Data Communication [CS255]
Mini Project Report
Semester : IV

Department of Computer Science
NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA (NITK) SURATHKAL, MANGALORE - 575 025

Submitted by:

- S B L Prateek – 231CS149
- A R Sharan Kumar – 231CS101
- Manoj Barki - 231CS233

Data Communication Final Project Report

Problem Statement

Problem:
Consider CSMA/CA protocol in Wifi networks that working in ad-hoc mode.

Evaluate performance of the protocol without RTS/CTS scheme when the number of nodes within a communication range increases from 2 to 30.

Tasks Analysis

1. Review:

- Ad-hoc mode:

> A wireless network structure where devices can communicate directly with each other.

⇒ Also called peer-to-peer mode. (Decentralized, No access point)

- Request to Send / Clear to Send mechanism

A reservation scheme used in the wireless networks. It is used to minimize frame collisions before transmitting.

2. Design the network topology

- Disable RTS/CTS

```
UintegerValue threshold = 1000;
Config::SetDefault("ns3::WifiRemoteStationManager::RtsCtsThreshold",  threshold);
```

⇒ If packet size < 1000 bytes then RTS/ CTS will be disabled

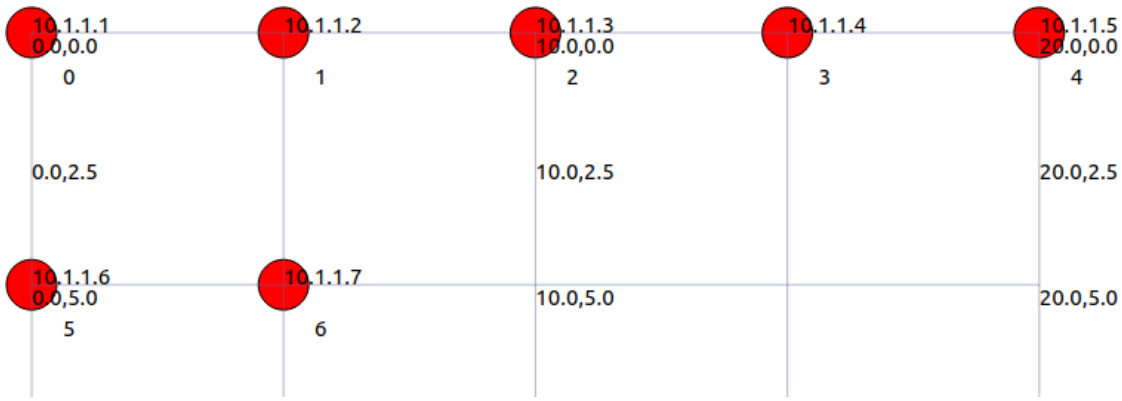
- Create nodes and install Wi-fi model

```
// Create the nodes that compose the network
NodeContainer nodes;
nodes.Create(nNodes);

// Configure the PHY and channel helpers
YansWifiChannelHelper channel = YansWifiChannelHelper::Default();
YansWifiPhyHelper phy;
/* Create a channel object and associate it to PHY layer  object manager
to make sure that all the PHY layer objects created by the YansWifiPhyHelper
share the same underlying channel*/
phy.SetChannel(channel.Create());

// WifiMacHelper is used to set MAC parameters.
WifiMacHelper mac;

/* Instantiate WifiHelper (By default, configure the standard in use to be 802.11ax
and configure a compatible rate control algorithm - IdealWifiManager)*/
WifiHelper wifi;

// Configure the MAC layer
mac.SetType("ns3::AdhocWifiMac");

NetDeviceContainer  devices;
devices = wifi.Install(phy, mac, nodes);
```

⇒ Configure MAC layer using “ns3::AdhocWifiMac” to make the network works in ad-hoc mode

- Install the mobility model

```
/* Instantiate a MobilityHelper object and set some attributes controlling
the "position allocator" functionality */
MobilityHelper mobility;
mobility.SetPositionAllocator(
    "ns3::GridPositionAllocator",
    "MinX", DoubleValue(0.0),
    "MinY", DoubleValue(0.0),
    "DeltaX", DoubleValue(5.0),
    "DeltaY", DoubleValue(5.0),
    "GridWidth", UintegerValue(5),
    "LayoutType", StringValue("RowFirst"));

/* Set the mobility model to be ns3::ConstantPositionMoblityModel to
fixed the position of the devices */
mobility.SetMobilityModel("ns3::ConstantPositionMobilityModel");
mobility.Install(nodes);
```



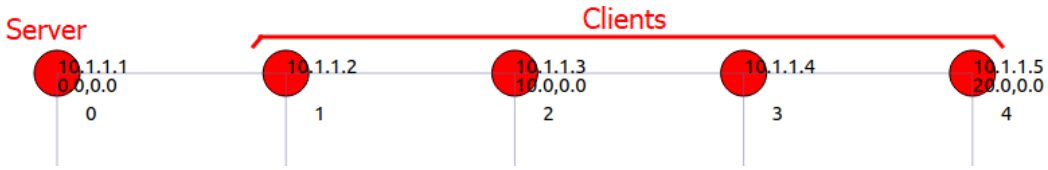
Visualization of the grid with 7 nodes in netanim

- Install Internet Stack

```
// Use InternetStackHelper to install protocol stacks
InternetStackHelper stack;
stack.Install(nodes);

// Use Ipv4AddressHelper to assign IP addresses to our device interfaces
Ipv4AddressHelper address;
// Use the network 10.1.1.0 to create the addresses needed for our devices
address.SetBase("10.1.1.0", "255.255.255.0");
/* Save the created interfaces in a container to make it easy to pull out
addressing information later for use in setting up the applications */
Ipv4InterfaceContainer nodeInterfaces;
nodeInterfaces = address.Assign(devices);
```

3. Setup applications

- Design:



  Example of a design with 5 nodes

  - 1 node is a server (default is the first node)
  - The rest are clients
  - All clients will simultaneously send packets to the server

- UdpEchoServer

```
// Create a UdpEchoServerHelper and provide the server port number
UdpEchoServerHelper echoServer(9);

// Instantiate the server on the choosen server node
ApplicationContainer serverApps = echoServer.Install(nodes.Get(serverNode));
serverApps.Start(Seconds(2.0));
serverApps.Stop(Seconds(simTime));
```

- UdpEchoClient

```
// Create a UdpEchoClientHelper and provide the remote address and port
UdpEchoClientHelper echoClient(nodeInterfaces.GetAddress(serverNode), 9);
echoClient.SetAttribute("MaxPackets", UintegerValue(maxPackets)); // Default is 10
echoClient.SetAttribute("Interval", TimeValue(Seconds(interval)));// Default is 1s
echoClient.SetAttribute("PacketSize", UintegerValue(packetSize)); // Default is 512B

// Install the client on every other node except the serverNode
for (uint32_t i = 0; i < nNodes; i++)
{
  if (i == serverNode) continue; // Exclude the server node
  ApplicationContainer clientApp = echoClient.Install(nodes.Get(i));
  clientApp.Start(Seconds(2.0));
  clientApp.Stop(Seconds(simTime));
}
```

4. Collect data

- Pcap

```
// Enable pcap
if (pcap)
{
  phy.EnablePcap("final", devices.Get(serverNode), true);
}
```

- Flow Monitor

  ⇒ Our main method to collect data

```
// Install flowMonitor to collect data
Ptr<FlowMonitor> flowMonitor;
FlowMonitorHelper flowHelper;
flowMonitor = flowHelper.InstallAll();
```

```
// Save the result of flow monitor in an xml file
char filename[100];
sprintf(filename, "final-%d-nodes.xml", nNodes);
flowMonitor->SerializeToXmlFile(filename, true, true);
```

- Collect data for visualization

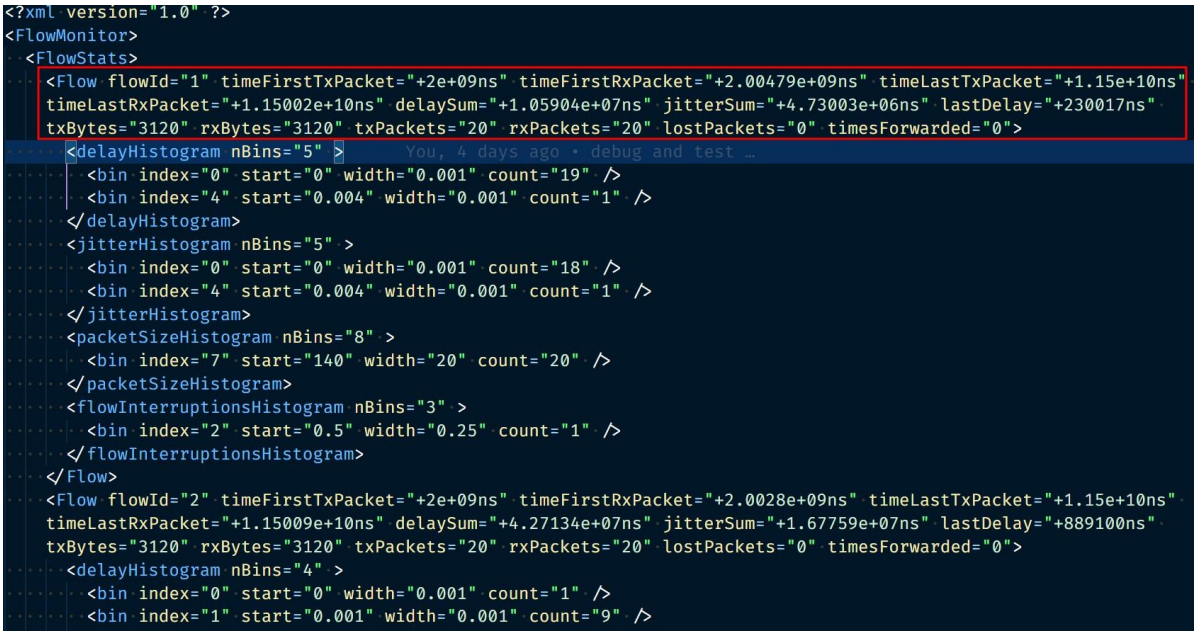
```
// Use netanim to visualize the experiment and save the result in an xml file
char animfile[100];
sprintf(animfile, "anim-%d-nodes.xml", nNodes);
AnimationInterface anim(animfile);
```

5. Analyze data

- Data from Flow Monitor
  - Data are stored in flows. Each flows contains all data about packets sent by a particular host to another.

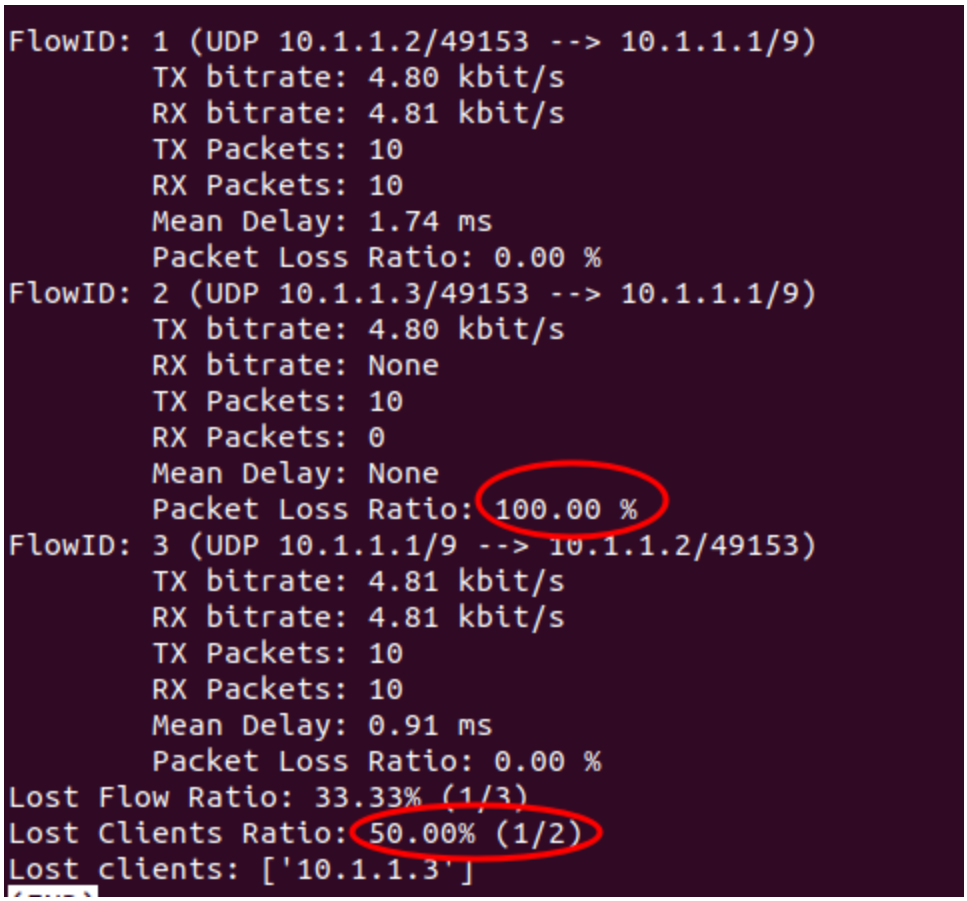
```
<Flow flowId="1" sourceAddress="10.1.1.2" destinationAddress="10.1.1.1" protocol="17" sourcePort="49153"
destinationPort="9">
```

- Data from the flows contains a lot of interesting information such as
  - The time the first and last packet is transmitted and received
  - The total delay
  - The total bytes and packets transmitted and received
  - Number of lost packets



```
<?xml version="1.0" ?>
<FlowMonitor>
  <FlowStats>
    <Flow flowId="1" timeFirstTxPacket="+2e+09ns" timeFirstRxPacket="+2.00479e+09ns" timeLastTxPacket="+1.15e+10ns"
    timeLastRxPacket="+1.15002e+10ns" delaySum="+1.05904e+07ns" jitterSum="+4.73003e+06ns" lastDelay="+230017ns"
    txBytes="3120" rxBytes="3120" txPackets="20" rxPackets="20" lostPackets="0" timesForwarded="0">
      <delayHistogram nBins="5" >        You, 4 days ago • debug and test …
        <bin index="0" start="0" width="0.001" count="19" />
        <bin index="4" start="0.004" width="0.001" count="1" />
      </delayHistogram>
      <jitterHistogram nBins="5" >
        <bin index="0" start="0" width="0.001" count="18" />
        <bin index="4" start="0.004" width="0.001" count="1" />
      </jitterHistogram>
      <packetSizeHistogram nBins="8" >
        <bin index="7" start="140" width="20" count="20" />
      </packetSizeHistogram>
      <flowInterruptionsHistogram nBins="3" >
        <bin index="2" start="0.5" width="0.25" count="1" />
      </flowInterruptionsHistogram>
    </Flow>
    <Flow flowId="2" timeFirstTxPacket="+2e+09ns" timeFirstRxPacket="+2.0028e+09ns" timeLastTxPacket="+1.15e+10ns"
    timeLastRxPacket="+1.15009e+10ns" delaySum="+4.27134e+07ns" jitterSum="+1.67759e+07ns" lastDelay="+889100ns"
    txBytes="3120" rxBytes="3120" txPackets="20" rxPackets="20" lostPackets="0" timesForwarded="0">
      <delayHistogram nBins="4" >
        <bin index="0" start="0" width="0.001" count="1" />
        <bin index="1" start="0.001" width="0.001" count="9" />
```

- Written a python script to parse the XML file and extract important data
  - An example of data with 3 nodes

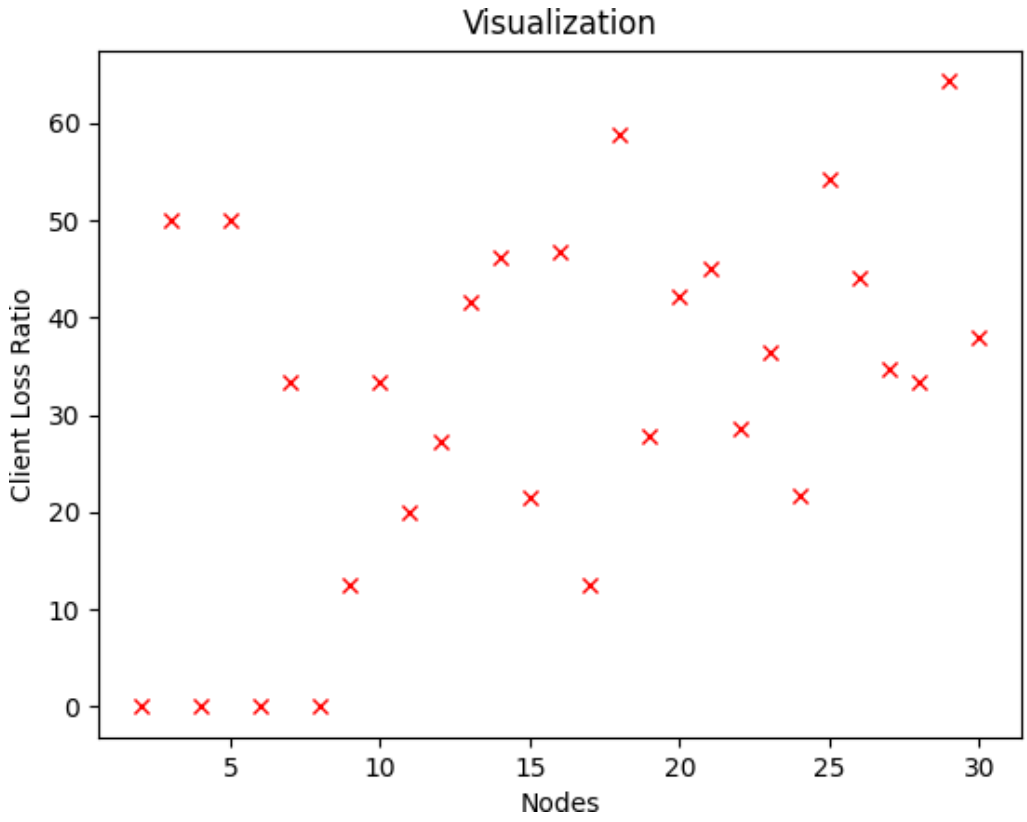


  - All packets sent by some clients are completely lost

- Summarized the ratio of lost clients when total of nodes range from 2 to 30

```
0.00 50.00 0.00 50.00 0.00 33.33 0.00 12.50 33.33 20.00 27.27 41.67 46.15 21.43 46.67 12.50 58.82 27.78 4
2.11 45.00 28.57 36.36 21.74 54.17 44.00 34.62 33.33 64.29 37.93
```

- Used matplotlib to visualize the lost clients ratio

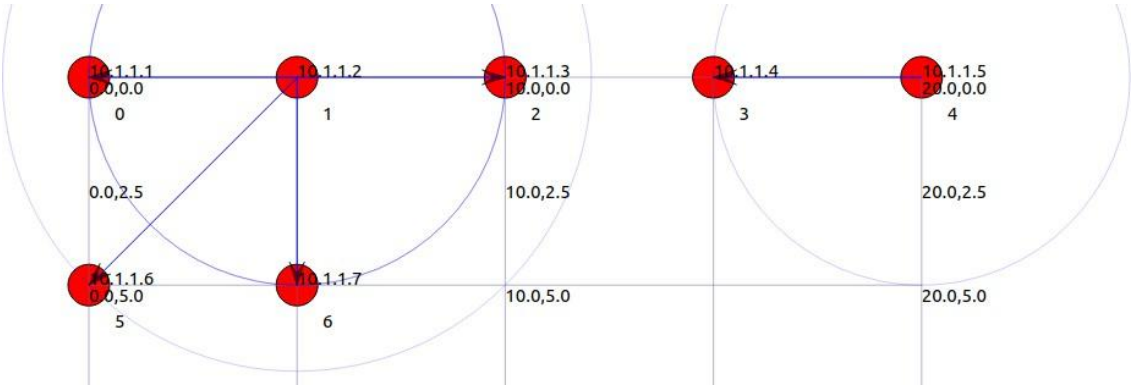


- When the number of nodes increases, the ratio of lost client also increases

6. Explain the result

- Our theory:

  In ad-hoc wireless network where all devices are connected to each other, if we disable RTS/CTS, data collision is more likely to happen. Therefore when the number of nodes is increased, the number of clients and the number of packets being transmitted is also increased. More collision happens and more packets are lost.



7. Code implementation

- All implementation will be stored in a repository on GitHub at:

  https://github.com/sblprateek05/ns3-csma-ca-simulation

8. References

- Configure ad-hoc mode
  - https://www.nsnam.org/docs/release/3.19/doxygen/wifi-simple-adhoc_8cc.html
  - Infrastructure vs Ad-hoc mode

    ⇒ In infrastructure mode there are beacon frames sent by the AP and in the Ad-hoc mode, there aren't any
    - Code example
- Disable RTS/CTS
  - https://ns-3-users.narkive.com/LFdwIHU6/disable-rts-cts