0,10,AODV,7.5
18,0,0,10,AODV,7.5
```

python3 bacaCSV.py

```
import csv
with open('manet-routing.output.csv') as csv_file:
    csv_reader = csv.reader(csv_file, delimiter=',')
    line_count = 0
    startTime = 100
    dtTh = 0.0
    a = 0
    for row in csv_reader:
        if line_count == 0:
            #print column names
            print("Column name:%s" %row)
            line_count += 1
        else:
            #Convert the first column to float for comparison
            current_time = float(row[0])
            if current_time > startTime:
                        #Print the desired columns
                print(row[0] +"\t"+ row[1] +"\t"+ row[2])
                dtTh = dtTh + float(row[1]) #Accumulate the receive rate
                a = a + 1
                line_count += 1
    print("Proses selesai")
    print("Throughput total = %f " %dtTh)
    print("total data = %i " %a)
```

```
print("Throughput rata-rata= %f kbps" %(dtTh/a))
```

Hasil yang didapat:

```
197     12.288  24
198     10.24   20
199     7.68    15
Proses selesai
Throughput total = 744.448000
total data = 99
Throughput rata-rata= 7.519677 kbps
```

3. Analisa QoS yaitu delay, packet loss, PDR (packet delivery ratio) pada file: *manet-routing-compare.flowmon*

manet-routing-compare.flowmon | manet-routing-compare.cc | manet-routing.output.csv

```
<?xml version="1.0" ?>
<FlowMonitor>
  <FlowStats>
    <Flow flowId="1" timeFirstTxPacket="+1.00346e+11ns"
 timeFirstRxPacket="+1.17651e+11ns" timeLastTxPacket="+1.99846e+11ns"
 timeLastRxPacket="+1.66597e+11ns" delaySum="+4.47229e+09ns" jitterSum="+4.70549e+09ns"
 lastDelay="+336454ns" txBytes="36708" rxBytes="6716" txPackets="399" rxPackets="73"
 lostPackets="286" timesForwarded="154">
      <packetsDropped reasonCode="0" number="1" />
      <packetsDropped reasonCode="1" number="0" />
      <packetsDropped reasonCode="2" number="0" />
      <packetsDropped reasonCode="3" number="0" />
      <packetsDropped reasonCode="4" number="0" />
      <packetsDropped reasonCode="5" number="0" />
      <packetsDropped reasonCode="6" number="75" />
      <bytesDropped reasonCode="0" bytes="92" />
      <bytesDropped reasonCode="1" bytes="0" />
      <bytesDropped reasonCode="2" bytes="0" />
      <bytesDropped reasonCode="3" bytes="0" />
      <bytesDropped reasonCode="4" bytes="0" />
      <bytesDropped reasonCode="5" bytes="0" />
      <bytesDropped reasonCode="6" bytes="6900" />
    </Flow>
```

a. Copykan file flowmon-parse-results.py
dari folder: ~/ ns-allinone-3.36.1/ns-3.36.1/src/flow-monitor/examples
ke folder: ~ns-allinone-3.36.1/ns-3.36.1

cp /home/zenhadi/Downloads/ns-allinone-3.36.1/ns-3.36.1/src/flow-monitor/examples/flowmon-parse-results.py /home/zenhadi/Downloads/ns-allinone-3.36.1/ns-3.36.1

b. Amati dan pelajari file: flowmon-parse-results.py

```
# gedit  flowmon-parse-results.py

from __future__ import division
import sys
import os
try:
   from xml.etree import cElementTree as ElementTree
except ImportError:
   from xml.etree import ElementTree

def parse_time_ns(tm):
   if tm.endswith('ns'):
      return long(tm[:-4])
   raise ValueError(tm)

## FiveTuple
class FiveTuple(object):
   ## class variables
   ## @var sourceAddress
   #  source address
   ## @var destinationAddress
   #  destination address
   ## @var protocol
   #  network protocol
   ## @var sourcePort
   #  source port
   ## @var destinationPort
   #  destination port
   ## @var __slots__
   #  class variable list
   __slots__ = ['sourceAddress', 'destinationAddress', 'protocol', 'sourcePort', 'destinationPort']
   def __init__(self, el):
      '''The initializer.
      @param self The object pointer.
      @param el The element.
      '''
      self.sourceAddress = el.get('sourceAddress')
      self.destinationAddress = el.get('destinationAddress')
      self.sourcePort = int(el.get('sourcePort'))
      self.destinationPort = int(el.get('destinationPort'))
      self.protocol = int(el.get('protocol'))

## Histogram
class Histogram(object):
   ## class variables
   ## @var bins
   #  histogram bins
   ## @var nbins
```

```
    #  number of bins
    ## @var number_of_flows
    #  number of flows
    ## @var __slots__
    #  class variable list
    __slots__ = 'bins', 'nbins', 'number_of_flows'
    def __init__(self, el=None):
        ''' The initializer.
        @param self The object pointer.
        @param el The element.
        '''
        self.bins = []
        if el is not None:
            #self.nbins = int(el.get('nBins'))
            for bin in el.findall('bin'):
                self.bins.append( (float(bin.get("start")), float(bin.get("width")), int(bin.get("count"))) )

## Flow
class Flow(object):
    ## class variables
    ## @var flowId
    #  delay ID
    ## @var delayMean
    #  mean delay
    ## @var packetLossRatio
    #  packet loss ratio
    ## @var rxBitrate
    #  receive bit rate
    ## @var txBitrate
    #  transmit bit rate
    ## @var fiveTuple
    #  five tuple
    ## @var packetSizeMean
    #  packet size mean
    ## @var probe_stats_unsorted
    #  unsirted probe stats
    ## @var hopCount
    #  hop count
    ## @var flowInterruptionsHistogram
    #  flow histogram
    ## @var rx_duration
    #  receive duration
    ## @var __slots__
    #  class variable list
    __slots__ = ['flowId', 'delayMean', 'packetLossRatio', 'rxBitrate', 'txBitrate',
                 'fiveTuple', 'packetSizeMean', 'probe_stats_unsorted',
                 'hopCount', 'flowInterruptionsHistogram', 'rx_duration']
    def __init__(self, flow_el):
```

```
        ''' The initializer.
        @param self The object pointer.
        @param flow_el The element.
        '''
        self.flowId = int(flow_el.get('flowId'))
        rxPackets = long(flow_el.get('rxPackets'))
        txPackets = long(flow_el.get('txPackets'))
        tx_duration = float(long(flow_el.get('timeLastTxPacket')[:-4]) -
long(flow_el.get('timeFirstTxPacket')[:-4]))*1e-9
        rx_duration = float(long(flow_el.get('timeLastRxPacket')[:-4]) -
long(flow_el.get('timeFirstRxPacket')[:-4]))*1e-9
        self.rx_duration = rx_duration
        self.probe_stats_unsorted = []
        if rxPackets:
            self.hopCount = float(flow_el.get('timesForwarded')) / rxPackets + 1
        else:
            self.hopCount = -1000
        if rxPackets:
            self.delayMean = float(flow_el.get('delaySum')[:-4]) / rxPackets * 1e-9
            self.packetSizeMean = float(flow_el.get('rxBytes')) / rxPackets
        else:
            self.delayMean = None
            self.packetSizeMean = None
        if rx_duration > 0:
            self.rxBitrate = long(flow_el.get('rxBytes'))*8 / rx_duration
        else:
            self.rxBitrate = None
        if tx_duration > 0:
            self.txBitrate = long(flow_el.get('txBytes'))*8 / tx_duration
        else:
            self.txBitrate = None
        lost = float(flow_el.get('lostPackets'))
        print "rxBytes: %s; txPackets: %s; rxPackets: %s; lostPackets: %s" % (flow_el.get('rxBytes'),
txPackets, rxPackets, lost)
        if rxPackets == 0:
            self.packetLossRatio = None
        else:
            self.packetLossRatio = (lost / (rxPackets + lost))

        interrupt_hist_elem = flow_el.find("flowInterruptionsHistogram")
        if interrupt_hist_elem is None:
            self.flowInterruptionsHistogram = None
        else:
            self.flowInterruptionsHistogram = Histogram(interrupt_hist_elem)

## ProbeFlowStats
class ProbeFlowStats(object):
    ## class variables
```

```
    ## @var probeId
    #  probe ID
    ## @var packets
    #  network packets
    ## @var bytes
    #  bytes
    ## @var delayFromFirstProbe
    #  delay from first probe
    ## @var __slots__
    #  class variable list
    __slots__ = ['probeId', 'packets', 'bytes', 'delayFromFirstProbe']

## Simulation
class Simulation(object):
    ## class variables
    ## @var flows
    #  list of flows
    def __init__(self, simulation_el):
        ''' The initializer.
        @param self The object pointer.
        @param simulation_el The element.
        '''
        self.flows = []
        FlowClassifier_el, = simulation_el.findall("Ipv4FlowClassifier")
        flow_map = {}
        for flow_el in simulation_el.findall("FlowStats/Flow"):
            flow = Flow(flow_el)
            flow_map[flow.flowId] = flow
            self.flows.append(flow)
        for flow_cls in FlowClassifier_el.findall("Flow"):
            flowId = int(flow_cls.get('flowId'))
            flow_map[flowId].fiveTuple = FiveTuple(flow_cls)

        for probe_elem in simulation_el.findall("FlowProbes/FlowProbe"):
            probeId = int(probe_elem.get('index'))
            for stats in probe_elem.findall("FlowStats"):
                flowId = int(stats.get('flowId'))
                s = ProbeFlowStats()
                s.packets = int(stats.get('packets'))
                s.bytes = long(stats.get('bytes'))
                s.probeId = probeId
                if s.packets > 0:
                    s.delayFromFirstProbe =  parse_time_ns(stats.get('delayFromFirstProbeSum')) /
float(s.packets)
                else:
                    s.delayFromFirstProbe = 0
                flow_map[flowId].probe_stats_unsorted.append(s)
```

```
def main(argv):
    file_obj = open(argv[1])
    print "Reading XML file ",

    sys.stdout.flush()
    level = 0
    sim_list = []
    for event, elem in ElementTree.iterparse(file_obj, events=("start", "end")):
        if event == "start":
            level += 1
        if event == "end":
            level -= 1
            if level == 0 and elem.tag == 'FlowMonitor':
                sim = Simulation(elem)
                sim_list.append(sim)
                elem.clear() # won't need this any more
                sys.stdout.write(".")
                sys.stdout.flush()
    print " done."

    a = 0  #jumlah paket
    b = 0.0 #rata-rata delay
    c = 0.0 #packet loss rate
    for sim in sim_list:
        for flow in sim.flows:
            t = flow.fiveTuple
            proto = {6: 'TCP', 17: 'UDP'} [t.protocol]
            print "FlowID: %i (%s %s/%s --> %s/%i)" % \
                (flow.flowId, proto, t.sourceAddress, t.sourcePort, t.destinationAddress,
t.destinationPort)
            if flow.txBitrate is None:
                print "\tTX bitrate: None"
            else:
                print "\tTX bitrate: %.2f kbit/s" % (flow.txBitrate*1e-3,)
            if flow.rxBitrate is None:
                print "\tRX bitrate: None"
            else:
                print "\tRX bitrate: %.2f kbit/s" % (flow.rxBitrate*1e-3,)
            if flow.delayMean is None:
                print "\tMean Delay: None"
            else:
                print "\tMean Delay: %.2f ms" % (flow.delayMean*1e3,)
                        a = a + 1
                        b = b + flow.delayMean
            if flow.packetLossRatio is None:
                print "\tPacket Loss Ratio: None"
            else:
                print "\tPacket Loss Ratio: %.2f %%" % (flow.packetLossRatio*100)
```

```
                c = c + flow.packetLossRatio*100

        print "Mean Delay Total: %.2f ms" %(b*1e3/a)
        print "Packet loss rate: %.2f %%" %(c/a)
        print "PDR: %.2f %%" %(100- c/a)
        print "Jumlah flowid: %i " % a

if __name__ == '__main__':
   main(sys.argv)
```

c. Jalankan file tersebut untuk mendapatkan data dari file manet-routing-compare.flowmon

```
# python3  flowmon-parse-results.py  manet-routing-compare.flowmon
FlowID: 903 (UDP 10.1.1.27/654 --> 10.1.1.42/654)
        TX bitrate: 128.00 kbit/s
        RX bitrate: 768.00 kbit/s
        Mean Delay: 6.78 ms
        Packet Loss Ratio: 0.00 %
FlowID: 904 (UDP 10.1.1.28/654 --> 10.1.1.49/654)
        TX bitrate: 45.33 kbit/s
        RX bitrate: 45.33 kbit/s
        Mean Delay: 0.78 ms
        Packet Loss Ratio: 0.00 %
FlowID: 905 (UDP 10.1.1.49/654 --> 10.1.1.13/654)
        TX bitrate: 62.32 kbit/s
        RX bitrate: 87.70 kbit/s
        Mean Delay: 4.24 ms
        Packet Loss Ratio: 0.00 %
```

4. Analisa rata-rata rxBytes, lostPackets, Packet Loss Ratio, Throughput, dan Delay

   Buat file analisa.py sebagai berikut dan jalankan

   # python3 analisa.py

```
import xml.etree.ElementTree as ET
import csv
# Ganti dengan nama file XML
filename = "manet-routing-compare.flowmon"
# Parsing XML
tree = ET.parse(filename)
root = tree.getroot()
```

```
# Variabel akumulasi
total_rx_bytes = 0
total_tx_packets = 0
total_lost_packets = 0
total_rx_packets = 0
flow_count = 0
total_delay_sum = 0
total_duration = 0

# Untuk CSV output
csv_rows = []

for flowStats in root.findall("FlowStats"):
    for flow in flowStats.findall("Flow"):
        try:
            rx_bytes = int(flow.attrib.get("rxBytes", 0))
            tx_packets = int(flow.attrib.get("txPackets", 0))
            rx_packets = int(flow.attrib.get("rxPackets", 0))
            lost_packets = int(flow.attrib.get("lostPackets", 0))
            delay_sum_ns = float(flow.attrib.get("delaySum", 0).replace("+", "").replace("ns", ""))
            time_first_tx = float(flow.attrib.get("timeFirstTxPacket", 0).replace("+", "").replace("ns",
"""))
            time_last_rx = float(flow.attrib.get("timeLastRxPacket", 0).replace("+", "").replace("ns",
"""))
            flow_id = flow.attrib.get("flowId", "unknown")

            duration_ns = max(time_last_rx - time_first_tx, 1)  # Hindari 0
            duration_s = duration_ns * 1e-9  # Konversi ke detik

            # Hitung metrics
            packet_loss_ratio = lost_packets / (rx_packets + lost_packets) if (rx_packets +
lost_packets) > 0 else 0
```

```
            throughput_kbps = (8 * rx_bytes / 1000) / duration_s  # dalam kbps
            avg_delay_ms = (delay_sum_ns / rx_packets) * 1e-6 if rx_packets > 0 else 0  # ms

            #Akumulasi
            flow_count += 1
            total_rx_bytes += rx_bytes
            total_tx_packets += tx_packets
            total_rx_packets += rx_packets
            total_lost_packets += lost_packets
            total_delay_sum += delay_sum_ns
            total_duration += duration_s

            packet_loss_ratio = (
               lost_packets / (rx_packets + lost_packets)
               if (rx_packets + lost_packets) > 0 else 0
            )

            #Simpan tiap flow ke csv
            csv_rows.append([
               flow_id, rx_bytes, tx_packets, rx_packets,
               lost_packets, round(packet_loss_ratio, 4),
               round(throughput_kbps, 2),
               round(avg_delay_ms, 3)
            ])

         except Exception as e:
            print(f"Error parsing flow: {e}")

      # Rata-rata
      avg_rx_bytes = total_rx_bytes / flow_count if flow_count > 0 else 0
      avg_lost_packets = total_lost_packets / flow_count if flow_count > 0 else 0
      avg_packet_loss_ratio = (
```

```
    total_lost_packets / (total_rx_packets + total_lost_packets)
    if (total_rx_packets + total_lost_packets) > 0 else 0
)
avg_throughput_kbps = (8 * total_rx_bytes / 1000) / total_duration if total_duration > 0 else 0
avg_delay_ms = (total_delay_sum / total_rx_packets) * 1e-6 if total_rx_packets > 0 else 0

print(f"Rata-rata rxBytes: {avg_rx_bytes:.2f}")
print(f"Rata-rata lostPackets: {avg_lost_packets:.2f}")
print(f"Rata-rata Packet Loss Ratio: {avg_packet_loss_ratio:.4f}")
print(f"Rata-rata Throughput     : {avg_throughput_kbps:.2f} kbps")
print(f"Rata-rata Delay          : {avg_delay_ms:.3f} ms")

# Simpan ke CSV
with open("flow_results.csv", "w", newline="") as csvfile:
    writer = csv.writer(csvfile)
    writer.writerow(["FlowID", "rxBytes", "txPackets", "rxPackets", "lostPackets",
"PacketLossRatio", "Throughput(kbps)", "AvgDelay(ms)"])
    writer.writerows(csv_rows)

print("Hasil disimpan di: flow_results.csv")
```

Hasil yang didapat:

```
Rata-rata rxBytes: 453.16
Rata-rata lostPackets: 3.06
Rata-rata Packet Loss Ratio: 0.2704
Rata-rata Throughput     : 0.19 kbps
Rata-rata Delay          : 78.296 ms
Hasil disimpan di: flow_results.csv
```

File flow_results.csv

```
1 FlowID,rxBytes,txPackets,rxPackets,lostPackets,PacketLossRatio,Throughput(kbps),AvgDelay(ms)
2 1,6716,399,73,286,0.7967,0.81,61.264
3 2,11960,399,130,257,0.6641,1.68,52.213
4 3,7636,398,83,293,0.7793,0.63,321.924
5 4,15364,398,167,224,0.5729,1.24,319.563
6 5,92,398,1,357,0.9972,25.38,28.743
7 6,48,1,1,0,0.0,27.43,13.973
8 7,144,4,3,1,0.25,3.15,2.597
9 8,48,1,1,0,0.0,34.91,11.551
10 9,30,1,1,0,0.0,30.0,8.534
```

5. Bandingkan hasilnya bila menggunakan protokol routing OLSR untuk:

   Rata-rata rxBytes

   Rata-rata lostPackets

   Rata-rata Packet Loss Ratio

   Rata-rata Throughput

   Rata-rata Delay

   Buat grafik batang untuk perbandingan AODV dan OLSR.