International Journal of Engineering & Technology, 7 (4.15) (2018) 318-322

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research paper

Performance Evaluation of Distributed Multi Media Wireless Sensor Network

Tehmina Karamat Khan[1]*, Zulkefli Bin Muhammad Yusof[2], Asadullah Shah[3]

International Islamic University Malaysia
**Corresponding author E-mail: zulmy@iium.edu.my*

Abstract

The demand for multimedia services i.e. audio, video and data with improve QoS and optimum utilization of resources in WSN's has posed new challenges. As the intensity of traffic increases; it demands for higher bandwidth and dedicated resources to reduce packet loss and delay. There have been analytical models proposed where priorities were assigned to video and voice packets to reduce packet loss and optimize resource utilization. In this paper distributed scheme is proposed to handle video, voice and data packets by having multiple sink nodes. There are shared sink nodes where video, voice and data packets are serviced and dedicated sink nodes only for video and voice packets. The proposed scheme has shown that the packet loss for data packets is higher than voice and video packets. The simulation results show that the performance of the network is improved when priorities are assigned to video and voice packets by giving dedicated resources.

Keywords: *WSN, Packet Loss, Resource Utilization, Distributed Priority Scheme, Network Performance.*

1. Introduction

WSN is the network of remote motes also called nodes deployed for sensing environmental condition, motion, Audio or Video where human interaction is negligible where it operates autonomously. These sensors collect the information of special environmental condition or data of interest, Multimedia node collects the Video, Audio or both and transfer the collected Information to a base entity called the sink, unlike the motes BS is generally more powerful with rich resources, and where this information can be analyzed or transmitted via the Internet. WMSNs has a large scale of applications in the multiple Fields such as target-tracking, environmental-monitoring, system-control, health-monitoring or exploration in remote environment. Recently, the rapid development in the field of WMSNs has increase the expectation where its applications such as the analysis of spectral-density, pictorial-information with simultaneously transmissions, a maximum throughput and a higher data delivery rate (Multimedia Sensor Networks: Recent Trends, Research Challenges and Future Directions 2017 International Conference on Communication, Computing and Digital Systems (C-CODE))

2. Background

In [] analytical solution was devised for WSN where packets arrive in burst, get service and depart the service facility in burst. The proposed priority model was compared with the non-priority model and it has showed that the priority scheme has significantly increased the performance of the system by decreasing the blocking of packets and service time of packets.

3. Proposed model

We have proposed a model in which we have assigned priority to video and voice packets over data packets. There are three packet generation nodes i.e. Node 0 generates video packets at 10Mbps with packet size of 256 bytes, video packets get processed at base station 0, 1 and 2. Node 0 generates video packets with link capacity of 10Mbps and packet size of 256 bytes, Node 1 generates voice packets at 5Mbps with packet size of 128 bytes and Node 2 generates data packets at 5Mbps rate with packet size of 64 bytes, as given in Table 1.a. There are three service stations ST-0, ST-1 and ST-2 for processing video, voice and data packets. ST-2 is dedicated for video packets while ST-0 and ST-1 are shared service stations. Video packets get processed at all service stations whereas voice packets are only processed at ST-0 and ST-1 while data packets are processed only at ST-0, all voice and data packets when arrived for service and finds service centre busy processing packets are lost and these lost packets never return to the system. Video packets when arrive at service facility and find it busy wait in queue. The scenario is shown in figure a.

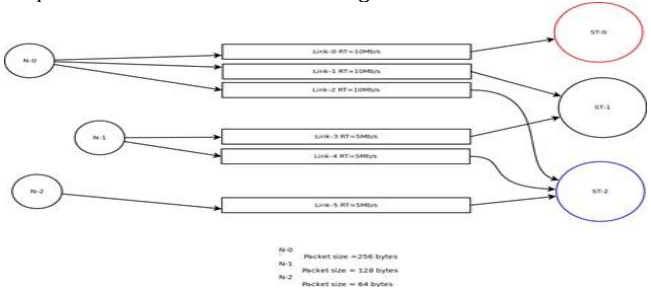


Fig. a:

Copyright © 2018 Authors. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Table 1 (a): Parameters details

Client Node	Traffic type	Packet size	Data rate	ns3::DropTailQueue size	Priority	Base station	Max-Packets
N-0	Video	256 bytes	10 Mbps	5000	1	ST-0, ST-1, ST-2	5000
N-1	Audio	128 Mbps	5 Mbps	5000/2	2	ST-1, ST-2	5000
N-2	data	64 Mbps	5 Mbps	(5000/2)/2	3	ST-2	5000

3.1. Term based method (TBM)

In ns-3, the abstraction of the computing device is called Node; the abstraction is represented in C++ by the Node class, which permits adding functionalities, like applications, protocol stacks, and peripheral cards with their associated drivers. A node can be connected through an object representing a communication cannel; its abstraction is called Channel and it is represented in C++ by the class of the same name. NetDevices are the network device abstraction that involve the software controller and simulated hardware. A network device is installed to enable communication among nodes through the Channels. A node can be connected to more than one channel through multiple network devices. The abstraction of the network device is represented in C++ by the class of the same name. Topology helpers permit assigning physical addresses,
installing network devices in a node, configuring the node's protocol stack, and then connecting the netdevices to the channel. Through the Node Container, node objects are created within ns-3 that will represent the computers in the simulation (Fig 1).

```
uint32_t N = 3;
NodeContainer serverNode;
NodeContainer clientNodes;
serverNode.Create (N);
clientNodes.Create (N);
```

Fig. 1: Node Container, Node creation

In first line 'N' represent number of nodes to be created. The second and third line only declares a node container called serverNodes and clientNodes respectively. The fourth and fifth line calls the Create method in the node objects and asks the container to create N nodes. The following step in the construction of a topology is to connect the nodes within the network. For example, to construct a point-to- point link, use the topology helpers object and within it use a PointToPointHelper to configure and connect the point-to- point network device objects PointToPointNetDevice) and point-to-point channel helpers (PontToPointHelperChannel), as shown in Fig.2

```
void Linksettig()
{
    PointToPointHelper p2p1,p2p2,p2p3,p2p4,p2p5,p2p6;
    p2p1.SetDeviceAttribute ("DataRate", DataRateValue (DataRate ("10Mbps")));
    p2p1.SetChannelAttribute ("Delay",   StringValue ("2ms"));
    p2p1.SetQueue("ns3::DropTailQueue",  "MaxPackets",UintegerValue("6Mbps"));
    p2p2.SetDeviceAttribute ("DataRate", DataRateValue (DataRate (RateA)));
    p2p2.SetChannelAttribute ("Delay",   StringValue ("2ms"));
    p2p2.SetQueue("ns3::DropTailQueue",  "MaxPackets",UintegerValue(1000/2));
    p2p3.SetDeviceAttribute ("DataRate", DataRateValue (DataRate ("3Mbps")));
    p2p3.SetChannelAttribute ("Delay",   StringValue ("2ms"));
    p2p3.SetQueue("ns3::DropTailQueue",  "MaxPackets",UintegerValue((1000/2)/2));
}
```

Fig. 2: Create and assign attributes to point-to-point links

Next, it is necessary to have a list of the objects, NetDevice, for which a NetDeviceContainer is used to manage the task. Fig 3 presents the final configuration of the devices and the channel.

```
std::vector<NetDeviceContainer> deviceAdjacencyList (7);
for(uint32_t i=0 ;i<=deviceAdjacencyList.size();i++)
{
    deviceAdjacencyList[i] = p2p1.Install (nodeAdjacencyList[i]);
}
```

Fig.3: Configuration of devices and channel

The first line declares the device container and the third line of the install method of the point-to-point link helper takes a NodeContainer as parameter. After executing the pointTopoint.Install call, we will have six nodes, in which three nodes are considered as clients(source) nodes and remaining three are considered as sink(destination) nodes each with a netdevice and a channel between them.

```
NodeContainer allNodes = NodeContainer (serverNode, clientNodes);
InternetStackHelper internet;
internet.Install (allNodes);
```

Fig. 4: Installation of internet stack or protocol in server, and client nodes

The following step installs the protocol stack in the nodes (Fig.4). The helper is the InternetStackHelper, in charge of installing the protocol stack in the nodes. The install method takes a node container object as parameter; when it is executed, it installs the Internet protocols in each of the nodes. Next, we need to associate the node devices to the IP addresses. A helper exists to manage the assignment of IP addresses. Fig. 5

```
Ipv4AddressHelper ipv4;
std::vector<Ipv4InterfaceContainer> interfaceAdjacencyList (7);
for(uint32_t i=0; i<nodeAdjacencyList.size (); ++i)
{
    std::ostringstream subnet;
    subnet<<"10.1."<<i+1<<".0";
    ipv4.SetBase (subnet.str ().c_str (), "255.255.255.0");
    interfaceAdjacencyList[i] = ipv4.Assign (deviceAdjacencyList[i]);
}
```

Fig. 5: Assignment of IP addresses

The first line of code declares an object helper of addresses. By default, the addresses assigned will start at one and will increase one by one. The NS-3 system at low level remembers all the addresses assigned and will generate an error if by accident the same address is assigned twice. The Ipv4InterfaceContainer class makes associations between IP addresses and the devices using an Ipv4InterfaceContainer object. For data to be received in the client nodes it is necessary to adhere a sink application. In present scenario we have three sink nodes so we have to assign sink properties to three nodes as shown in Fig 6.

```
uint16_t port = 50000;
uint16_t port1 = 50002;
uint16_t port2 = 50001;
Address sink1_LocalAddress1 (InetSocketAddress (Ipv4Address::GetAny (), port));
PacketSinkHelper sink_1_Helper1 ("ns3::UdpSocketFactory", sink1_LocalAddress1);
ApplicationContainer sinkApp;
sinkApp = sink_1_Helper1.Install (serverNode.Get(0));
sinkApp.Start (Seconds (0.5));
sinkApp.Stop (Seconds (100.0));
Address sink2_LocalAddress1 (InetSocketAddress (Ipv4Address::GetAny (), port1));
PacketSinkHelper sink_2_Helper1 ("ns3::UdpSocketFactory", sink2_LocalAddress1);
ApplicationContainer sinkApp2;
sinkApp2 = sink_2_Helper1.Install (serverNode.Get(1));
sinkApp2.Start (Seconds (0.5));
sinkApp2.Stop (Seconds (100.0));
Address sink3_LocalAddress1 (InetSocketAddress (Ipv4Address::GetAny (), port2));
PacketSinkHelper sink_3_Helper1 ("ns3::UdpSocketFactory", sink3_LocalAddress1);
ApplicationContainer sinkApp3;
sinkApp3 = sink_3_Helper1.Install (serverNode.Get(2));
sinkApp3.Start (Seconds (0.5));
sinkApp3.Stop (Seconds (100.0));
```

Fig. 6: Installation of sink app on servers

The first three lines of code define different ports at witch sink nodes will listen to incoming traffic we have assigned different port to each sink to overcome traffic conflict from clients. The Sink Helper, through its Set Attribute method, can configure the client nodes to receive data. The Local attribute refers to the node where the data is received, and the following parameter of the method is a socket.
Thereafter, an application is taken from the application container and it is installed in the node it will receive. Finally, the time is set, in which the node is ready to receive data.
To watching simulations nodes, need to be at proper positions, so we have to allocate constant positions to each node using Constant Position Mobility Model as shown in Fig. 7

```
MobilityHelper mobility;
Ptr<ListPositionAllocator> positionAlloc = CreateObject<ListPositionAllocator> ();
positionAlloc->Add (Vector (0.0, 0.0, 0.0));
positionAlloc->Add (Vector (1.0, 0.0, 0.0));
mobility.SetPositionAllocator (positionAlloc);
mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
mobility.Install (serverNode);
mobility.Install (clientNodes);
```

Fig.7: Install mobility helper

Now, we have a point-to-point network constructed with protocol stack and IP addresses assigned. At this point, the application needs to generate traffic. In order to generate traffic the following method is called. Fig 8

```
void TrafficGenerator(int pktsize,int maxpkt,float interval,float startat,float stospat,
       int pnum,Ipv4InterfaceContainer interfaces,int intfAdd,NodeContainer nodes,uint16_t port )
{
    UdpEchoClientHelper echoClient (interfaces.GetAddress (intfAdd), port);
    echoClient.SetAttribute ("MaxPackets", UintegerValue (maxpkt));
    echoClient.SetAttribute ("Interval", TimeValue (Seconds (interval)));
    echoClient.SetAttribute ("PacketSize", UintegerValue (pktsize));
    ApplicationContainer clientApps = echoClient.Install (nodes.Get (pnum));
    clientApps.Start (Seconds (startat));
    clientApps.Stop (Seconds (stospat));
}
```

Fig.8: Traffic generator

To calculate throughput ThroughputMonitor method is called every 1 second as in the following Fig. 9

```
void ThroughputMonitor (FlowMonitorHelper *fmhelper, Ptr<FlowMonitor> flowMon)
{
    map<FlowId, FlowMonitor::FlowStats> flowStats = flowMon->GetFlowStats();
    Ptr<Ipv4FlowClassifier> classing = DynamicCast<Ipv4FlowClassifier> (fmhelper->GetClassifier());
    for (std::map<FlowId, FlowMonitor::FlowStats>::const_iterator stats = flowStats.begin (); stats != flowStats.e
    {
      Ipv4FlowClassifier::FiveTuple fiveTuple = classing->FindFlow (stats->first);
      cout<<"Flow ID            : " << stats->first <<" ; "<< fiveTuple.sourceAddress <<" -----> "<<fiveTuple.dest
      cout<<"Tx Packets = " << stats->second.txPackets<<std::endl;
      cout<<"Rx Packets = " << stats->second.rxPackets<<std::endl;
      cout<<"Duration       : "<<stats->second.timeLastRxPacket.GetSeconds()-stats->second.timeFirstTxPacket.GetSe
      cout<<"Last Received Packet   : "<< stats->second.timeLastRxPacket.GetSeconds()<<" Seconds"<<std::endl;
      cout<<"Throughput: " << stats->second.rxBytes * 8.0 / (stats->second.timeLastRxPacket.GetSeconds()-stats->se
      cout<<"---------------------------------------------------------------------------"<<std::endl;
      }
        Simulator::Schedule(Seconds(0.1),&ThroughputMonitor, fmhelper, flowMon);
}
```

Fig.9: Throughput monitor

To run the simulation, use the Simulator: Run global function (Fig.11).

```
FlowMonitorHelper fmHelper;
Ptr<FlowMonitor> allMon = fmHelper.InstallAll();
Simulator::Schedule(Seconds(1),&ThroughputMonitor,&fmHelper, allMon,dataset);
```

Fig 10: Creating Flow monitor

```
Simulator::Run ();
```

Fig. 11: Run Simulation

When this function is called, the system will start searching through the list of events programmed and will execute them. Finally, the cleaning process must be conducted through the global Simulator: Destroy function (Fig. 12).

```
Simulator::Destroy ();
```

Fig. 12: Destroy function

To work according to the ns-3 guidelines, it is necessary to copy the file in the scratch directory where it is compiled through the . /waf command and it is run through ./waf - -run scratch/MyFile without .cc extension.

4. Analysis and Results

The statistics obtained through the ns-3 FlowMonitor application are total values along the simulation. These are: bytes transmitted and received, packets transmitted received and lost. Fig. 9 shows the configuration of these statistics. In addition, the throughput as mean value are also obtained. Flow monitor that must be installed in the network previously created.

4.1. Getting the throughput

Throughput is obtained by knowing three parameters. The first is the time in which the first packet was transmitted; the second is the time in which the last packet was received; and the third was the number of packets received. Two TraceSources exist in the network device; these are useful to obtain throughput through samples. Fig. 9 presents the code used to measure throughput in the network. For the first packet through PhyTxBegin the First-Packet (TraceSink) function is called whenever it is transmitted by the channel; said function registers the time in which it is transmitted. For the second packet through PhyRxEnd, the LastPacket (TraceSink) function is called each time it is received in the client node. Thus, this function registers the time in which it is received and registers the sum of packets received to that moment in the client node. These two instructions register the throughput, for example, within a switch node for a client node connected to it.

4.2. Analysis of proposed scheme by throughput

According to our proposed analytical model where priorities were assigned to video and voice-packets to reduce packet loss and optimize resource utilization. The following result shows highest throughput for video packets, as video packets are serviced in all dedicated servers and it has higher priority over other type of packets that is voice and data packets. The black line shows throughput for voice packets with second highest priority. And the last blue line shows data packets with lowest priority.

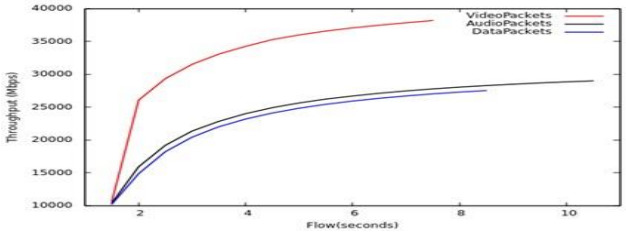


Fig. 13: Throughput

It is observed that our proposed scheme improves QoS and optimum utilization of resources. The following Fig 14 also shows the throughput result as bar chart.

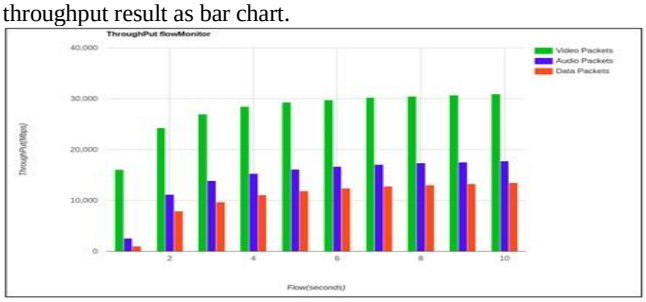


Fig. 14: Throughput chart

4.3. Obtaining delay results

End-to-end delay is the time taken for a packet to be transmitted across a network from source to destination. Delay is obtained by subtracting packet send time from packet received time. In our model for improved QoS we need highest throughput for video traffic transmission but lowest delay time, In Fig 16 graph shows the expected delay time results. Initially video traffic have some delay time but it is very low as compared to delay time of audio and data traffic. After some milliseconds delay time reaches to stable position and remains constant as illustrated in the following graph. In NS-3 delay time can be obtained by calling Delay-Time() method as shown in Fig 9. Delay time is in milliseconds as shown at y-axis and 'flow' is time in seconds at x-axis in Fig 16.

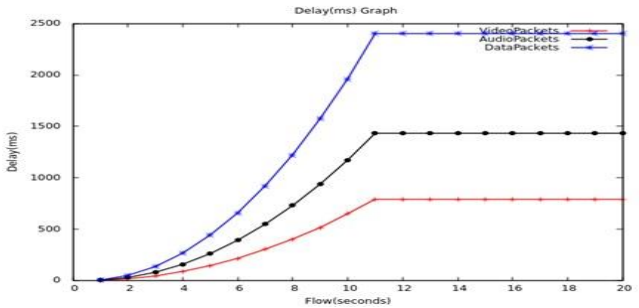


Fig. 15: Delay Time Result

We have allotted video traffic with high priority to insure improved QoS of traffic transmission so in the following chart green bars represents lowest delay time for video traffic. Blue bars represent delay time of audio transmission with second highest priority and last red bar shows high delay time of data packets as it is assigned with lowest priority.

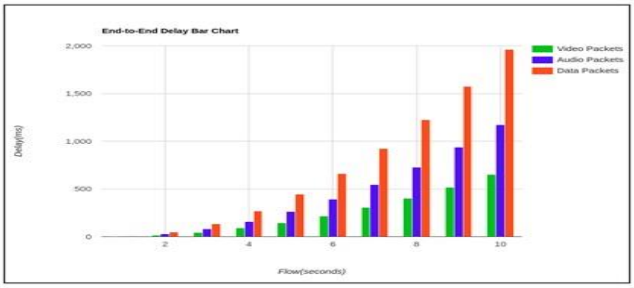


Fig. 16: Delay Time Result

Table 1 (b): Performance comparison for video, voice and data packets

Flow Id:1	Flow Id:2	Flow Id:3
UDP 10.1.1.2/49153 ----> 10.1.1.1/50000	UDP 10.1.2.2/49153 ----> 10.1.2.1/50000	UDP 10.1.3.2/49153 ----> 10.1.3.1/50000
Tx bitrate:-1.136e+13kbps	Tx bitrate:-6.24e+12kbps	Tx bitrate:-3.68e+12kbps
Rx bitrate:9932.55kbps	Rx bitrate:4938.35kbps	Rx bitrate:3916.85kbps
Mean delay:459.829ms	Mean delay:381.453ms	Mean delay:190.188ms
Packet Loss ratio:19.98%	Packet Loss ratio:39.98%	Packet Loss ratio:59.98%
timeFirstTxPacket= 1e+09ns	timeFirstTxPacket= 1e+09ns	timeFirstTxPacket= 1e+09ns
timeFirstRxPacket= 1.00223e+09ns	timeFirstRxPacket= 1.00225e+09ns	timeFirstRxPacket= 1.00219e+09ns
timeLastTxPacket= 1e+09ns	timeLastTxPacket= 1e+09ns	timeLastTxPacket= 1e+09ns
timeLastRxPacket= 1.01743e+09ns	timeLastRxPacket= 1.76065e+09ns	timeLastRxPacket= 1.37819e+09ns
jitterSum= 9.152e+08ns	jitterSum= 7.584e+08ns	jitterSum= 3.76e+08ns
lastDelay= 1.83978e+12ns	lastDelay= 1.14474e+12ns	lastDelay= 3.80566e+11ns
txBytes= 1420000	txBytes= 780000	txBytes= 460000
rxBytes= 1136284	rxBytes= 468156	rxBytes= 184092
txPackets= 5000	txPackets= 5000	txPackets= 5000
rxPackets= 4001	rxPackets= 3001	rxPackets= 2001
lostPackets= 999	lostPackets= 1999	lostPackets= 2999
timesForwarded= 0	timesForwarded= 0	timesForwarded= 0
Packets Dropped:	Packets Dropped:	Packets Dropped:
No Route:0	No Route:0	No Route:0
TTL Expire:0	TTL Expire:0	TTL Expire:0
Bad Checksum:0	Bad Checksum:0	Bad Checksum:0
Queue:0	Queue:0	Queue:0
Interface Down:999	Interface Down:1999	Interface Down:2999
Bytes Dropped:	Bytes Dropped:	Bytes Dropped:
No Route:0	No Route:0	No Route:0
TTL Expire:0	TTL Expire:0	TTL Expire:0
Bad Checksum:0	Bad Checksum:0	Bad Checksum:0
Queue:0	Queue:0	Queue:0
Interface Down:283716	Interface Down:311844	Interface Down:275908
delayHistogram nBins:918	delayHistogram nBins:761	delayHistogram nBins:379
Index:2 Start:0.002 Width:0.001 Count:4	Index:2 Start:0.002 Width:0.001 Count:3	Index:2 Start:0.002 Width:0.001 Count:5
Index:3 Start:0.003 Width:0.001 Count:4	Index:3 Start:0.003 Width:0.001 Count:4	Index:3 Start:0.003 Width:0.001 Count:5
Index:4 Start:0.004 Width:0.001 Count:5	Index:4 Start:0.004 Width:0.001 Count:4	Index:4 Start:0.004 Width:0.001 Count:5
Index:5 Start:0.005 Width:0.001 Count:4	Index:5 Start:0.005 Width:0.001 Count:4	Index:5 Start:0.005 Width:0.001 Count:6
Index:6 Start:0.006 Width:0.001 Count:4	Index:6 Start:0.006 Width:0.001 Count:4	Index:6 Start:0.006 Width:0.001 Count:5
Index:7 Start:0.007 Width:0.001 Count:5	Index:7 Start:0.007 Width:0.001 Count:4	Index:7 Start:0.007 Width:0.001 Count:5
Index:8 Start:0.008 Width:0.001 Count:4	Index:8 Start:0.008 Width:0.001 Count:4	Index:8 Start:0.008 Width:0.001 Count:6
Index:9 Start:0.009 Width:0.001 Count:5	Index:9 Start:0.009 Width:0.001 Count:4	Index:9 Start:0.009 Width:0.001 Count:5
Index:10 Start:0.01 Width:0.001 Count:4	Index:10 Start:0.01 Width:0.001 Count:4	Index:10 Start:0.01 Width:0.001 Count:5
Index:11 Start:0.011 Width:0.001 Count:4	Index:11 Start:0.011 Width:0.001 Count:4	Index:11 Start:0.011 Width:0.001 Count:6

Table 1(c): Throughput of video, voice and data packets

```
ns-3@ns3-HP-EliteBook-8440p: /media/ns-3/01D31F438A73EF80/MULTIBOOT/ns-3.27
File  Edit  View  Search  Terminal  Help
Tx Packets = 5000
Rx Packets = 4001
Duration                : 0.917429
Last Received Packet    : 1.91743 Seconds
Throughput: 9.44941 Mbps
---------------------------------------------------------------------------
Flow ID                 : 2 ; 10.1.2.2 -----> 10.1.2.1
Tx Packets = 5000
Rx Packets = 3001
Duration                : 0.760653
Last Received Packet    : 1.76065 Seconds
Throughput: 4.69563 Mbps
---------------------------------------------------------------------------
Flow ID                 : 3 ; 10.1.3.2 -----> 10.1.3.1
Tx Packets = 5000
Rx Packets = 2001
Duration                : 0.378188
Last Received Packet    : 1.37819 Seconds
Throughput: 3.71379 Mbps
---------------------------------------------------------------------------
F-id :1  Throughput  = 0.819405
F-id :2  Throughput = 0.307302
F-id :3  Throughput = 0.102451
ns-3@ns3-HP-ElIteBook-8440p:/media/ns-3/01D31F438A73EF80/MULTIBOOT/ns-3.27$
```

5. Conclusion

There are some results in xml form which can be seen in Net Anim tool in which flow id 1 is for video packets, flow id 2 is for voice packets and flow id 3 represents data packets being sent from base nodes to sink nodes. From the below table 1.b it is observed that video packets loss, jitter and delay is less than voice and data packets though the rate of transmission of video packets is higher than the voice and data packets. The below table also highlights the effectiveness in terms of improved performance and resource utilization of priority scheme. Packet loss ratio for video, voice and data packets is 19.98%, 39.98 and 59.98%, packet loss is 999, 1999 and 2999 packets respectively. Jitter sum is $9.152e^{08}ns$, $7.584e^{08}ns$ and $3.76e^{08}ns$ for video, voice and data packets.

Throughput for video, voice and data packets is 9.44941, 4.69563 and 3.71379 respectively. It is observed that the throughput for video packets is higher than the voice and data packets, an indicator that shows that assigning dedicated channels improves throughput.

References

[1] Ahmed, S. F., Azim, C. F., & Memon, A. R. (2011). Minimization of the excessive noise for broad-band active noise feed-forward control system. International Journal of Academic Research, 3(2), 1034-1039.

[2] Ahmed, S. F., Ali, A., Joyo, M. K., Rehan, M., Siddiqui, F. A., Bhatti, J. A., ... & Dezfouli, M. M. S. (2018, May). Mobility assistance robot for disabled persons using electromyography (EMG) sensor. In Innovative Research and Development (ICIRD), 2018 IEEE International Conference on (pp. 1-5). IEEE.

[3] Q. Minhas, H. Mahmood, and H. Malik, "The Role of Ad Hoc Networks in Mobile Telecommunication," 2009.

[4] A. Bentaleb, A. Boubetra, and S. Harous, "Survey of Clustering Schemes in Mobile Ad hoc Networks," Commun. Netw., vol. 5, no. May, pp. 8–14, 2013.

[5] Ahmed, S. F., Ali, A., Joyo, M. K., Rehan, M., Siddiqui, F. A., Bhatti, J. A., ... & Dezfouli, M. M. S. (2018, May). Mobility assistance robot for disabled persons using electromyography (EMG) sensor. In Innovative Research and Development (ICIRD), 2018 IEEE International Conference on (pp. 1-5). IEEE.

[6] Altaf, S., Shah, A., Imtiaz, N., Shah, A.S., Ahmed, S.F., "Visualization representing benefits of pre-requirement specification traceability" International Journal of Engineering and Technology(UAE) 7 (2.5), 44-52.

[7] S. Kaur and V. Kumari, "A Proposed Model for Load Balancing in MANET," vol. 97, no. 15, pp. 24–26, 2014.

[8] Ahmed, S.F., Banky, G., Blicblau, A., Joyo, M.K., "Augmented reality with Haptic technology based online experimental based distance learning education technique," 2016 AIP Conference Proceedings L. Chang, W. Yafeng, H. Fan, and Y. Dacheng, "A novel enhanced weighted clustering algorithm for mobile networks," Proc. - 5th Int. Conf. Wirel. Commun. Netw. Mob. Comput. WiCOM 2009, no. 2007, pp. 1–4, 2009.

[9] C. Johnen and L. H. Nguyen, "Robust self-stabilizing weight-based clustering algorithm," Theor. Comput. Sci., vol. 410, no. 6–7, pp. 581–594, 2009.

[10] A. Hussein, S. Yousef, and O. Arabiyat, "A Load-Balancing and Weighted Clustering Algorithm in Mobile Ad-Hoc Network," Kaspersky.Co.Za, 2009.

[11] A. Hussein, S. Yousef, S. Al-Khayatt, and O. S. Arabeyyat, "An efficient weighted distributed clustering algorithm for mobile ad hoc networks," 2010 Int. Conf. Comput. Eng. Syst., no. C, pp. 221–228, 2010.