PERCOBAAN 5

ANALISA QoS PADA MANET (Mobile Adhoc Network)

6.1 Tujuan :

Setelah melaksanakan praktikum ini mahasiswa diharapkan mampu :

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

6.2 Peralatan :

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

6.3 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.

6.4 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*

2. Analisa QoS yaitu throughput dengan file python

   # python 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:
        #startTime = row[0]
        #print ("StartTime: %s " %startTime)
   if line_count == 0:
     #print(f'Column names are {", ".join(row)}')
         print("Column name: %s " %row)
```

```
            line_count += 1
        elif row[0] > startTime:
            print(row[0] +"\t"+ row[1] +"\t"+ row[2])
                dtTh = dtTh + float(row[1])
                a = a + 1
            line_count += 1
    print("Proses selesai")
    print "Throughput total = %f " %dtTh
    print "total data = %i " %a
    print "Throughput = %f kbps" %(dtTh/a)
```

3. Analisa QoS yaitu delay, packet loss, PDR (packet delivery ratio)

```
# python 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)
```