**Bangladesh University of Engineering and Technology**
**Department of Computer Science and Engineering**
**CSE322: Computer Networks Sessional**

# *Distributed Cache Update Scheme in Dynamic Source Routing Protocol*

***An approach of improving QoS of mobile dd-hoc network***

Supervised by:
**Md. Tareq Mahmood (Lecturer,CSE BUET)**
Submitted by:
**Saikat Ghatak**
Student ID: 1705116

Date of submission : 22 Feb, 2022

# Contents


**1 Network topologies under simulation** **2**

1.1 TaskA . . . 2

1.1.1 Wireless high-rate (802.11 static) . . . 2

1.1.2 Wireless low-rate (802.15.4 static) . . . 2

1.2 TaskB . . . 3

1.2.1 Modification on DSR using MANET . . . 3

**2 Parameters under variation** **4**

**3 Overview of the proposed algorithm** **5**

**4 Modifications made in the simulator** **6**

4.1 Conventional DSR Codeflow . . . 6

4.2 Implementation of Proposed Algorithm on Conventional DSR in Ns3 . . . 7

4.3 Challenges for Calculation of Performance Metrics in DSR in Ns3 . . . 8

4.3.1 !!! Warning !!! : Read it Carefully or Waste more Time . . . 8

4.3.2 Why does not FlowMonitor collect data for dsr.cc in ns3? . . . 8

4.3.3 Alternative way for collect data for dsr in ns3? . . . 8

4.4 Short Approach for Calculation of Performance Metrics(Throughput Packet Received)in DSR in Ns3 . . . 9

**5 Results with graphs** **10**

5.1 Varying Parameters Without Modifications(Task A) . . . 10

5.1.1 Graphs for 802.11 static . . . 10

5.1.2 Graphs for 802.15.4 static . . . 26

5.2 Graphs for Simulation with Proposed Modification (Task B) . . . 42

5.2.1 Throughput . . . 42

5.2.2 Received Packet . . . 43

**6 Summary findings explaining the results** **44**

6.1 TaskA Summary . . . 44

6.1.1 Result Analysis for Wireless high-rate (802.11 static) . . . 44

6.1.2 Result Analysis for Wireless low-rate (802.15.4 static) . . . 45

6.2 Comparison Summary of Task$_A$ . . . 46

6.3 TaskB Summary . . . 47

6.3.1 Receive Packet Analysis . . . 47

6.3.2 Throughput Analysis . . . 47

# 1 Network topologies under simulation

## 1.1 TaskA

### 1.1.1 Wireless high-rate (802.11 static)

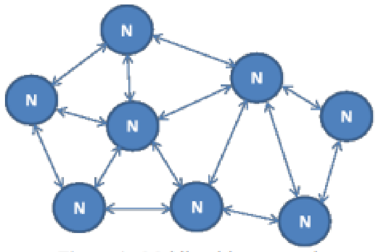
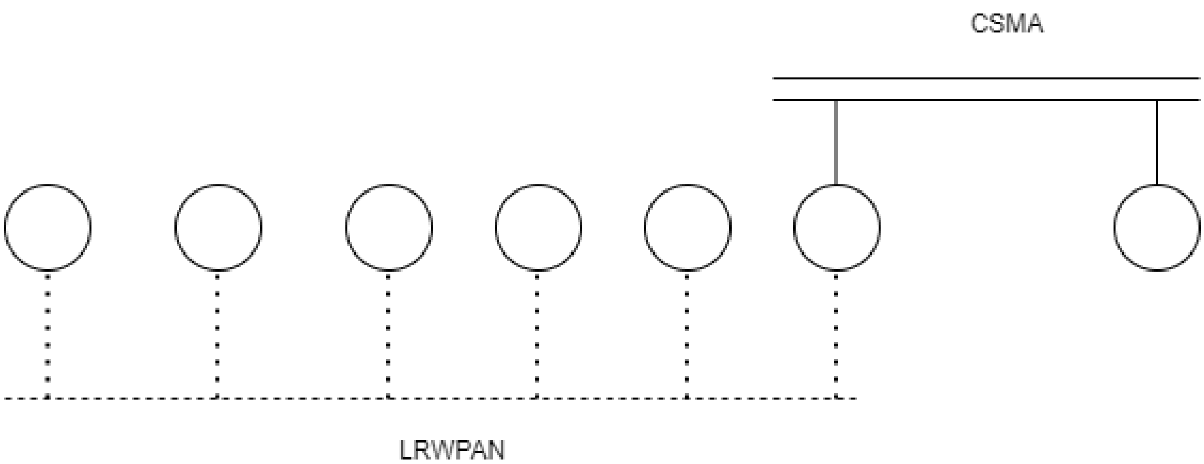



Figure 1: Mobile ad-hoc network

- **Topology Parameters :**
- Propagation Model: ConstantSpeedPropagationDelay
- Propagation Loss Model: RangePropagationLossModel
- Position Allocator: ListPositionAllocator
- MobilityModel: ConstantPositionMobilityModel
- Mac: AdhocWifiMAC
- Mac Standard: 802.11B
- Protocol: AODV

### 1.1.2 Wireless low-rate (802.15.4 static)



- **Topology Parameters :**
- Propagation Model: ConstantSpeedPropagationDelay
- Propagation Loss Model: RangePropagationLossModel
- Position Allocator: GridPositionAllocator
- MobilityModel: ConstantPositionMobilityModel
- LrWpanHelper
- SixLowPanHelper

## 1.2 TaskB

### 1.2.1 Modification on DSR using MANET

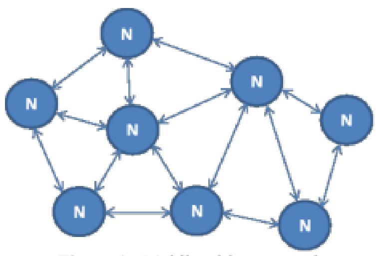



**Figure 1:** Mobile ad-hoc network

- **Topology Parameters :**
- Propagation Model: ConstantSpeedPropagationDelay
- Propagation Loss Model: FriisPropagationLossModel
- Position Allocator: RandomRectanglePositionAllocator
- MobilityModel: RandomWaypointMobilityModel
- Mac: AdhocWifiMAC
- Mac Standard: 802.11B
- Protocol: DSR

## 2 Parameters under variation

**Here topology was built based on Wireless high-rate (802.11 static) and Wireless low-rate (802.15.4 static) standard. Topology was simulated under variation to the parameters :**

- The number of nodes needs to be varied as 20, 40, 60, 80, and 100
- The number of flows (10, 20, 30, 40, and 50)
- The number of packets per second (100, 200, 300, 400, and 500)
- Coverage area (square coverage are varying one side as Txrange, 2 x Txrange, 3 x Txrange, 4 x Txrange, and 5 x Txrange)

## 3 Overview of the proposed algorithm

- When broken link information is detected RERR message is propagated and Route Cache is updated only the nodes involved in the routing path .
- Where New protocol UDSR updates the cache using distributed route cache update algorithm in distributed manner for all the nodes present in the network topology.
- The nodes involve in the path receives RERR message but the nodes not present in the routing path could not hears RERR message.
- Hence using new approach explicit RERR notification is made and send to all reachable nodes in the topology and update the Route Cache distributed manner
- . Such approach improves the QoS of network using different parameters.

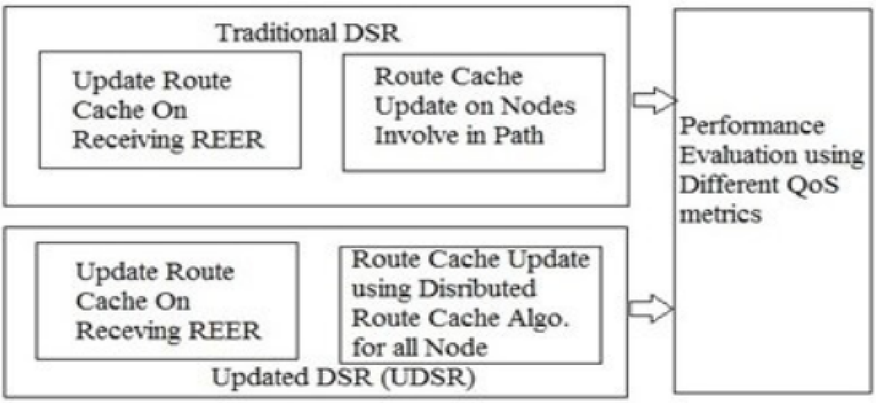



Fig.1: Block Diagram of Proposed System

# 4 Modifications made in the simulator

## 4.1 Conventional DSR Codeflow

```
ns-allinone-3.35 > ns-3.35 > src > dsr > model > dsr-routing.cc > {} ns3 > {} dsr > Receive(Ptr<Packet>, Ipv4Header const &, Ptr<Ipv4Interface>)
    else if (optionType == 3)       // This is a route error header
      {
        // populate this route error
        NS_LOG_INFO ("The option type value " << (uint32_t)optionType);

        dsrOption = GetOption (optionType);
        optionLength = dsrOption->Process (p, packet, m_mainAddress, source, ip, protocol, isPromisc, promiscSource);

        if (optionLength == 0)
          {
            NS_LOG_INFO ("Discard this packet");
            m_dropTrace (p);
          }
        NS_LOG_INFO ("The option Length " << (uint32_t)optionLength);
```

Figure 1: Receive() method call Process() method in ”dsr-routing.cc” during REER message

```
ns-allinone-3.35 > ns-3.35 > src > dsr > model > dsr-options.cc > {} ns3 > {} dsr > Process(Ptr<Packet>, Ptr<Packet>, Ipv4Address, Ipv4Address, Ipv4Header const &, ui

          dsr->DeleteAllRoutesIncludeLink (errorSource, unreachAddress,ipv4Address);
          Ptr<Packet> newP = p->Copy ();
          uint32_t serialized = DoSendError (newP, rerrUnreach, rerrSize, ipv4Address , protocol);
```

Figure 2: Process() method call DeleteAllRoutesIncludeLink() and DoSendErr() method for route cache update it’s neighbour node in ”dsr-options.cc”

```
ns-allinone-3.35 > ns-3.35 > src > dsr > model > dsr-options.cc > {} ns3 > {} dsr > DoSendError(Ptr<Packet>, DsrOptionRerrUnreachHeader &, uint32_t, Ipv4Address, uint8_
  // Set the route entry
  SetRoute (nextAddress, ipv4Address);
  dsr->ForwardErrPacket (rerr, newSourceRoute, nextAddress, protocol, m_ipv4Route);
  return serializedSize;
}
```

Figure 3: DoSendErr() method call ForwardErrPacket() method for forwarding RERR packet in ”dsr-options.cc”

## 4.2 Implementation of Proposed Algorithm on Conventional DSR in Ns3

```
MDSR.cc    dsr-options.cc ×    dsr-routing.cc    dsr-rcache.cc    dsr-option-header.cc
ns-allinone-3.35 > ns-3.35 > src > dsr > model > dsr-options.cc > {} ns3 > {} dsr > Process(Ptr<Packet>, Ptr<Packet>, Ipv4Address, Ipv4Address, Ipv4

          /*
           * Delete all the routes including the unreachable node address from the route cache
           */
          Ptr<Node> node = GetNodeWithAddress (ipv4Address);

// ####################################################################################
                                      // ****** 1st Modified Start
//####################################################################################
          string s1= "10.1.1.";
          for(int node_index =1 ; node_index<nNode; node_index++)
          {
            string s2 =to_string(node_index);
            string s=s1+s2;
            int v = s.length() ;
            char str[v+1];
            for(int i=0; i<v; i++)
            {
                str[i]=s[i];
            }
            str[v]='\0';

            Ipv4Address eachNodeAddress =str;
            dsr->DeleteAllRoutesIncludeLink (eachNodeAddress, unreachAddress,ipv4Address);
             dsr->DeleteAllRoutesIncludeLink (errorSource, unreachAddress,eachNodeAddress);
          }
// ####################################################################################
                                      // ****** 1st Modified End
//####################################################################################


          dsr->DeleteAllRoutesIncludeLink (errorSource, unreachAddress,ipv4Address);
          Ptr<Packet> newP = p->Copy ();
          uint32_t serialized = DoSendError (newP, rerrUnreach, rerrSize, ipv4Address , protocol);
```

(a) RERR Message Broadcast to All Node

```
ns-allinone-3.35 > ns-3.35 > src > dsr > model > dsr-options.cc > {} ns3 > {} dsr > Process(Ptr<Packet>, Ptr<Packet>, Ipv4Address, Ipv4Address, Ipv4Hea


          dsr->DeleteAllRoutesIncludeLink (errorSrc, unreachNode, ipv4Address);
// ####################################################################################
                                      // ****** 2nd Modified Start
//####################################################################################
           string s1= "10.1.1.";
          for(int node_index =1 ; node_index<nNode; node_index++)
          {
            string s2 =to_string(node_index);
            string s=s1+s2;
            int v = s.length() ;
            char str[v+1];
            for(int i=0; i<v; i++)
            {
                str[i]=s[i];
            }
            str[v]='\0';

            Ipv4Address eachNodeAddress =str;
            dsr->DeleteAllRoutesIncludeLink (eachNodeAddress, unreachNode,ipv4Address);
             dsr->DeleteAllRoutesIncludeLink (errorSrc, unreachNode,eachNodeAddress);

          }

// ####################################################################################
                                      // ****** 2nd Modified End
// ####################################################################################
```

(b) Anothe RERR Message Broadcast to All Node

Figure 4: Modifications in dsr-options.cc class using Distributed Algorithm

## 4.3 Challenges for Calculation of Performance Metrics in DSR in Ns3

### 4.3.1 !!! Warning !!! : Read it Carefully or Waste more Time

### 4.3.2 Why does not FlowMonitor collect data for dsr.cc in ns3?

- FlowMonitor requires either UDP or TCP. DSR adds a DSR header to IP packets between IP and L4 (TCP and UDP). As a consequence, FlowMonitor can not recognize the L4 header and can not collect stats.

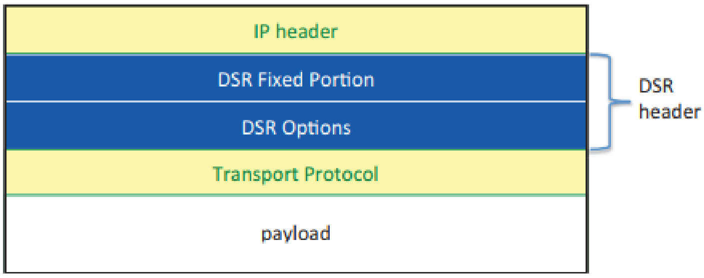



**Figure 2: DSR header encapsulation within IP**

- As a result , we can not collect data using flow monitor in dsr .

### 4.3.3 Alternative way for collect data for dsr in ns3?

- Using Statistics Module in ns3
- Using TraceMetrics (Need more analysis time )

## 4.4 Short Approach for Calculation of Performance Metrics(Throughput Packet Received)in DSR in Ns3

```
ns-allinone-3.35 > ns-3.35 > scratch > MDSR.cc > CheckThroughput()
}

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;

```

Figure 5: Total Throughput Calculation

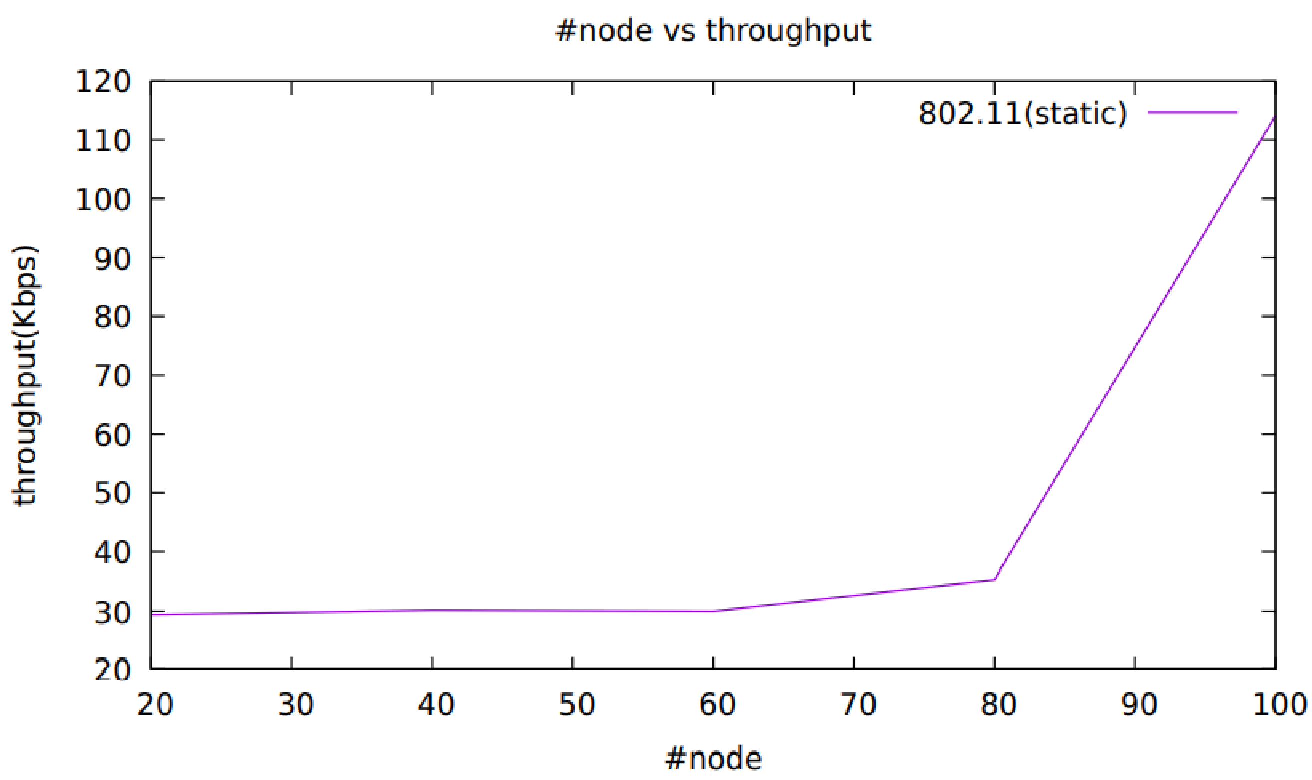
```
std::cout << "Throughput : " <<(m_tp /m_time_diff)<< "  Loading...
std::cout << "Total Packet Received : " <<TotalPacketReceived<< "\n";

```

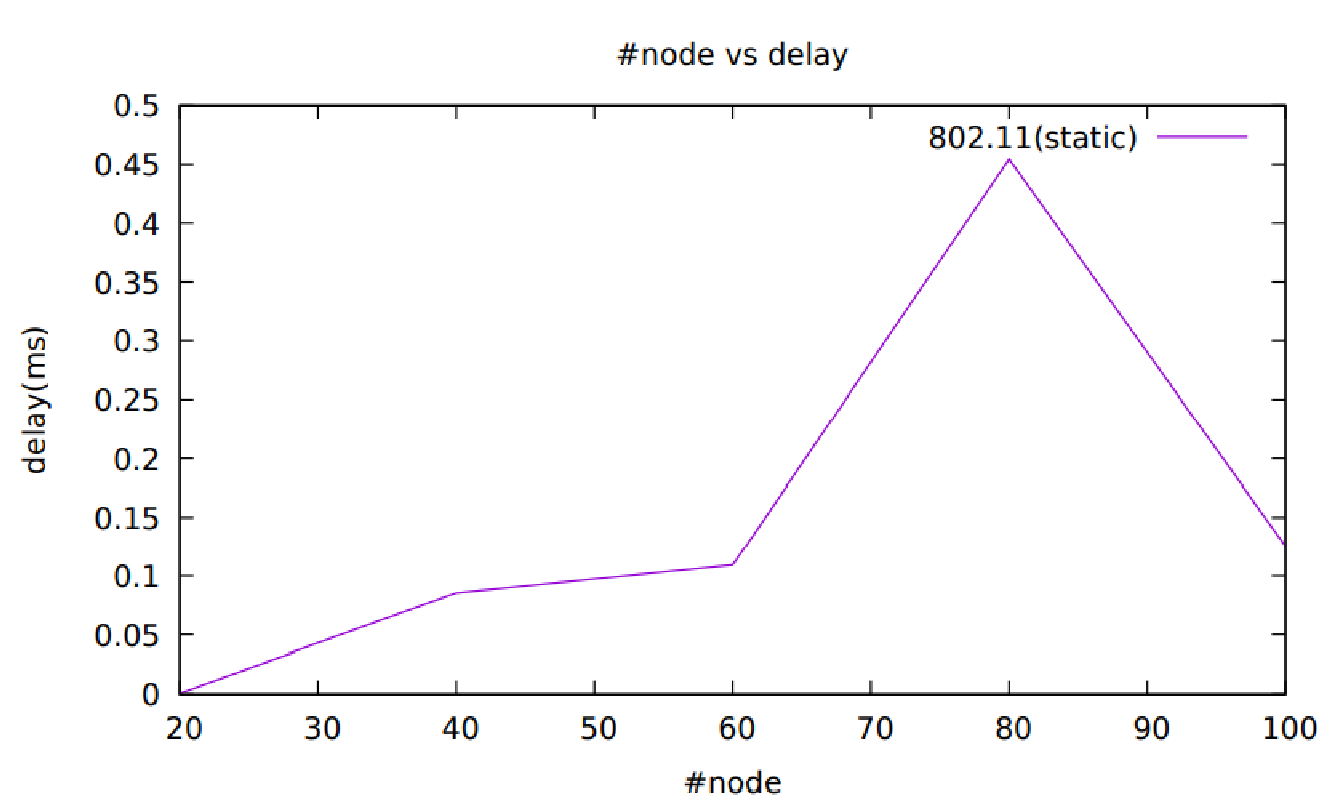
Figure 6: Average Througput Calculation and Print

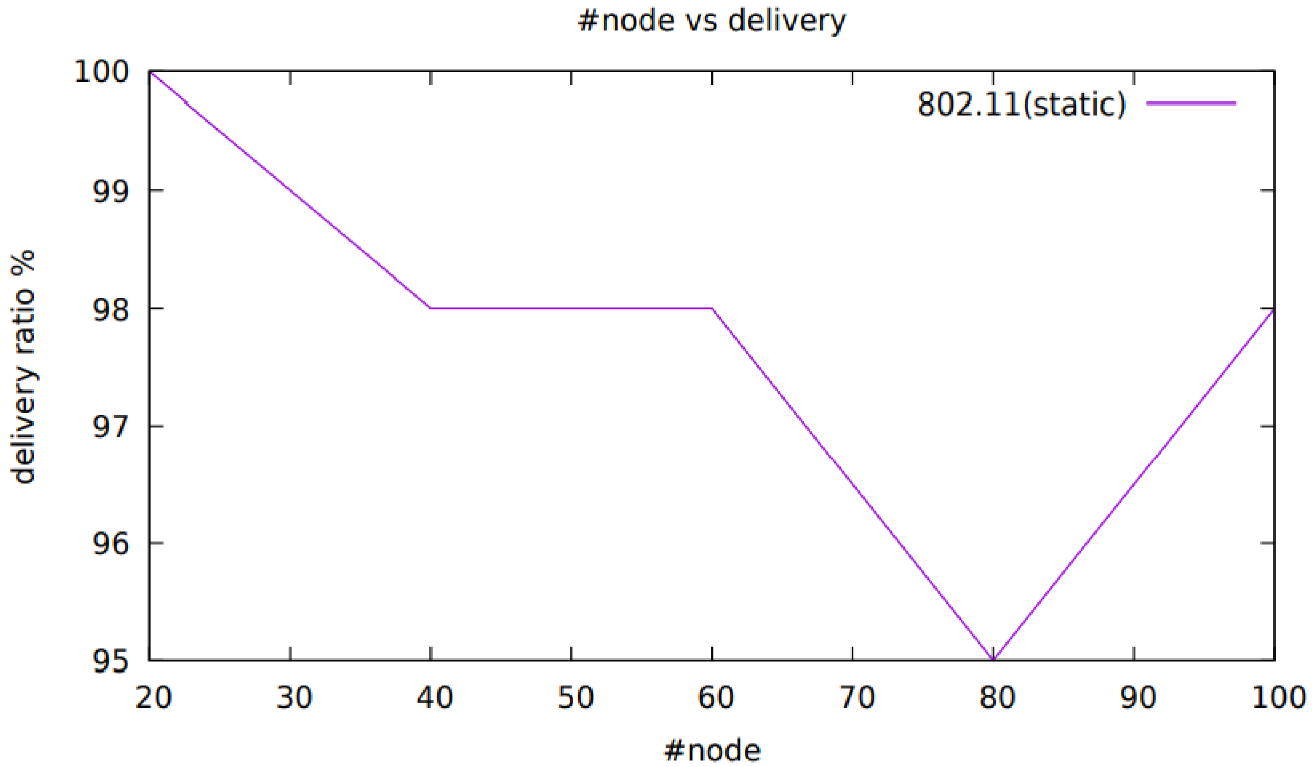
# 5 Results with graphs

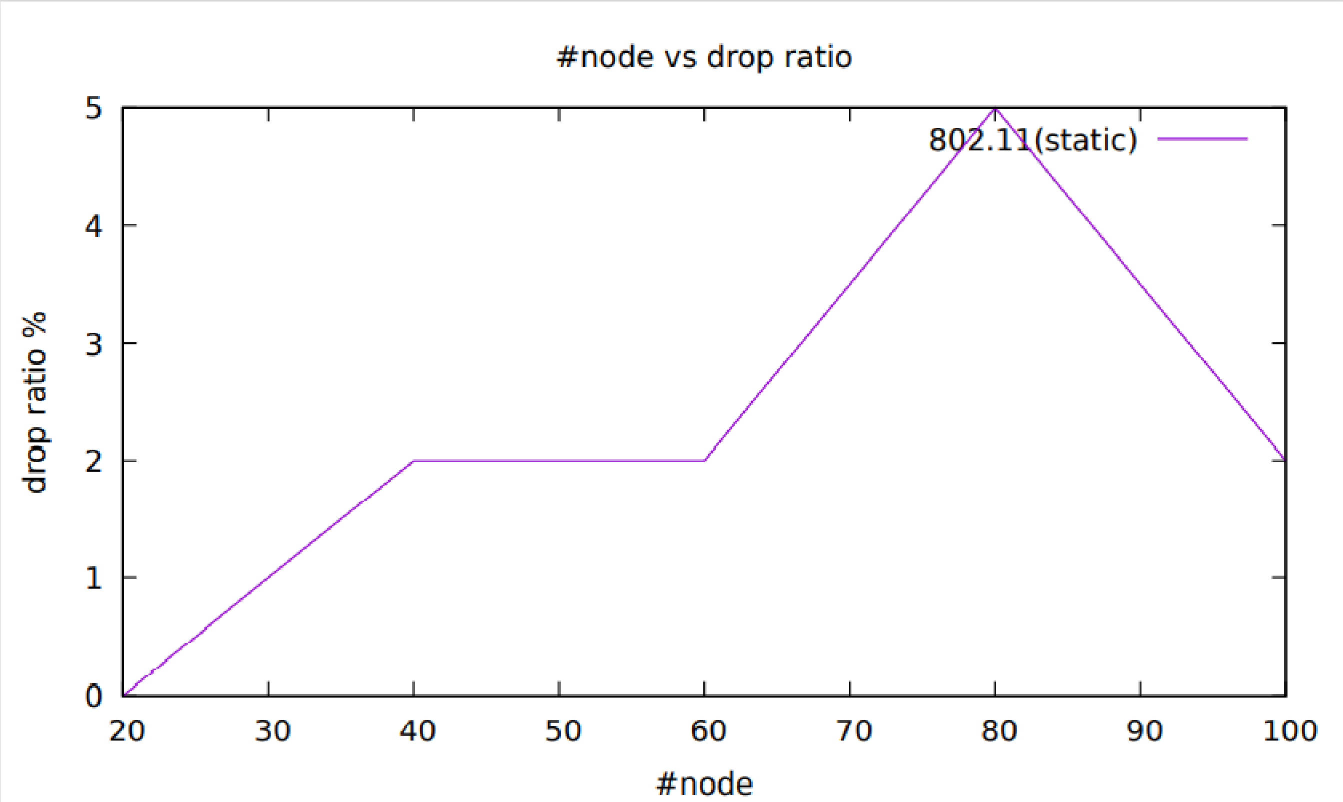
## 5.1 Varying Parameters Without Modifications(Task A)

### 5.1.1 Graphs for 802.11 static

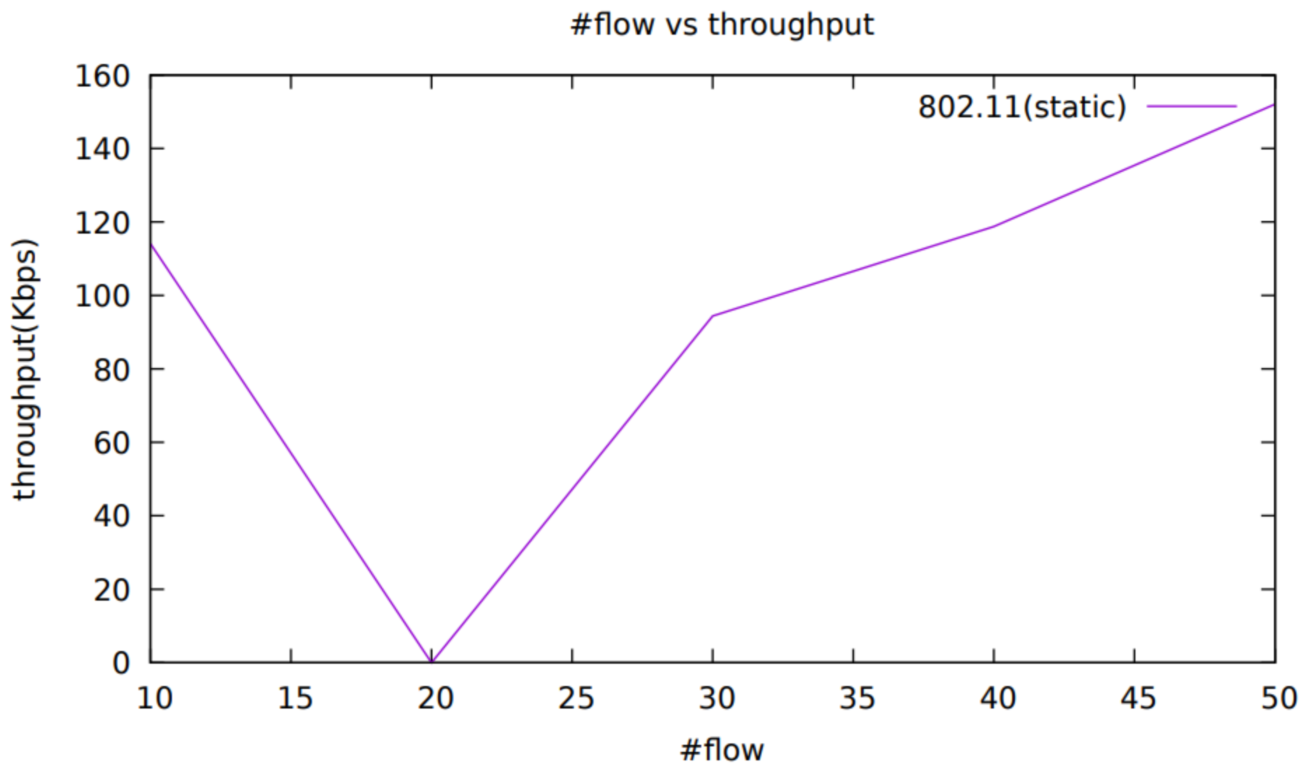


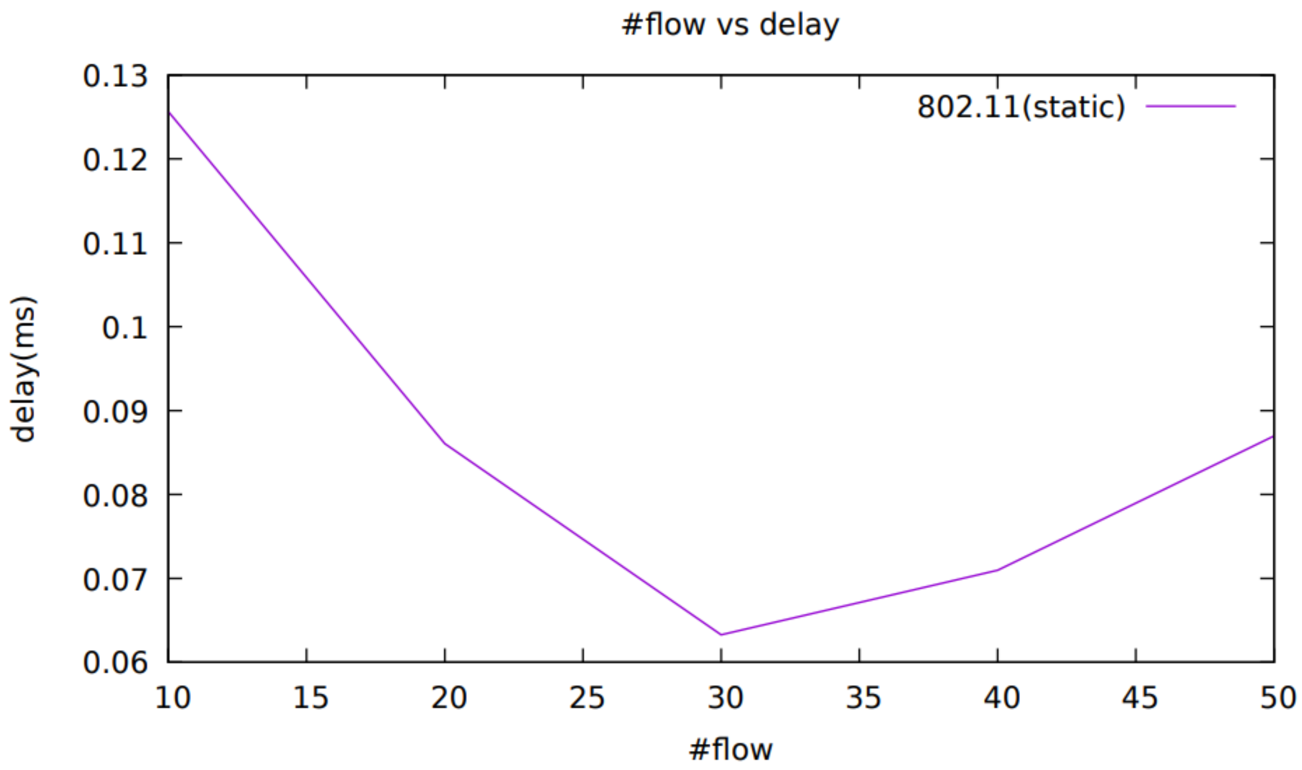

#node vs delay
802.11(static)
delay(ms)
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
20
30
40
50
60
70
80
90
100
#node

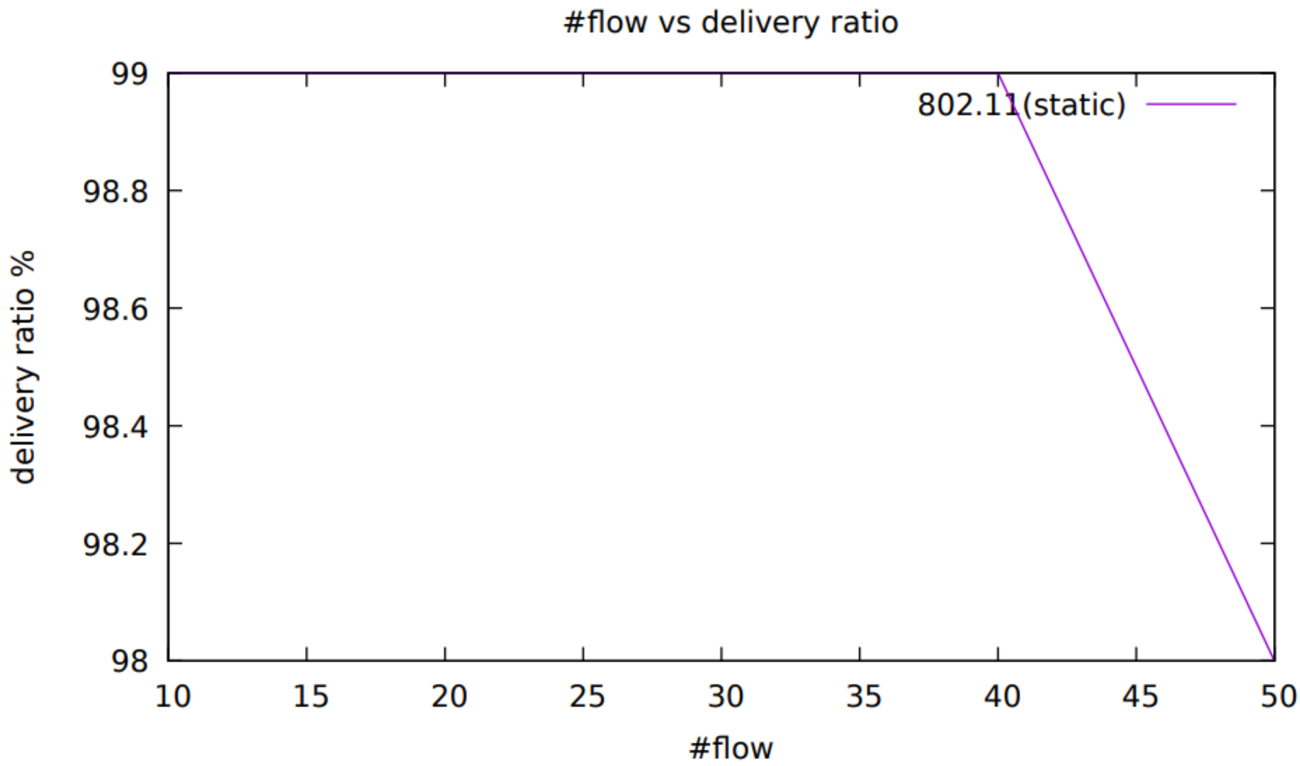

#node vs delivery
802.11(static)
delivery ratio %
100
99
98
97
96
95
20
30
40
50
60
70
80
90
100
#node

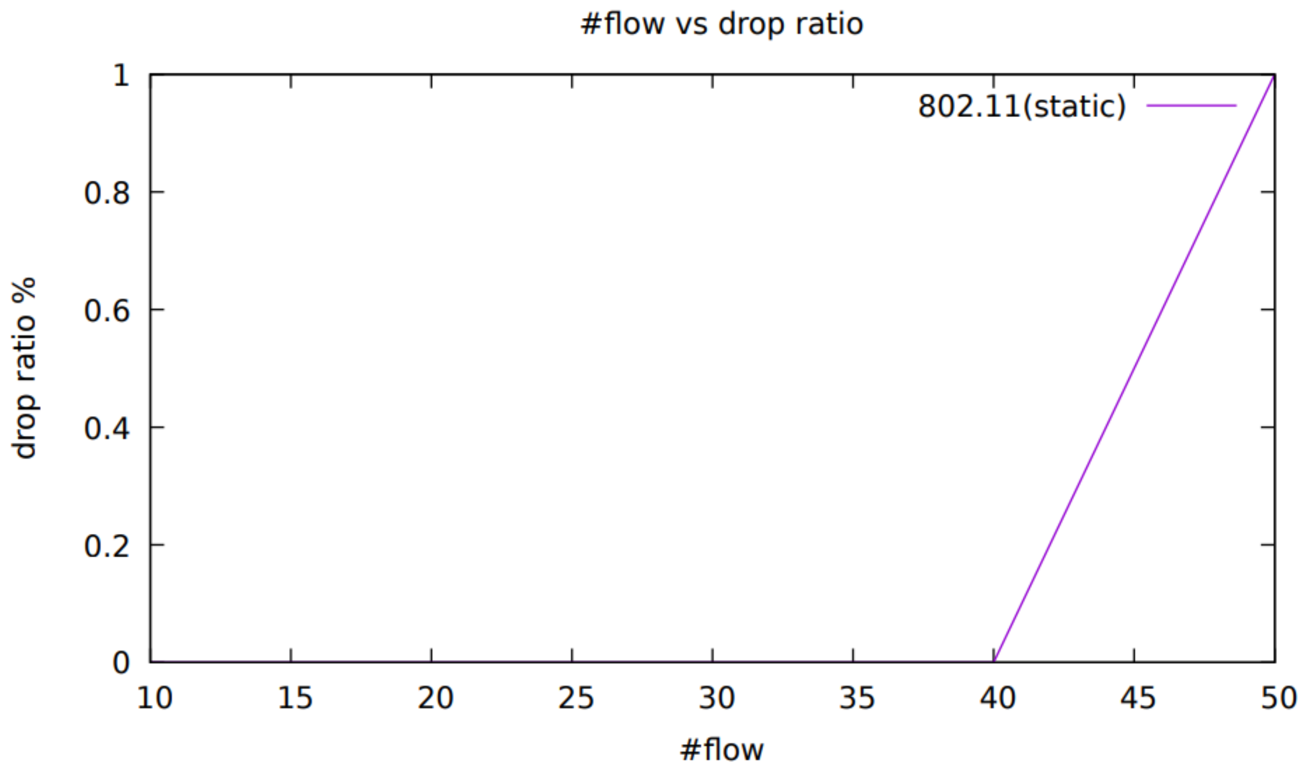


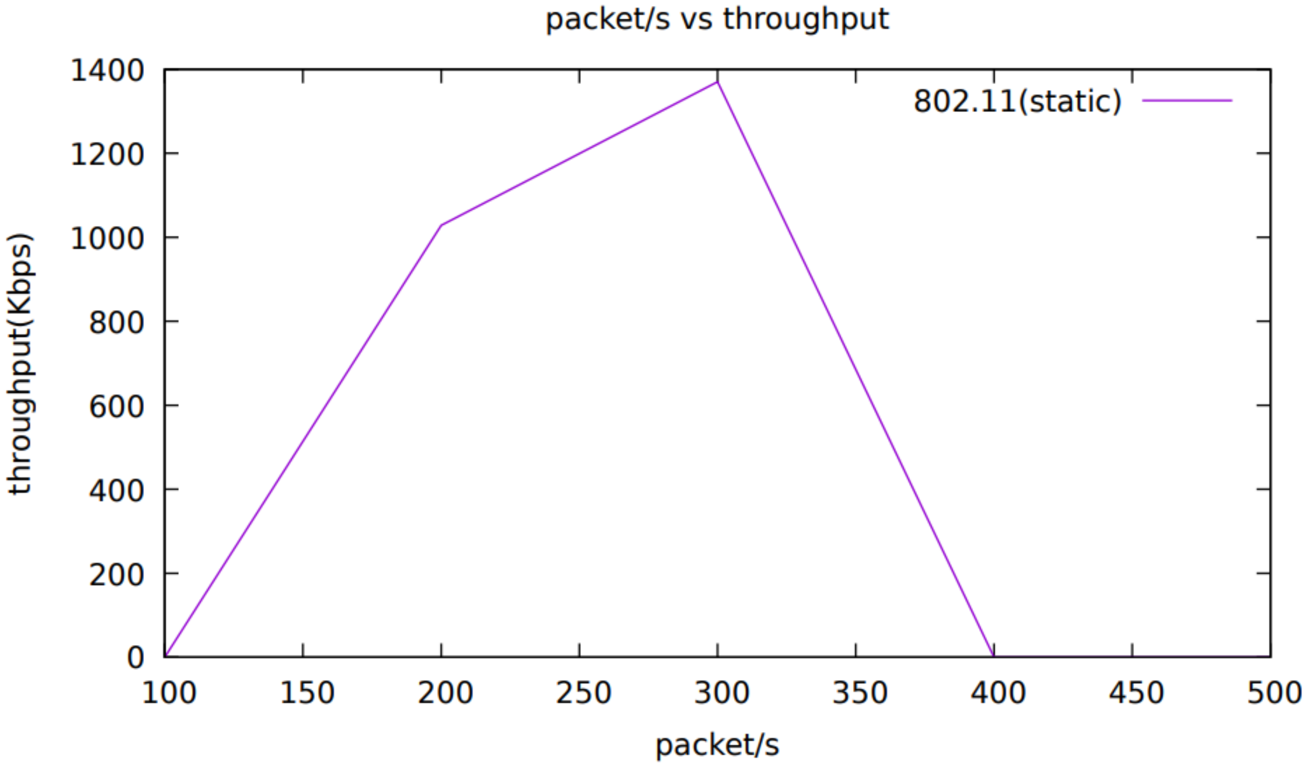
#node vs drop ratio
802.11(static)
drop ratio %
#node
0
1
2
3
4
5
20
30
40
50
60
70
80
90
100


#flow vs throughput
802.11(static)
throughput(Kbps)
160
140
120
100
80
60
40
20
0
10
15
20
25
30
35
40
45
50
#flow

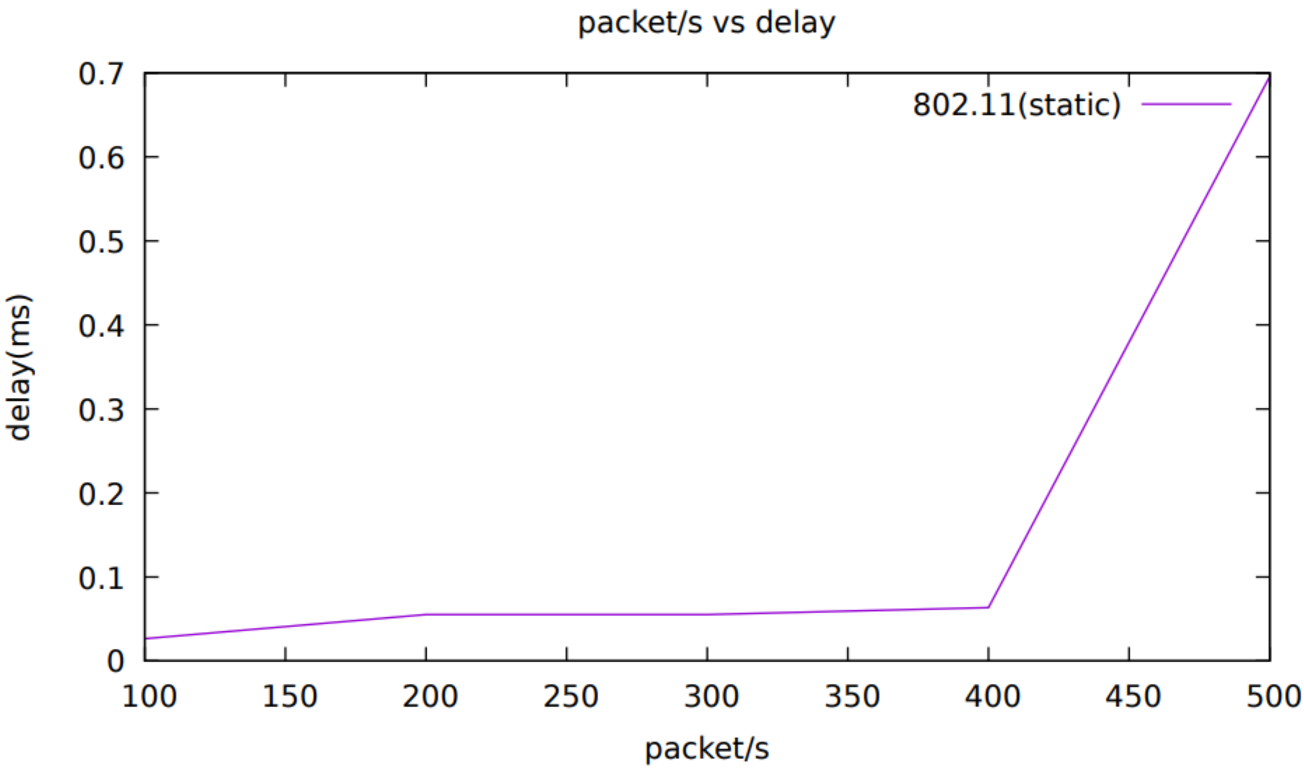

#flow vs delay
802.11(static)
delay(ms)
0.13
0.12
0.11
0.1
0.09
0.08
0.07
0.06
10
15
20
25
30
35
40
45
50
#flow

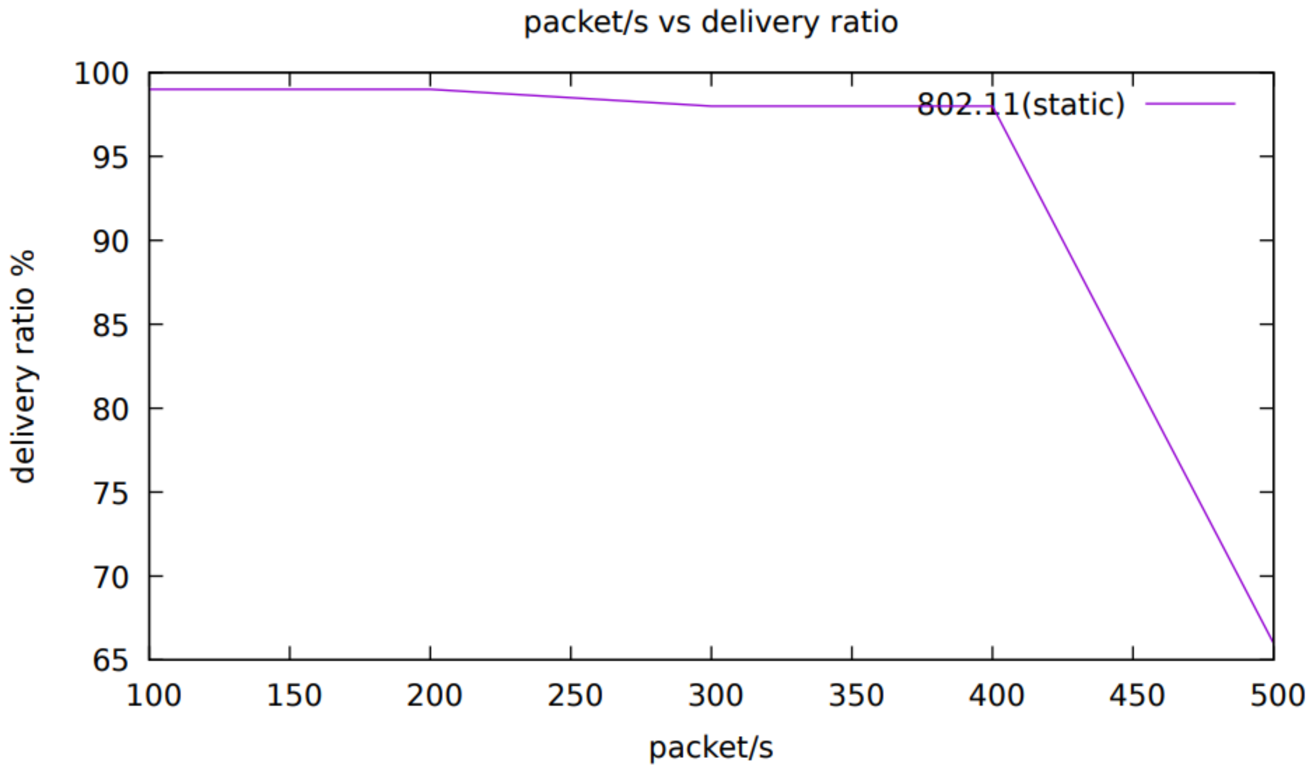

#flow vs delivery ratio
802.11(static)
delivery ratio %
99
98.8
98.6
98.4
98.2
98
10
15
20
25
30
35
40
45
50
#flow

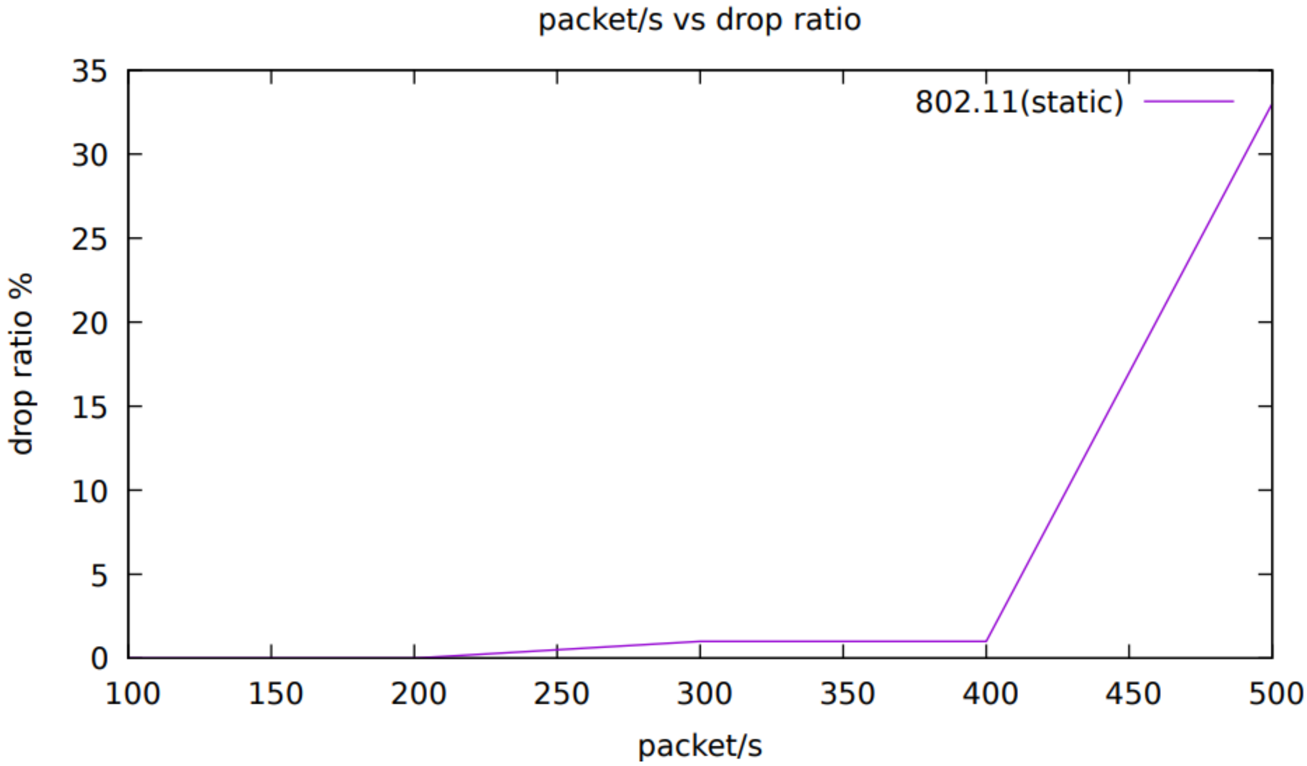

#flow vs drop ratio
802.11(static)
drop ratio %
#flow
1
0.8
0.6
0.4
0.2
0
10
15
20
25
30
35
40
45
50

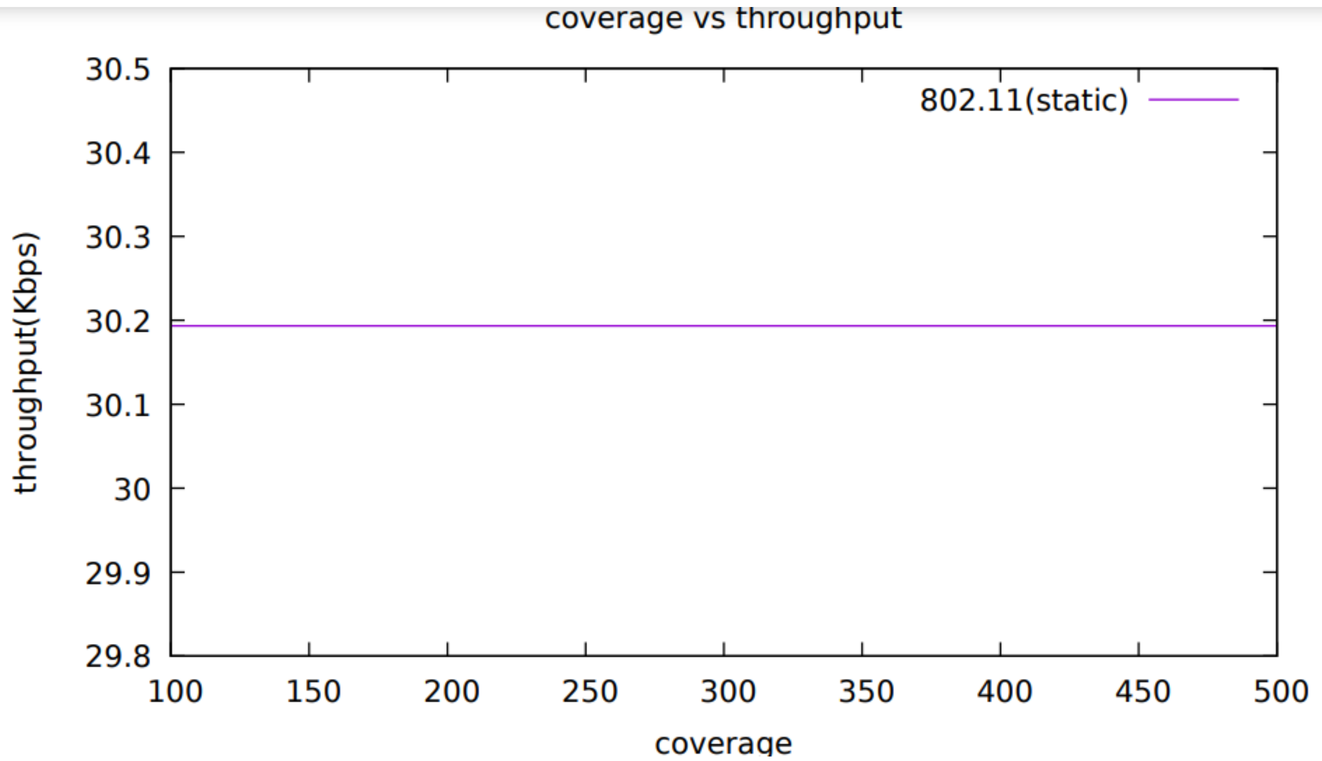


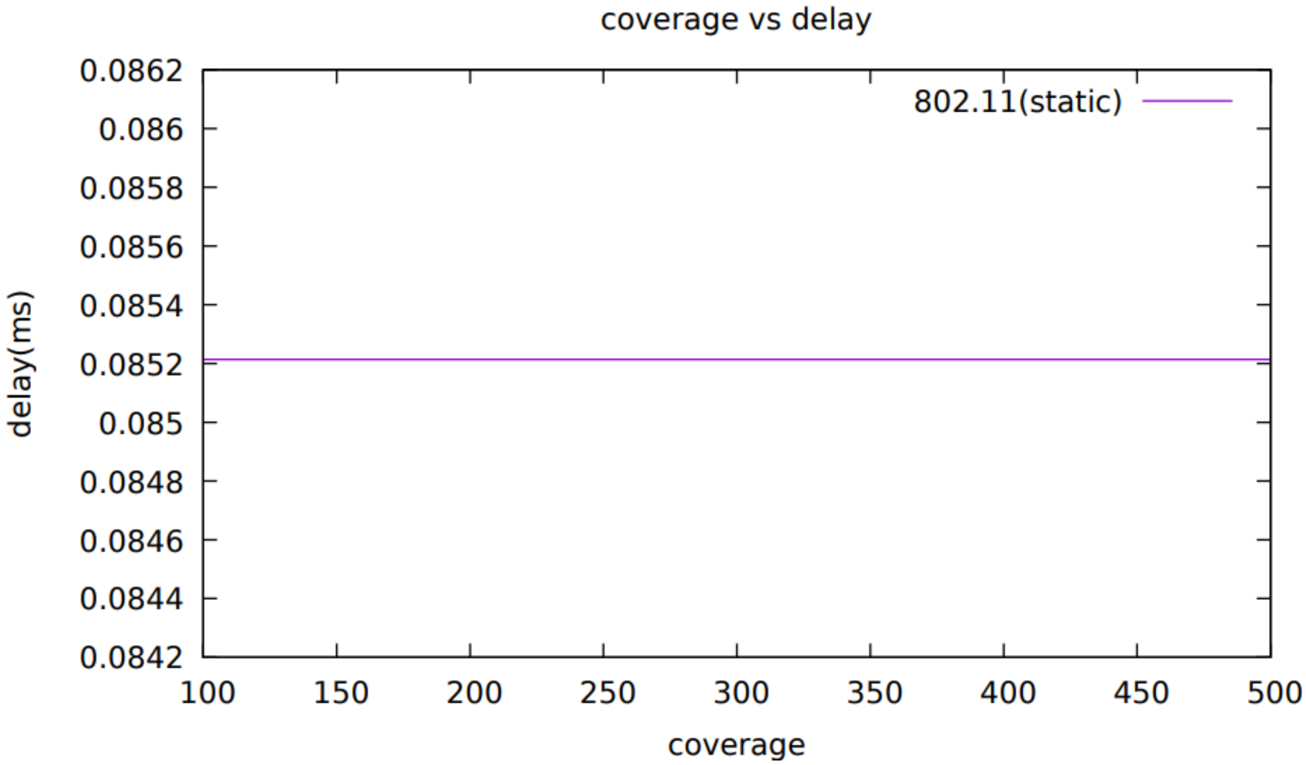
packet/s vs throughput
802.11(static)
throughput(Kbps)
1400
1200
1000
800
600
400
200
0
100
150
200
250
300
350
400
450
500
packet/s

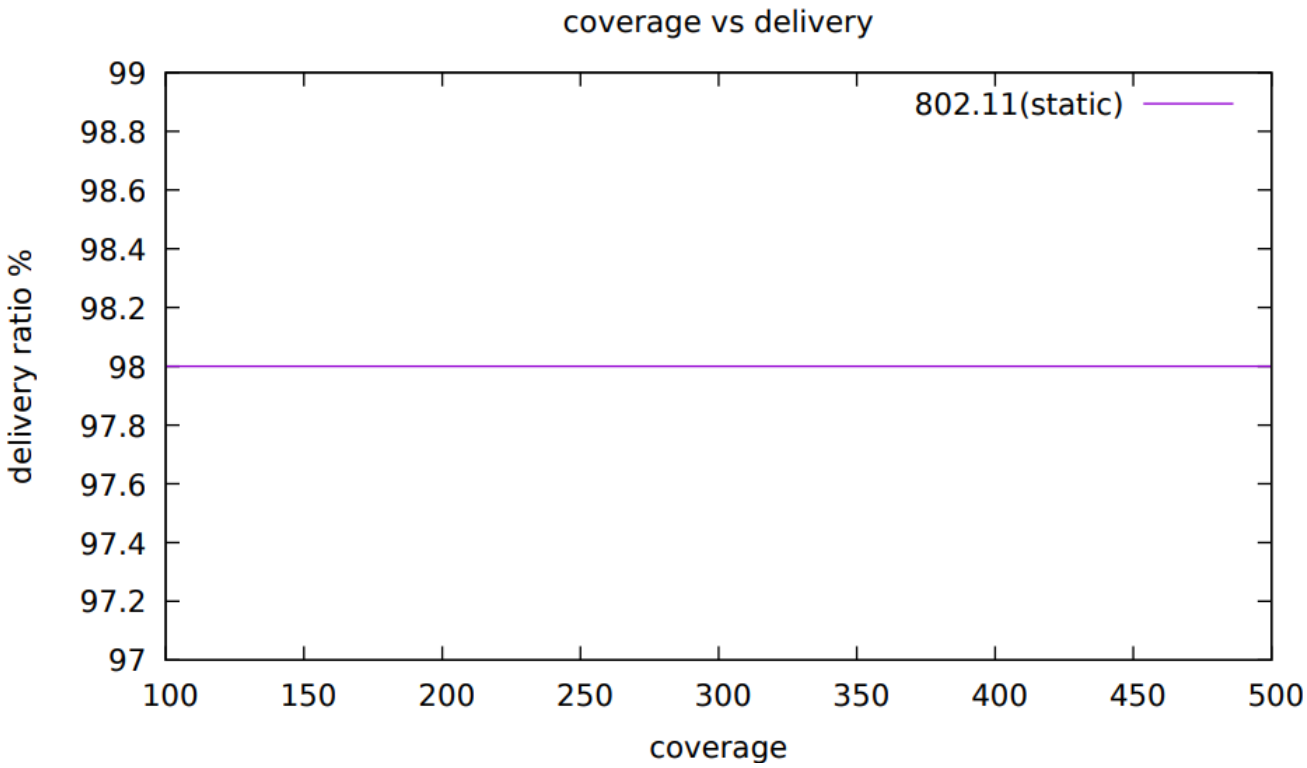

packet/s vs delay
802.11(static)
delay(ms)
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
100
150
200
250
300
350
400
450
500
packet/s

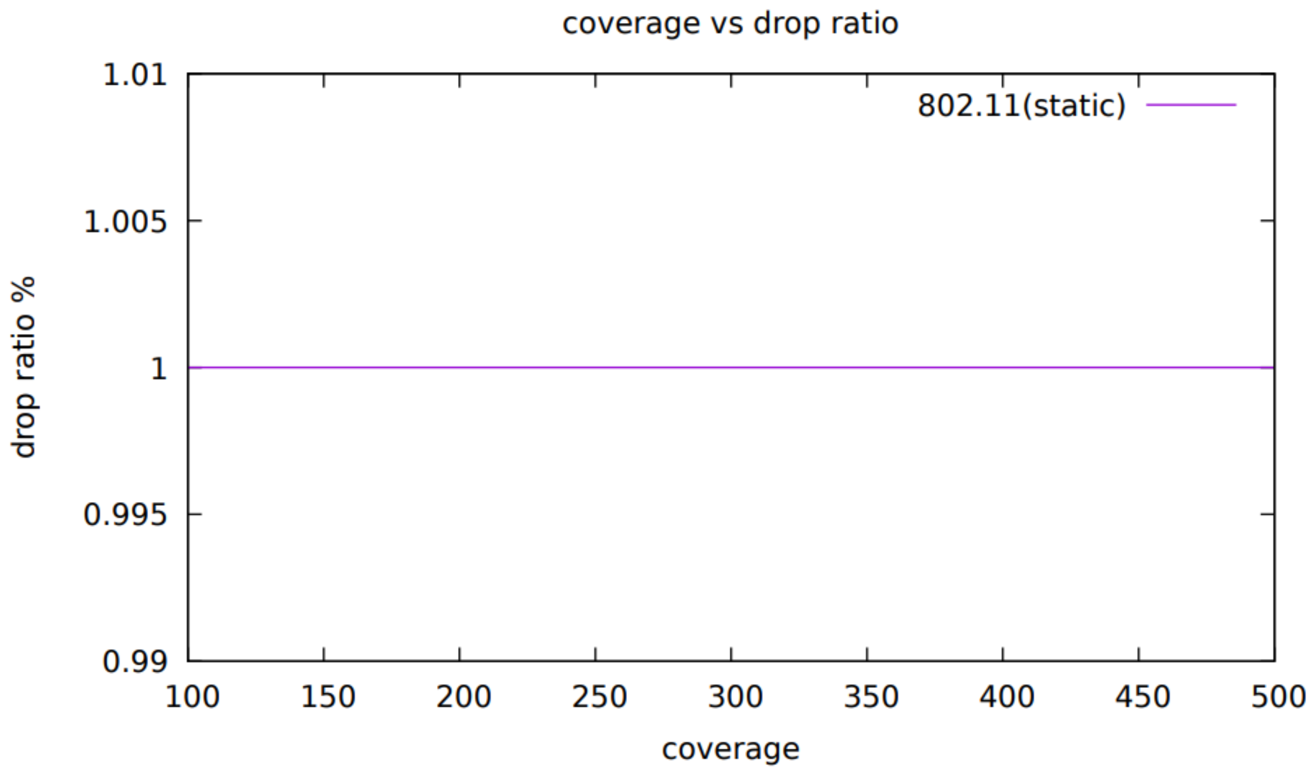

packet/s vs delivery ratio
802.11(static)
delivery ratio %
packet/s
100
95
90
85
80
75
70
65
100
150
200
250
300
350
400
450
500


packet/s vs drop ratio
802.11(static)
drop ratio %
packet/s
0
5
10
15
20
25
30
35
100
150
200
250
300
350
400
450
500

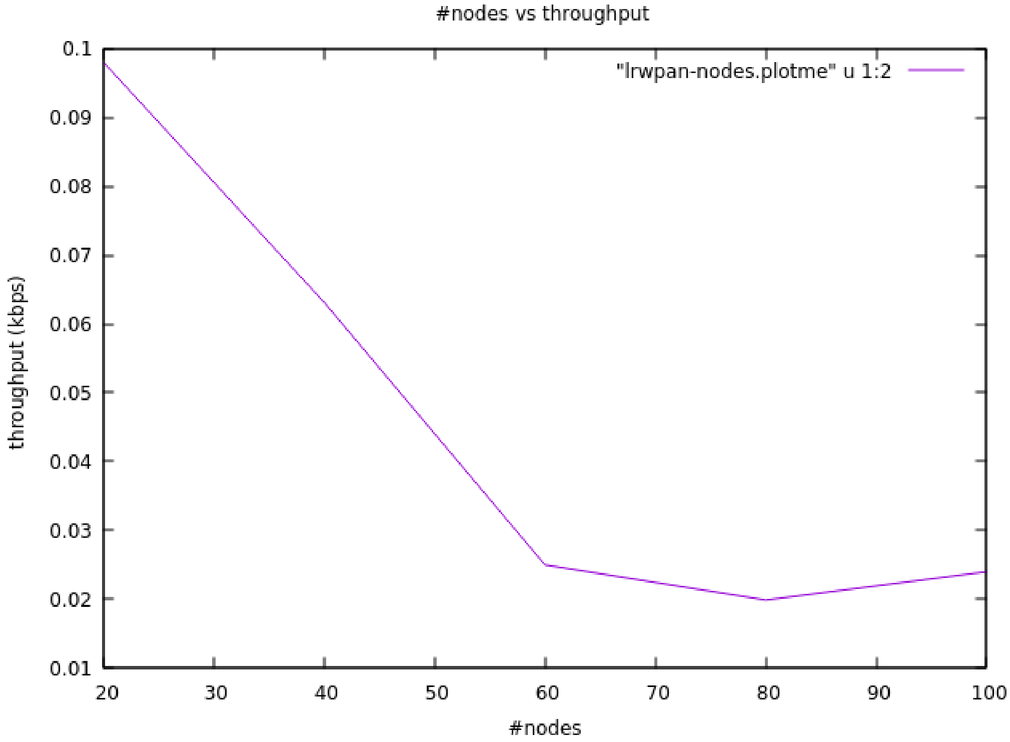

coverage vs throughput
802.11(static)
throughput(Kbps)
30.5
30.4
30.3
30.2
30.1
30
29.9
29.8
100
150
200
250
300
350
400
450
500
coverage

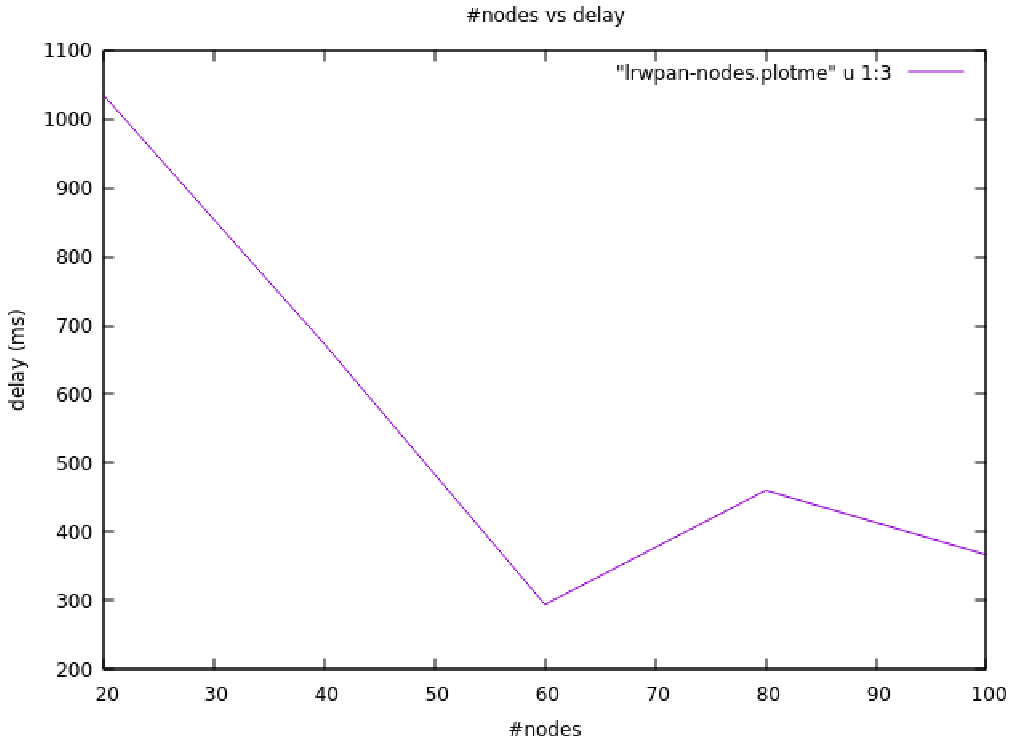

coverage vs delay
802.11(static)
delay(ms)
0.0862
0.086
0.0858
0.0856
0.0854
0.0852
0.085
0.0848
0.0846
0.0844
0.0842
100
150
200
250
300
350
400
450
500
coverage

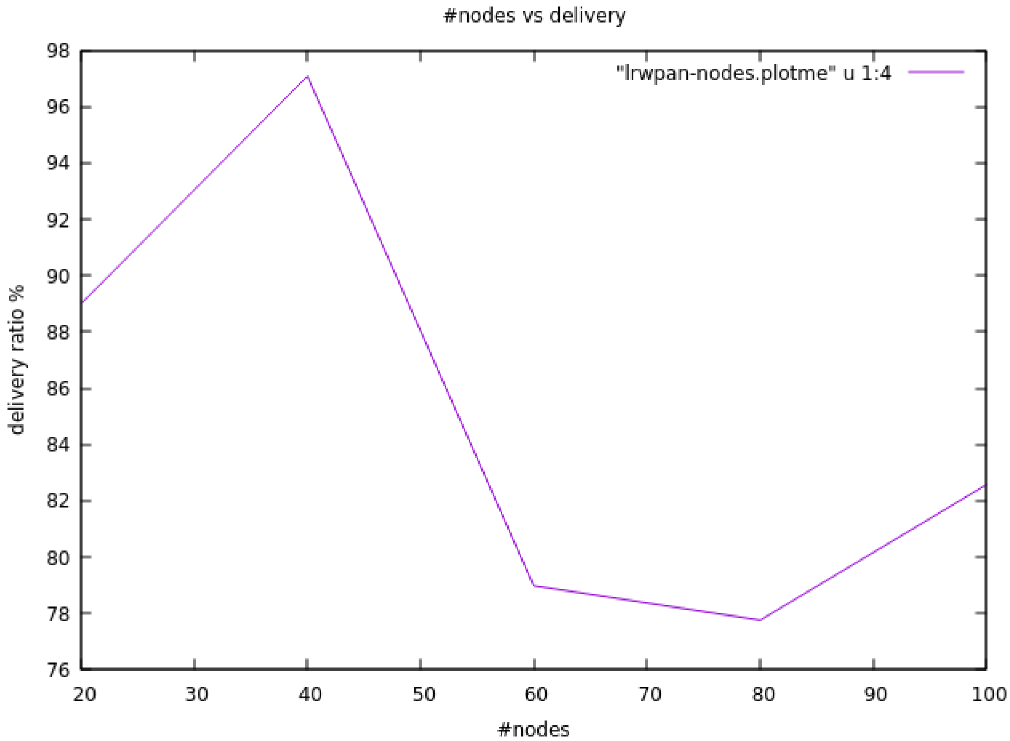

coverage vs delivery
802.11(static)
delivery ratio %
coverage
99
98.8
98.6
98.4
98.2
98
97.8
97.6
97.4
97.2
97
100
150
200
250
300
350
400
450
500

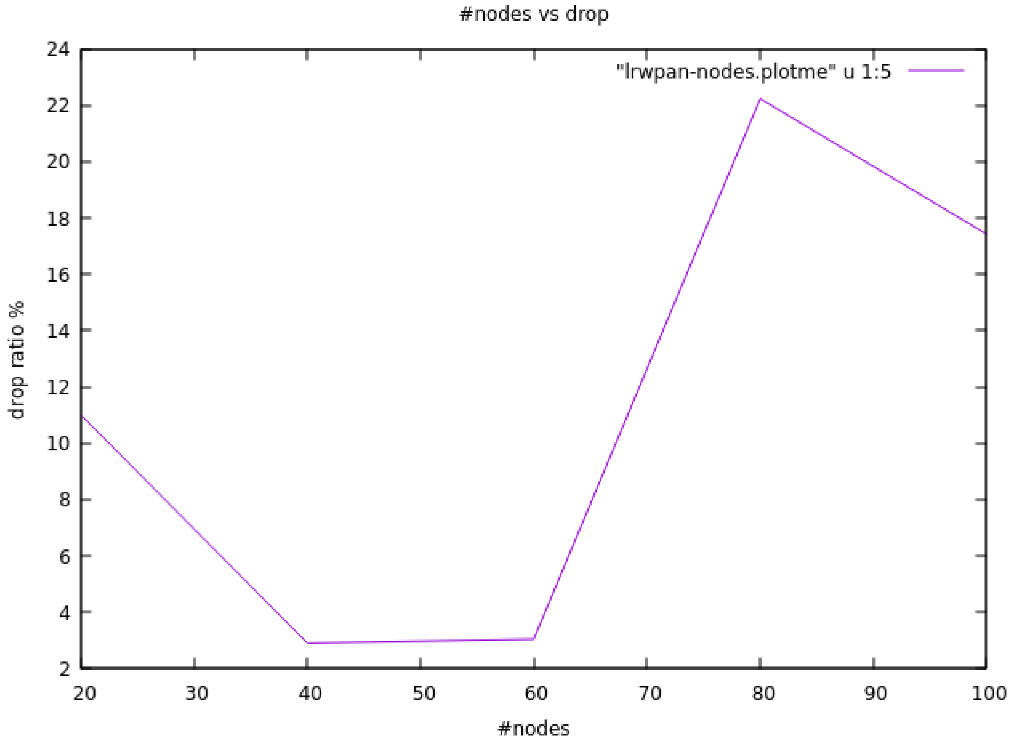

coverage vs drop ratio
802.11(static)
drop ratio %
coverage
1.01
1.005
1
0.995
0.99
100
150
200
250
300
350
400
450
500

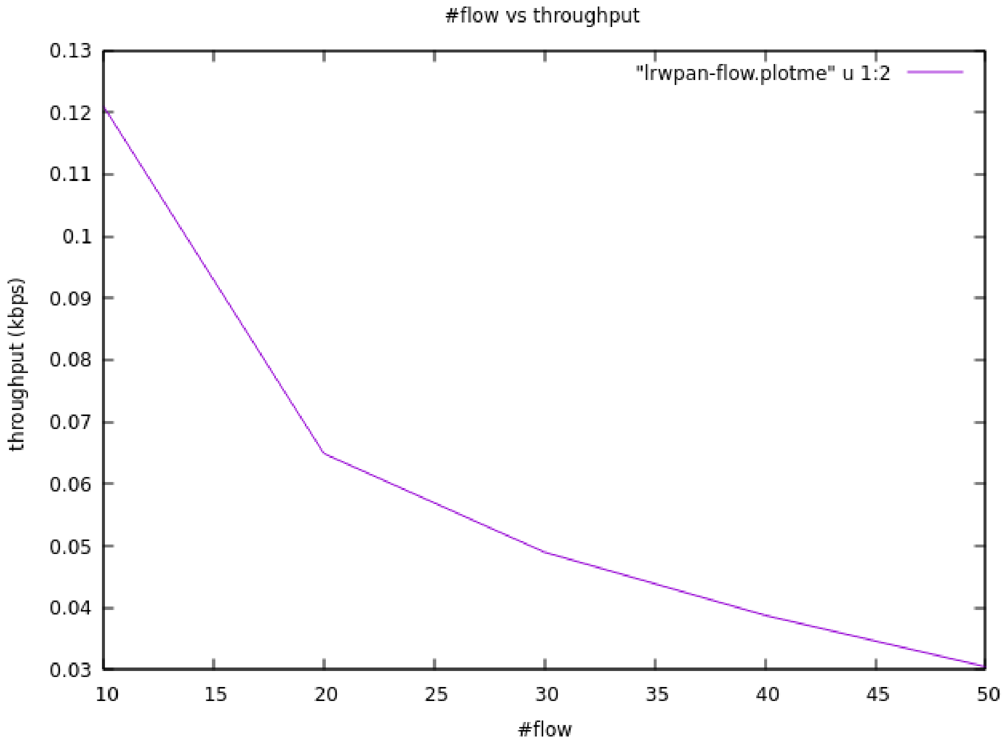
### 5.1.2 Graphs for 802.15.4 static

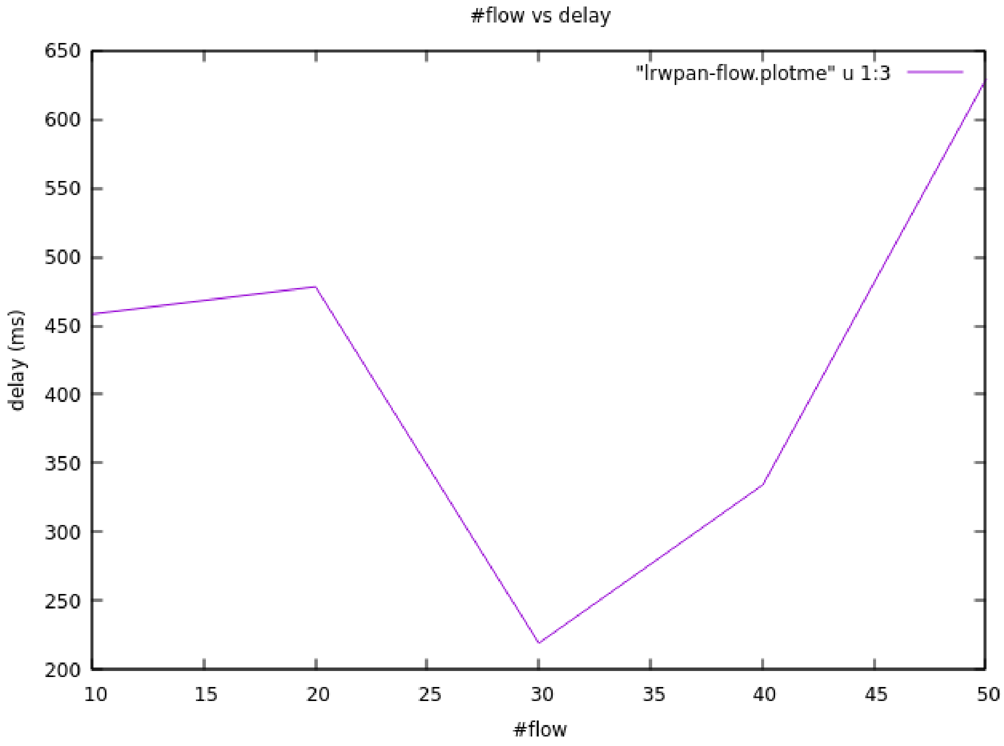


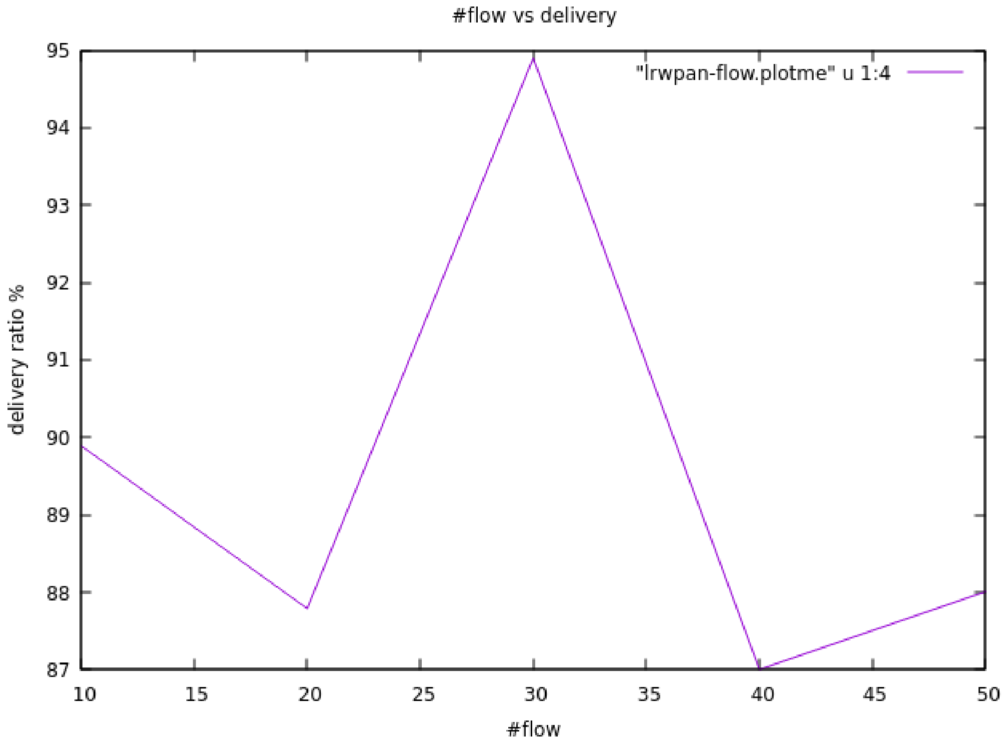

#nodes vs delay
"lrwpan-nodes.plotme" u 1:3
delay (ms)
1100
1000
900
800
700
600
500
400
300
200
20
30
40
50
60
70
80
90
100
#nodes

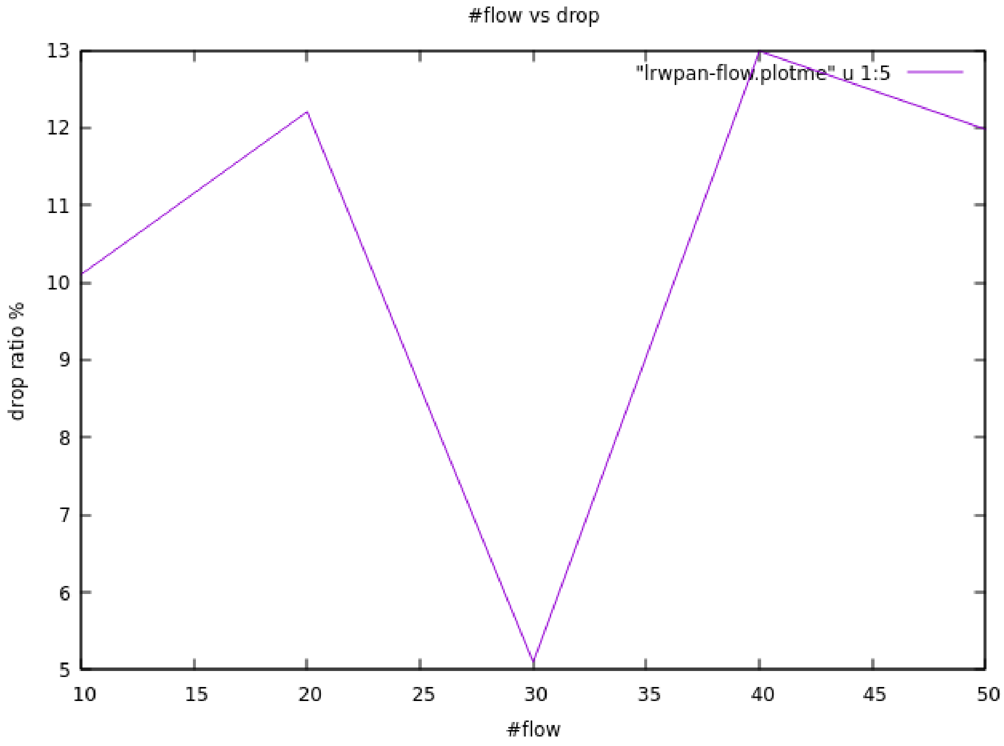

#nodes vs delivery
"lrwpan-nodes.plotme" u 1:4
delivery ratio %
98
96
94
92
90
88
86
84
82
80
78
76
20
30
40
50
60
70
80
90
100
#nodes

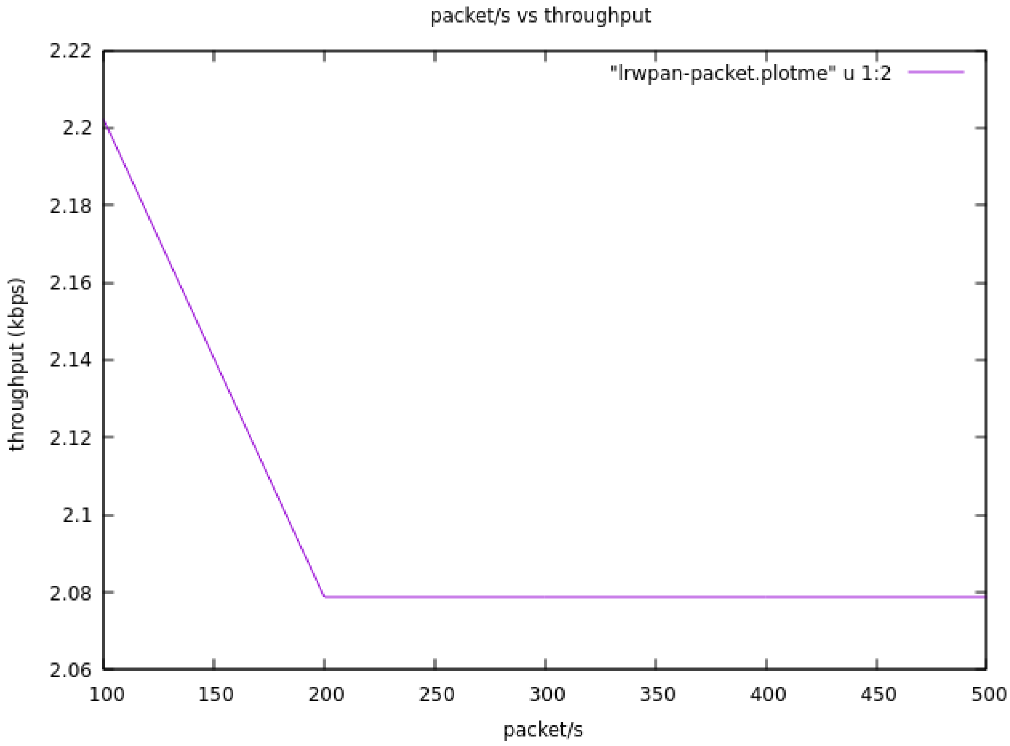

#nodes vs drop
"lrwpan-nodes.plotme" u 1:5
drop ratio %
#nodes
24
22
20
18
16
14
12
10
8
6
4
2
20
30
40
50
60
70
80
90
100

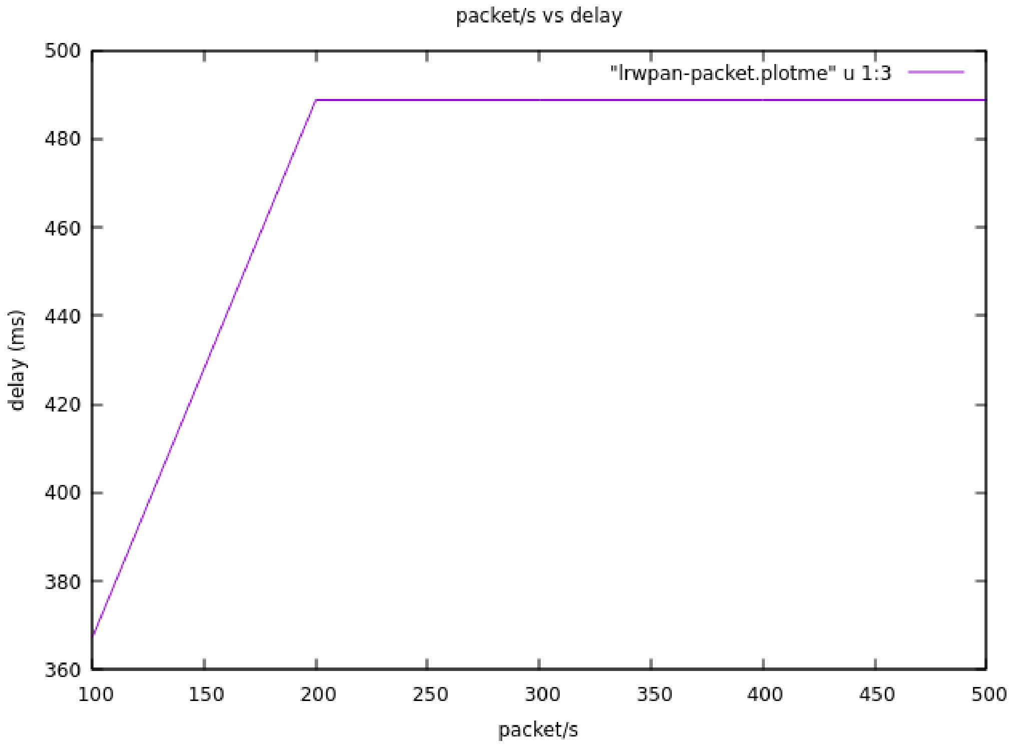

#flow vs throughput
"lrwpan-flow.plotme" u 1:2
throughput (kbps)
0.13
0.12
0.11
0.1
0.09
0.08
0.07
0.06
0.05
0.04
0.03
10
15
20
25
30
35
40
45
50
#flow

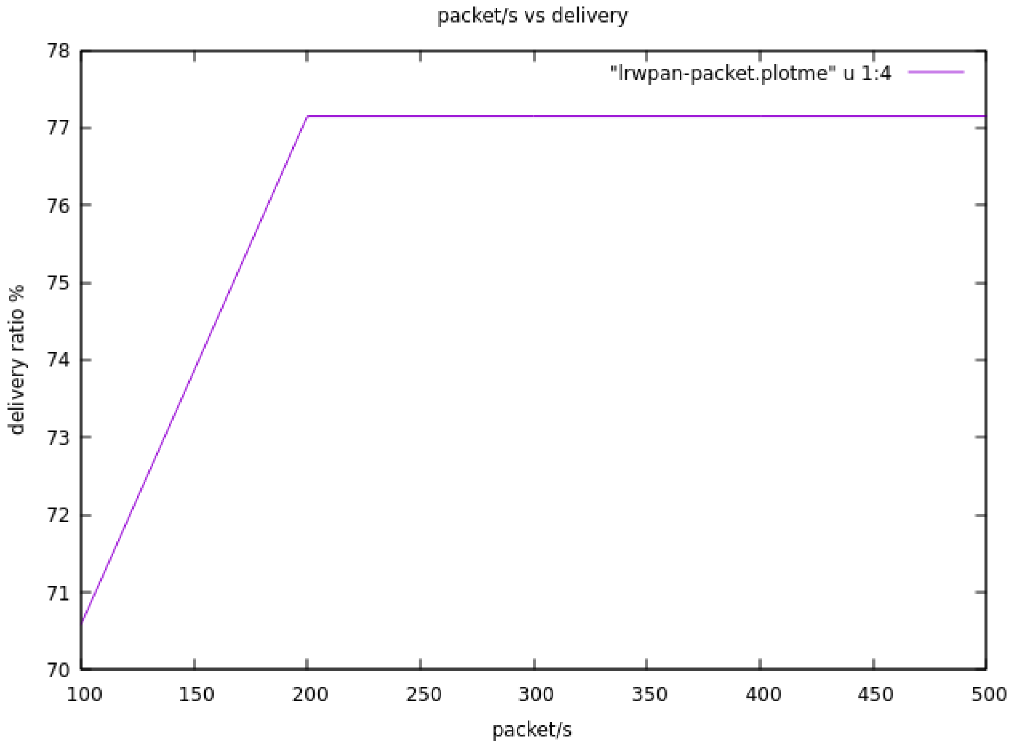

#flow vs delay
"lrwpan-flow.plotme" u 1:3
delay (ms)
#flow
650
600
550
500
450
400
350
300
250
200
10
15
20
25
30
35
40
45
50

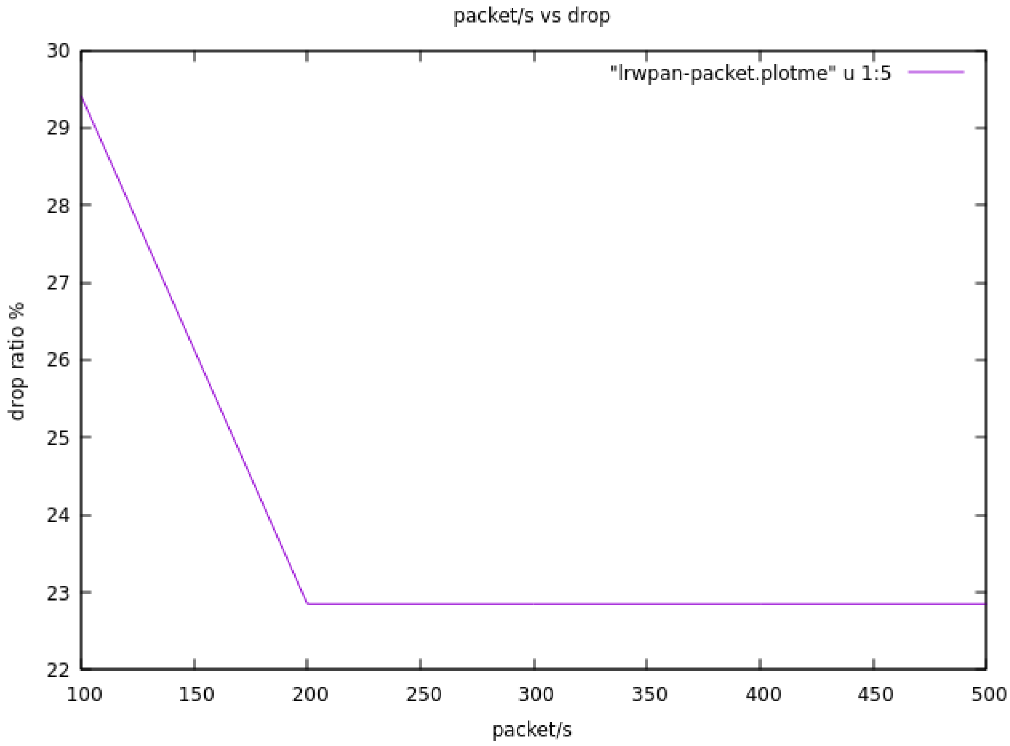

#flow vs delivery
"lrwpan-flow.plotme" u 1:4
delivery ratio %
95
94
93
92
91
90
89
88
87
10
15
20
25
30
35
40
45
50
#flow

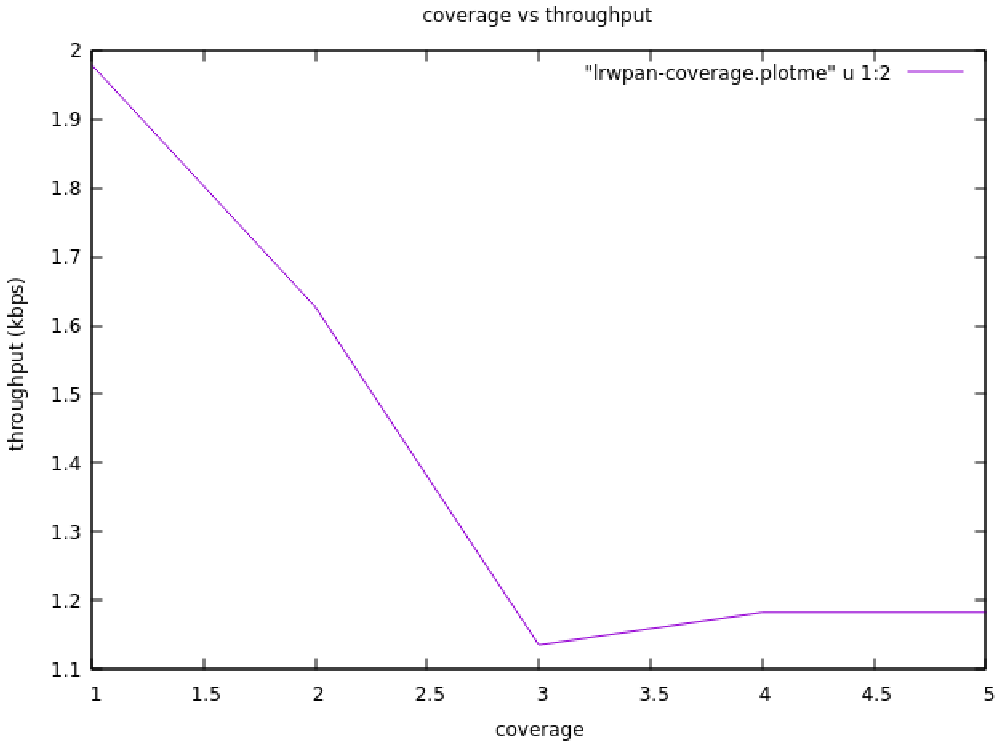

#flow vs drop
"lrwpan-flow.plotme" u 1:5
drop ratio %
#flow
13
12
11
10
9
8
7
6
5
10
15
20
25
30
35
40
45
50

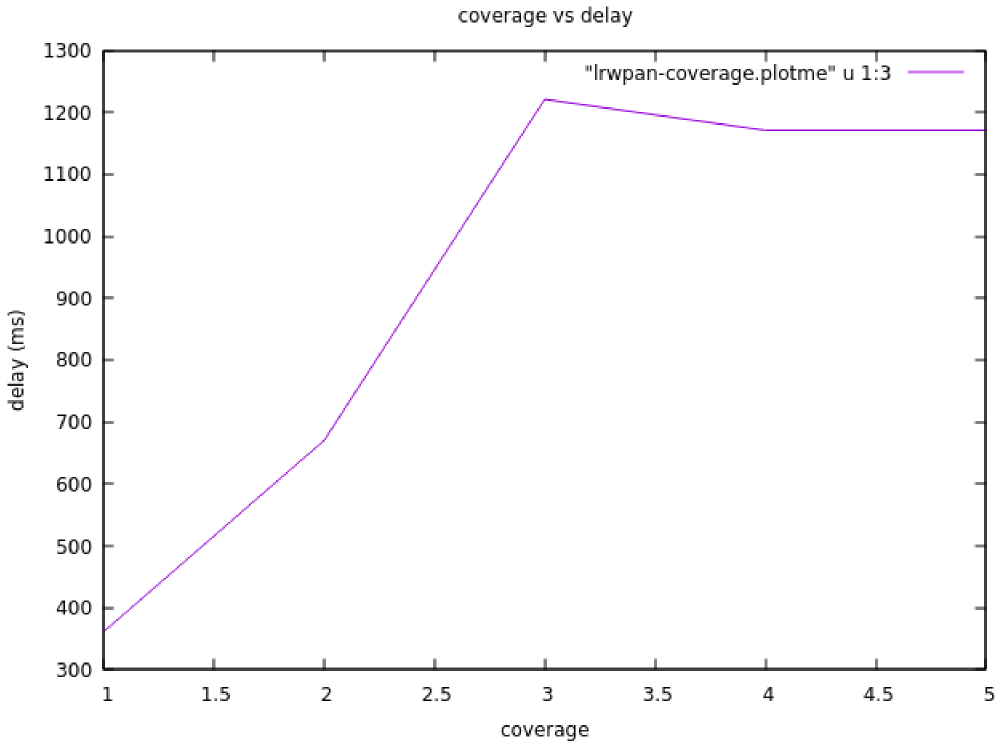

packet/s vs throughput
"lrwpan-packet.plotme" u 1:2
throughput (kbps)
2.22
2.2
2.18
2.16
2.14
2.12
2.1
2.08
2.06
100
150
200
250
300
350
400
450
500
packet/s

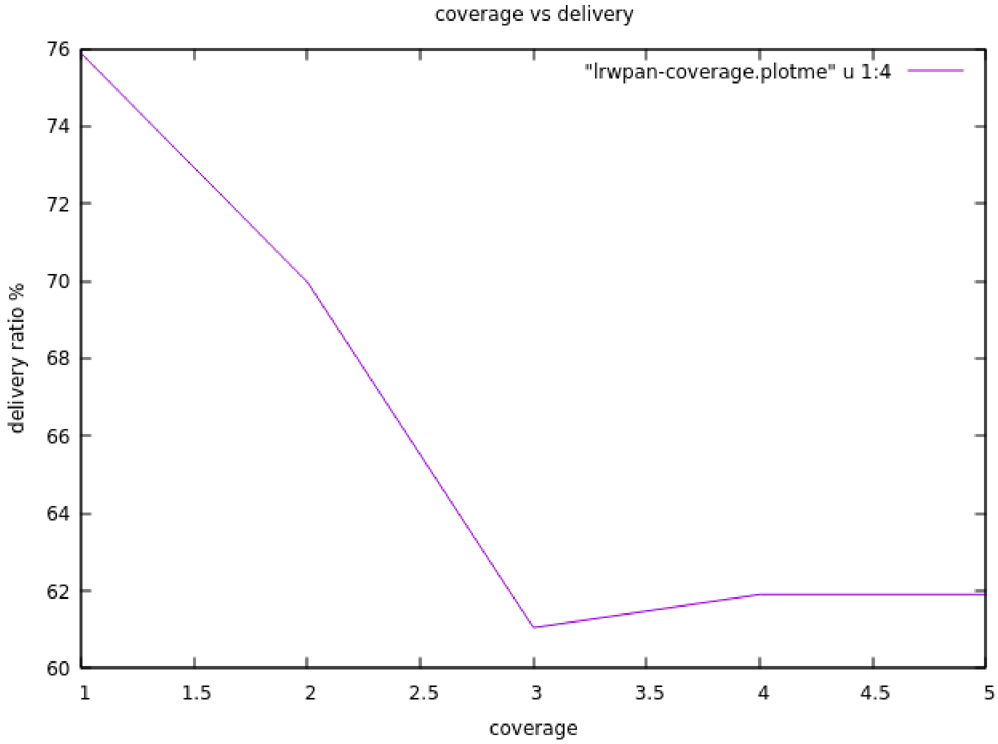

packet/s vs delay
"lrwpan-packet.plotme" u 1:3
delay (ms)
packet/s
500
480
460
440
420
400
380
360
100
150
200
250
300
350
400
450
500

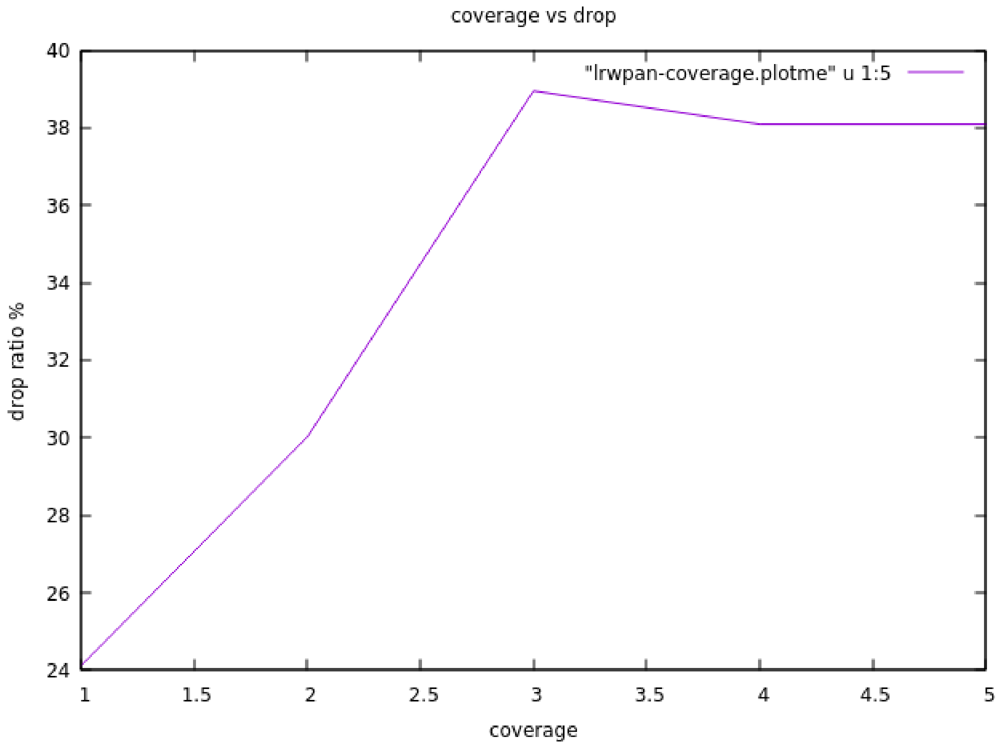

packet/s vs delivery
"lrwpan-packet.plotme" u 1:4
delivery ratio %
packet/s
78
77
76
75
74
73
72
71
70
100
150
200
250
300
350
400
450
500


packet/s vs drop
"lrwpan-packet.plotme" u 1:5
drop ratio %
30
29
28
27
26
25
24
23
22
100
150
200
250
300
350
400
450
500
packet/s


coverage vs throughput
"lrwpan-coverage.plotme" u 1:2
throughput (kbps)
coverage
2
1.9
1.8
1.7
1.6
1.5
1.4
1.3
1.2
1.1
1
1.5
2
2.5
3
3.5
4
4.5
5

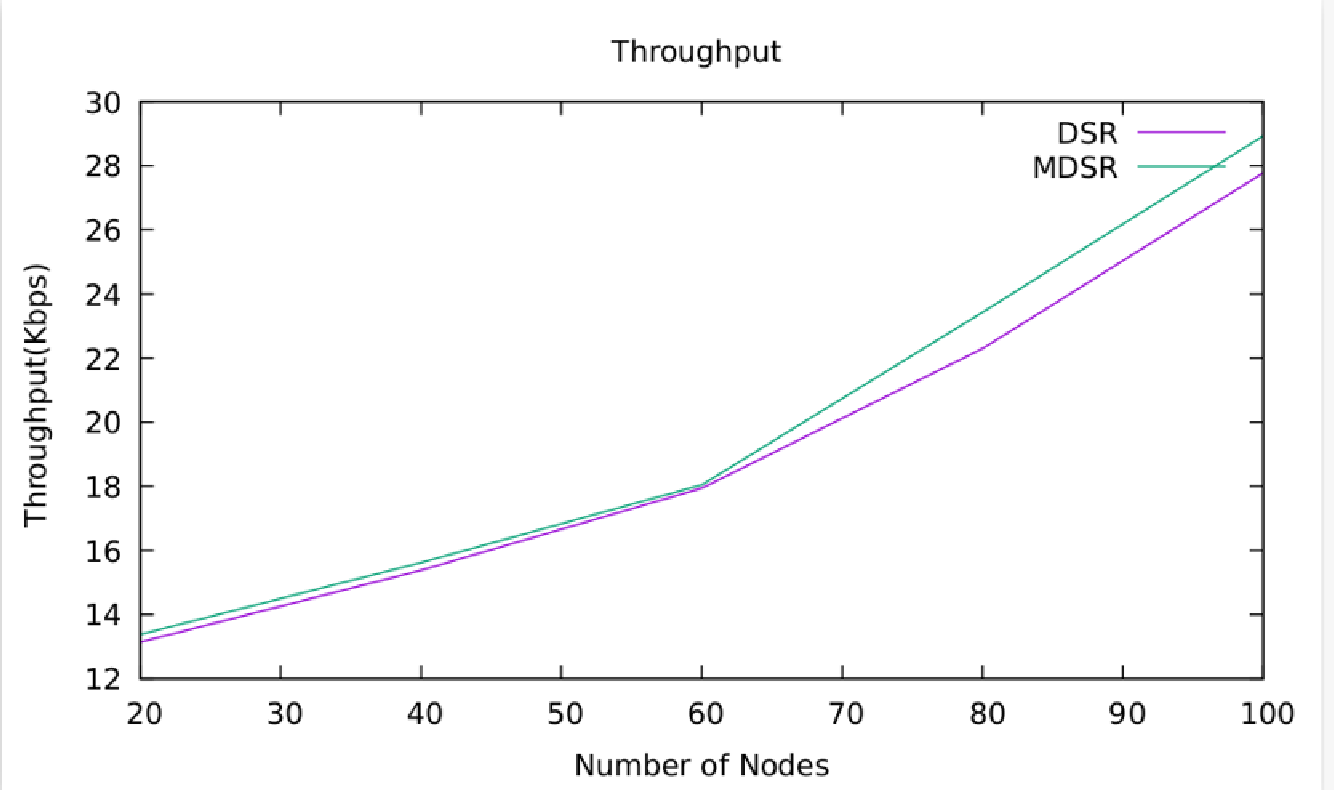

coverage vs delay
"lrwpan-coverage.plotme" u 1:3
1300
1200
1100
1000
900
800
700
600
500
400
300
delay (ms)
1
1.5
2
2.5
3
3.5
4
4.5
5
coverage


coverage vs delivery
"lrwpan-coverage.plotme" u 1:4
delivery ratio %
coverage
76
74
72
70
68
66
64
62
60
1
1.5
2
2.5
3
3.5
4
4.5
5

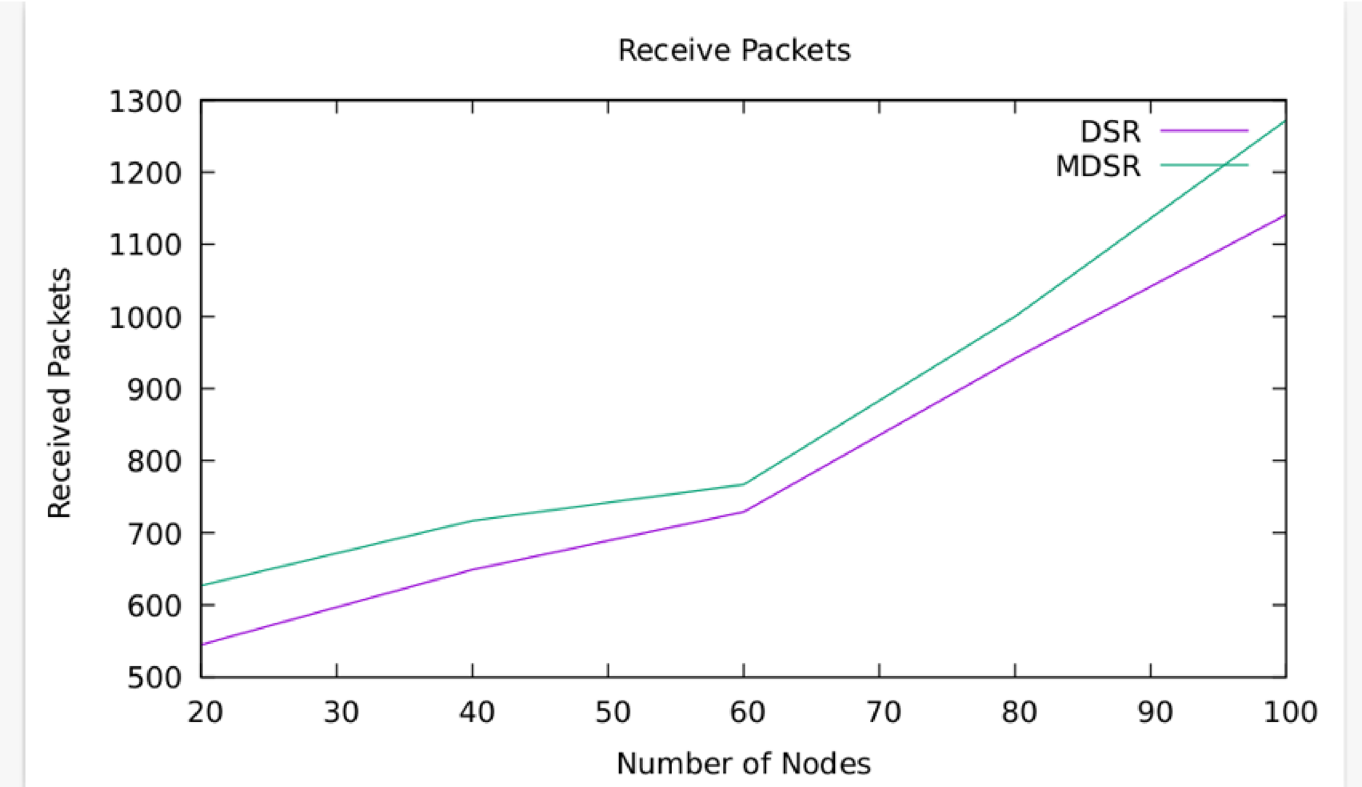

coverage vs drop
"lrwpan-coverage.plotme" u 1:5
drop ratio %
coverage
40
38
36
34
32
30
28
26
24
1
1.5
2
2.5
3
3.5
4
4.5
5

## 5.2 Graphs for Simulation with Proposed Modification (Task B)

### 5.2.1 Throughput



### 5.2.2 Received Packet



# 6 Summary findings explaining the results

## 6.1 TaskA Summary

### 6.1.1 Result Analysis for Wireless high-rate (802.11 static)

| When Value is Increased | Throughput | End-End-Dela y | Delivery Ratio | Drop Ratio |
|---|---|---|---|---|
| Node | Increased | Increased | First Decreased then Increased | First Increased then Decreased |
| Flow | First Decreased then Increased | First Decreased then Increased | Decreased after 40 | Increased after 40 |
| Packet/s | First Increased then Decreased | Increased | Decreased | Increased |
| Coverage | No Change | No Change | No Change | No Change |

### 6.1.2 Result Analysis for Wireless low-rate (802.15.4 static)

| When Value is Increased | Throughput | End-End-Delay | Delivery Ratio | Drop Ratio |
|---|---|---|---|---|
| Node | Decreased | Decreased | First Increased then Decreased | First Decreased then Increased |
| Flow | Decreased | First Increased then Decreased | First Increased then Decreased | First Decreased then Increased |
| Packet/s | Decreased | Increased | Increased | Decreased |
| Coverage | Decreased | Increased | Decreased | Increased |

## 6.2 Comparison Summary of Task$_A$

- **From the measurements we observed that performance of 802.15.4 with respect to the delay and packet dropped parameter is good as compared to 802.11, whereas with increased number of nodes jitter remains almost constant, for both 802.11 and 802.15.4 networks.**
- **Performance of 802.11 is observed to be consistently better than that of 802.15.4. The reason is that, the MAC layer of 802.11 can quickly and efficiently adapt to a higher number of nodes than that of 802.15.4.**

## 6.3 TaskB Summary

### 6.3.1 Receive Packet Analysis

- Observed Result :
- **In taskB the received packet graph shows that modified DSR(MDSR) receive more packet than normal DSR with the respect to the number of nodes.**
- Logical Reasons :
- **Modified DSR increases Received Packet as compare to conventional DSR protocol because all nodes know about broken link information using our approach, RERR notification is send to all the nodes and node update their cache .As a result modified DSR perform better than conventional DSR for Packet Delivery Ratio,Packet Loss Ratio.**

### 6.3.2 Throughput Analysis

- **Observed Result :**
- **In taskB the throughput graph shows that modified DSR(MDSR) has more throughput than normal DSR with the respect to the number of nodes.**
- **Logical Reasons :**
- **As compare to other routing protocol DSR has low overhead because of route cache present. Instead of initiating new RREQ every time, it sees in to cache. Our new approach reduces the overhead using REER notification to nodes present in topology. stale route entries are remove from the cache which reduce the overhead and improve QoS of network. As a result , the throughput is increased using modified DSR in the above throughput graph .**