SOURCE CODE

6.1 Test Cases in public channel

1. Public channel implementation (first.cc)

- Source code

```
#include "ns3/core-module.h"
 #include "ns3/network-module.h"
 #include "ns3/internet-module.h"
 #include "ns3/point-to-point-module.h"
 #include "ns3/applications-module.h"
 #include "ns3/netanim-module.h"

 using namespace ns3;
 NS_LOG_COMPONENT_DEFINE
("FirstScriptExample");

 int
 main (int argc, char *argv[])
 {
  CommandLine cmd;
  cmd.Parse (argc, argv);

  Time::SetResolution (Time::NS);
  LogComponentEnable
("UdpEchoClientApplication",
LOG_LEVEL_INFO);
  LogComponentEnable
("UdpEchoServerApplication",
LOG_LEVEL_INFO);

  NodeContainer nodes;
  nodes.Create (2);

  PointToPointHelper pointToPoint;
  pointToPoint.SetDeviceAttribute
("DataRate", StringValue ("5Mbps"));
  pointToPoint.SetChannelAttribute ("Delay",
StringValue ("2ms"));

  NetDeviceContainer devices;
  devices = pointToPoint.Install (nodes);
  InternetStackHelper stack;
  stack.Install (nodes);
```

```
Ipv4AddressHelper address;
  address.SetBase ("10.1.1.0", "255.255.255.0");

  Ipv4InterfaceContainer interfaces =
address.Assign (devices);

  UdpEchoServerHelper echoServer (9);

  ApplicationContainer serverApps =
echoServer.Install (nodes.Get (1));
  serverApps.Start (Seconds (1.0));
  serverApps.Stop (Seconds (10.0));

  UdpEchoClientHelper echoClient
(interfaces.GetAddress (1), 9);
  echoClient.SetAttribute ("MaxPackets",
UintegerValue (1));
  echoClient.SetAttribute ("Interval",
TimeValue (Seconds (1.0)));
  echoClient.SetAttribute ("PacketSize",
UintegerValue (1024));

  ApplicationContainer clientApps =
echoClient.Install (nodes.Get (0));
  clientApps.Start (Seconds (2.0));
  clientApps.Stop (Seconds (10.0));

  AnimationInterface anim ("anim1.xml");
  anim.SetConstantPosition(nodes.Get(0), 1.0,
2.0);
  anim.SetConstantPosition(nodes.Get(1), 10.0,
10.0);

  Simulator::Run ();
  Simulator::Destroy ();
  return 0;
}
```

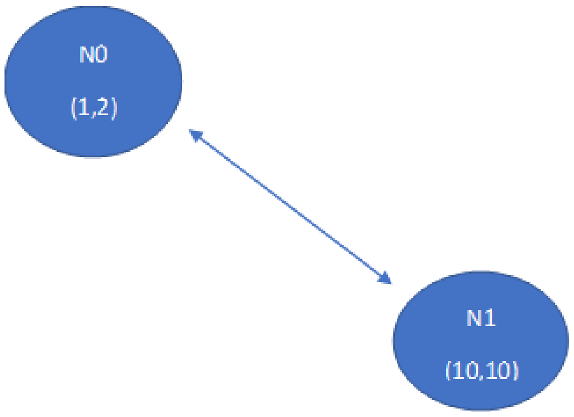
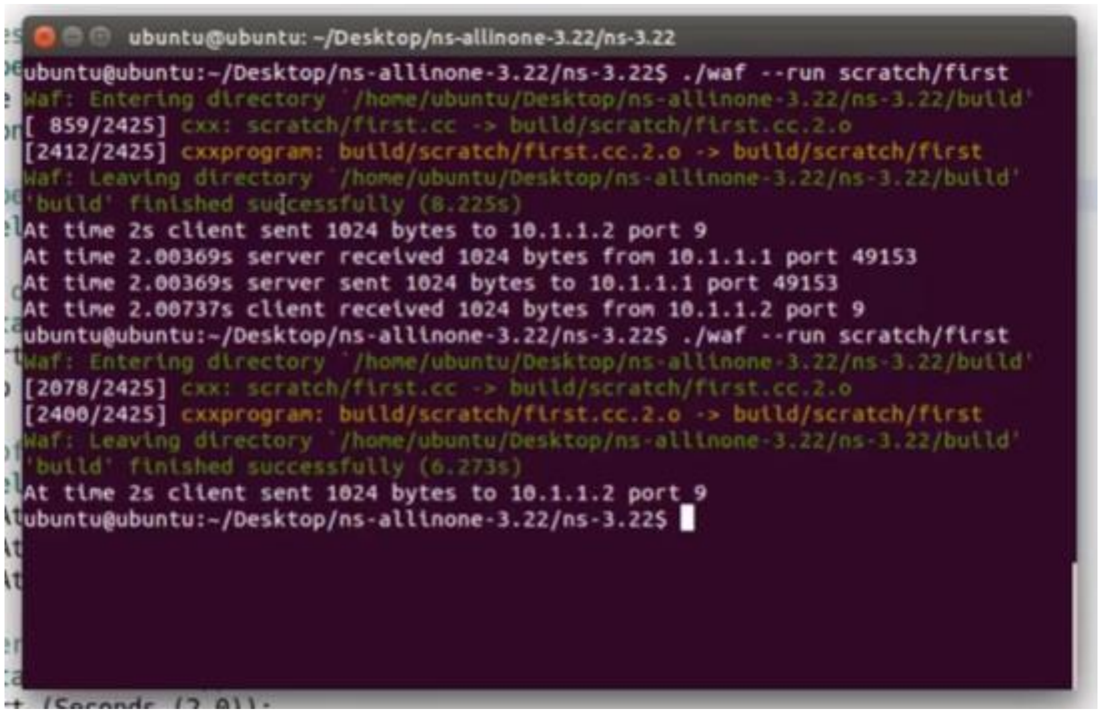


Fig.6: first.cc network

- <u>Output</u>



```
ubuntu@ubuntu: ~/Desktop/ns-allinone-3.22/ns-3.22
ubuntu@ubuntu:~/Desktop/ns-allinone-3.22/ns-3.22$ ./waf --run scratch/first
Waf: Entering directory `/home/ubuntu/Desktop/ns-allinone-3.22/ns-3.22/build'
[ 859/2425] cxx: scratch/first.cc -> build/scratch/first.cc.2.o
[2412/2425] cxxprogram: build/scratch/first.cc.2.o -> build/scratch/first
Waf: Leaving directory `/home/ubuntu/Desktop/ns-allinone-3.22/ns-3.22/build'
'build' finished successfully (8.225s)
At time 2s client sent 1024 bytes to 10.1.1.2 port 9
At time 2.00369s server received 1024 bytes from 10.1.1.1 port 49153
At time 2.00369s server sent 1024 bytes to 10.1.1.1 port 49153
At time 2.00737s client received 1024 bytes from 10.1.1.2 port 9
ubuntu@ubuntu:~/Desktop/ns-allinone-3.22/ns-3.22$ ./waf --run scratch/first
Waf: Entering directory `/home/ubuntu/Desktop/ns-allinone-3.22/ns-3.22/build'
[2078/2425] cxx: scratch/first.cc -> build/scratch/first.cc.2.o
[2400/2425] cxxprogram: build/scratch/first.cc.2.o -> build/scratch/first
Waf: Leaving directory `/home/ubuntu/Desktop/ns-allinone-3.22/ns-3.22/build'
'build' finished successfully (6.273s)
At time 2s client sent 1024 bytes to 10.1.1.2 port 9
ubuntu@ubuntu:~/Desktop/ns-allinone-3.22/ns-3.22$
```

Fig.7: Output for first.cc

<u>2. Public channel implementation with graphical output (seventh.cc)</u>

- <u>Source code</u>

```
#include <fstream>
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"
#include "ns3/stats-module.h"

using namespace ns3;

NS_LOG_COMPONENT_DEFINE
("SeventhScriptExample");

//
================================
//        node 0                node 1
//   +----------------+    +----------------+
//   |    ns-3 TCP    |    |    ns-3 TCP    |
//   +----------------+    +----------------+
//   |    10.1.1.1    |    |    10.1.1.2    |
//   +----------------+    +----------------+
//   | point-to-point |    | point-to-point |
//   +----------------+    +----------------+
//           |                     |
//           +---------------------+
//                5 Mbps, 2 ms
//
================================
//
class MyApp : public Application
{
public:
  MyApp ();
  virtual ~MyApp ();
/**
   * Register this type.
   * \return The TypeId.
   */
  static TypeId GetTypeId (void);
  void Setup (Ptr<Socket> socket, Address
address, uint32_t packetSize, uint32_t
nPackets, DataRate dataRate);
private:
  virtual void StartApplication (void);
virtual void StopApplication (void);
  void ScheduleTx (void);
  void SendPacket (void);
```

```
  Ptr<Socket>     m_socket;
  Address         m_peer;
  uint32_t        m_packetSize;
  uint32_t        m_nPackets;
  DataRate        m_dataRate;
  EventId         m_sendEvent;
  bool            m_running;
  uint32_t        m_packetsSent;
};

MyApp::MyApp ()
  : m_socket (0),
    m_peer (),
    m_packetSize (0),
    m_nPackets (0),
    m_dataRate (0),
    m_sendEvent (),
    m_running (false),
    m_packetsSent (0)
{
}
MyApp::~MyApp ()
{
  m_socket = 0;
}

/* static*/
TypeId MyApp::GetTypeId (void)
{
  static TypeId tid = TypeId ("MyApp")
    .SetParent<Application> ()
    .SetGroupName ("Tutorial")
    .AddConstructor<MyApp> ()
    ;
  return tid;
}

void
MyApp::Setup (Ptr<Socket> socket,
Address address, uint32_t packetSize,
uint32_t nPackets, DataRate dataRate)
{
  m_socket = socket;
  m_peer = address;
  m_packetSize = packetSize;
  m_nPackets = nPackets;
  m_dataRate = dataRate;
}
```

```
void
MyApp::StartApplication (void)
{
  m_running = true;
  m_packetsSent = 0;
  if (InetSocketAddress::IsMatchingType
(m_peer))
    {
      m_socket->Bind ();
    }
  else
    {
      m_socket->Bind6 ();
    }
  m_socket->Connect (m_peer);
  SendPacket ();
}

void
MyApp::StopApplication (void)
{
  m_running = false;

  if (m_sendEvent.IsRunning ())
    {
      Simulator::Cancel (m_sendEvent);
    }

  if (m_socket)
    {
      m_socket->Close ();
    }
}

void
MyApp::SendPacket (void)
{
  Ptr<Packet> packet = Create<Packet>
(m_packetSize);
  m_socket->Send (packet);

  if (++m_packetsSent < m_nPackets)
    {
      ScheduleTx ();
    }
}
void
MyApp::ScheduleTx (void)
```

```
{
  if (m_running)
    {
      Time tNext (Seconds (m_packetSize *
8 / static_cast<double>
(m_dataRate.GetBitRate ())));
      m_sendEvent = Simulator::Schedule
(tNext, &MyApp::SendPacket, this);
    }
}

static void
CwndChange
(Ptr<OutputStreamWrapper> stream,
uint32_t oldCwnd, uint32_t newCwnd)
{
  NS_LOG_UNCOND (Simulator::Now
().GetSeconds () << "\t" << newCwnd);
  *stream->GetStream () <<
Simulator::Now ().GetSeconds () << "\t"
<< oldCwnd << "\t" << newCwnd <<
std::endl;
}

static void
RxDrop (Ptr<PcapFileWrapper> file,
Ptr<const Packet> p)
{
  NS_LOG_UNCOND ("RxDrop at " <<
Simulator::Now ().GetSeconds ());
  file->Write (Simulator::Now (), p);
}

int
main (int argc, char *argv[])
{
  bool useV6 = false;

  CommandLine cmd;
  cmd.AddValue ("useIpv6", "Use Ipv6",
useV6);
  cmd.Parse (argc, argv);
  NodeContainer nodes;
  nodes.Create (2);

  PointToPointHelper pointToPoint;
pointToPoint.SetDeviceAttribute
("DataRate", StringValue ("5Mbps"));
```

```
pointToPoint.SetChannelAttribute
("Delay", StringValue ("2ms"));

  NetDeviceContainer devices;
  devices = pointToPoint.Install (nodes);

  Ptr<RateErrorModel> em =
CreateObject<RateErrorModel> ();
  em->SetAttribute ("ErrorRate",
DoubleValue (0.00001));
  devices.Get (1)->SetAttribute
("ReceiveErrorModel", PointerValue
(em));

  InternetStackHelper stack;
  stack.Install (nodes);

  uint16_t sinkPort = 8080;
  Address sinkAddress;
  Address anyAddress;
  std::string probeType;
  std::string tracePath;
  if (useV6 == false)
    {
      Ipv4AddressHelper address;
      address.SetBase ("10.1.1.0",
"255.255.255.0");
      Ipv4InterfaceContainer interfaces =
address.Assign (devices);
      sinkAddress = InetSocketAddress
(interfaces.GetAddress (1), sinkPort);
      anyAddress = InetSocketAddress
(Ipv4Address::GetAny (), sinkPort);
      probeType = "ns3::Ipv4PacketProbe";
      tracePath =
"/NodeList/*/$ns3::Ipv4L3Protocol/Tx";
    }
  else
    {
      Ipv6AddressHelper address;
      address.SetBase
("2001:0000:f00d:cafe::", Ipv6Prefix (64));
      Ipv6InterfaceContainer interfaces =
address.Assign (devices);
      sinkAddress = Inet6SocketAddress
(interfaces.GetAddress (1,1), sinkPort);
      anyAddress = Inet6SocketAddress
(Ipv6Address::GetAny (), sinkPort);
```

```
probeType = "ns3::Ipv6PacketProbe";
      tracePath =
"/NodeList/*/$ns3::Ipv6L3Protocol/Tx";
    }

  PacketSinkHelper packetSinkHelper
("ns3::TcpSocketFactory", anyAddress);
  ApplicationContainer sinkApps =
packetSinkHelper.Install (nodes.Get (1));
  sinkApps.Start (Seconds (0.));
  sinkApps.Stop (Seconds (20.));
  Ptr<Socket> ns3TcpSocket =
Socket::CreateSocket (nodes.Get (0),
TcpSocketFactory::GetTypeId ());

  Ptr<MyApp> app =
CreateObject<MyApp> ();
  app->Setup (ns3TcpSocket, sinkAddress,
1040, 1000, DataRate ("1Mbps"));
  nodes.Get (0)->AddApplication (app);
  app->SetStartTime (Seconds (1.));
  app->SetStopTime (Seconds (20.));

  AsciiTraceHelper asciiTraceHelper;
  Ptr<OutputStreamWrapper> stream =
asciiTraceHelper.CreateFileStream
("seventh.cwnd");
  ns3TcpSocket-
>TraceConnectWithoutContext
("CongestionWindow",
MakeBoundCallback (&CwndChange,
stream));

  PcapHelper pcapHelper;
  Ptr<PcapFileWrapper> file =
pcapHelper.CreateFile ("seventh.pcap",
std::ios::out, PcapHelper::DLT_PPP);
  devices.Get (1)-
>TraceConnectWithoutContext
("PhyRxDrop", MakeBoundCallback
(&RxDrop, file));
  // Use GnuplotHelper to plot the packet
byte count over time
  GnuplotHelper plotHelper;
  // Configure the plot.  The first argument
is the file name prefix
  // for the output files generated.  The
second, third, and fourth
```

```
  // arguments are, respectively, the plot title, x-axis, and y-axis labels
  plotHelper.ConfigurePlot ("seventh-packet-byte-count",
                            "Packet Byte Count vs. Time",
                            "Time (Seconds)",
                            "Packet Byte Count");
  // Specify the probe type, trace source path (in configuration namespace), and
  // probe output trace source ("OutputBytes") to plot.  The fourth argument
  // specifies the name of the data series label on the plot.  The last
  // argument formats the plot by specifying where the key should be placed.
  plotHelper.PlotProbe (probeType,
                        tracePath,
                        "OutputBytes",
                        "Packet Byte Count",
                        GnuplotAggregator::KEY_BELOW);
  // Use FileHelper to write out the packet byte count over time
  FileHelper fileHelper;

  // Configure the file to be written, and the formatting of output data.
  fileHelper.ConfigureFile ("seventh-packet-byte-count",
                            FileAggregator::FORMATTED);

  // Set the labels for this formatted output file.
  fileHelper.Set2dFormat ("Time (Seconds) = %.3e\tPacket Byte Count = %.0f");

  // Specify the probe type, trace source path (in configuration namespace), and
  // probe output trace source ("OutputBytes") to write.
  fileHelper.WriteProbe (probeType,
                         tracePath,
                         "OutputBytes");
  Simulator::Stop (Seconds (20));
  Simulator::Run ();
  Simulator::Destroy ();
  return 0;
}
```

- Output

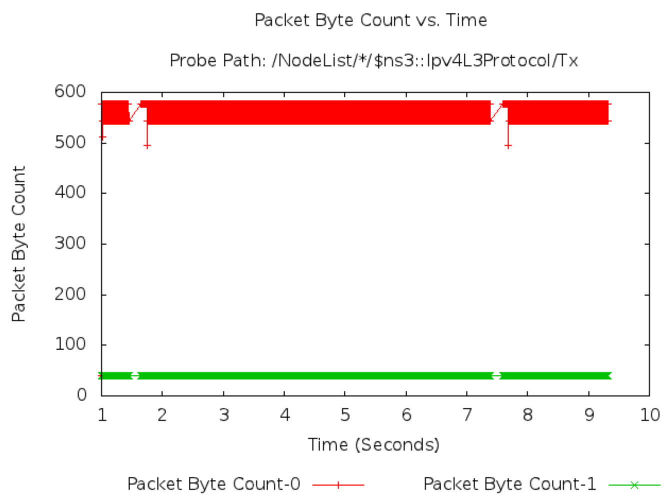


Fig.8. Packet count vs time graph for seventh.cc

6.2 Test cases in Quantum channel

1. Quantum channel implementation (qkd_channel_test.cc)

- Source code

```
// Network topology
//
//       n0 ---p2p-- n1 --p2p-- n2
//        |---------qkd---------|
//
// - udp flows from n0 to n2

#include <fstream>
#include "ns3/core-module.h"
#include "ns3/applications-module.h"
#include "ns3/internet-module.h"
#include "ns3/flow-monitor-module.h"
#include "ns3/mobility-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/gnuplot.h"

#include "ns3/qkd-helper.h"
#include "ns3/qkd-app-charging-helper.h"
#include "ns3/qkd-send.h"
#include "ns3/aodv-module.h"
#include "ns3/olsr-module.h"
#include "ns3/dsdv-module.h"

#include "ns3/network-module.h"
#include "ns3/fd-net-device-module.h"
#include "ns3/internet-apps-module.h"
#include "ns3/netanim-module.h"
```

```
#include "crypto++/aes.h"
#include "crypto++/modes.h"
#include "crypto++/filters.h"
#include <sstream>
#include <string>

using namespace ns3;
NS_LOG_COMPONENT_DEFINE
("QKD_CHANNEL_TEST");

uint32_t m_bytes_total = 0;
uint32_t m_bytes_received = 0;
uint32_t m_bytes_sent = 0;
uint32_t m_packets_received = 0;
double m_time = 0;
void
SentPacket(std::string context, Ptr<const
Packet> p){
   m_bytes_sent += p->GetSize();
}

void
ReceivedPacket(std::string context,
Ptr<const Packet> p, const Address&
addr){

m_bytes_received += p->GetSize();
   m_bytes_total += p->GetSize();
   m_packets_received++;

}
Ratio(uint32_t m_bytes_sent, uint32_t
m_packets_sent ){
   std::cout << "Sent (bytes):\t" <<
m_bytes_sent
   << "\tReceived (bytes):\t" <<
m_bytes_received
   << "\nSent (Packets):\t" <<
m_packets_sent
   << "\tReceived (Packets):\t" <<
m_packets_received
<< "\nRatio (bytes):\t" <<
(float)m_bytes_received/(float)m_bytes_s
ent
   << "\tRatio (packets):\t" <<
(float)m_packets_received/(float)m_packe
ts_sent << "\n";
}
```

```
int main (int argc, char *argv[])
{
   Packet::EnablePrinting();
   PacketMetadata::Enable ();
   //
   // Explicitly create the nodes required
by the topology (shown above).
   //
   NS_LOG_INFO ("Create nodes.");
   NodeContainer n;
   n.Create (3);

   NodeContainer n0n1 = NodeContainer
(n.Get(0), n.Get (1));
   NodeContainer n1n2 = NodeContainer
(n.Get(1), n.Get (2));

   //Enable OLSR
   //AodvHelper routingProtocol;
   //OlsrHelper routingProtocol;
   DsdvHelper routingProtocol;

   InternetStackHelper internet;
   internet.SetRoutingHelper
(routingProtocol);
   internet.Install (n);

   // Set Mobility for all nodes
   MobilityHelper mobility;
   Ptr<ListPositionAllocator>
positionAlloc = CreateObject
<ListPositionAllocator>();
   positionAlloc ->Add(Vector(0, 200, 0));
// node0
   positionAlloc ->Add(Vector(200, 200,
0)); // node1
   positionAlloc ->Add(Vector(400, 200,
0)); // node2
mobility.SetPositionAllocator(positionAll
oc);

mobility.SetMobilityModel("ns3::Constan
tPositionMobilityModel");
   mobility.Install(n);

   // We create the channels first without
any IP addressing information
```

```
   NS_LOG_INFO ("Create channels.");
   PointToPointHelper p2p;
   p2p.SetDeviceAttribute ("DataRate",
StringValue ("5Mbps"));
   p2p.SetChannelAttribute ("Delay",
StringValue ("2ms"));

   NetDeviceContainer d0d1 = p2p.Install
(n0n1);
   NetDeviceContainer d1d2 = p2p.Install
(n1n2);

   //
   // We've got the "hardware" in place.
Now we need to add IP addresses.
   //
   NS_LOG_INFO ("Assign IP
Addresses.");
   Ipv4AddressHelper ipv4;

   ipv4.SetBase ("10.1.1.0",
"255.255.255.0");
   Ipv4InterfaceContainer i0i1 =
ipv4.Assign (d0d1);

   ipv4.SetBase ("10.1.2.0",
"255.255.255.0");
   Ipv4InterfaceContainer i1i2 =
ipv4.Assign (d1d2);

   //
   // Explicitly create the channels
required by the topology (shown above).
   //
   //  install QKD Managers on the nodes
   QKDHelper QHelper;
   QHelper.InstallQKDManager (n);


   //create QKD connection between nodes
0 and 1
   NetDeviceContainer qkdNetDevices01
= QHelper.InstallQKD (
      d0d1.Get(0), d0d1.Get(1),
      1048576,    //min
      11324620, //thr
      52428800,   //max
      52428800     //current    //20485770
   );

   //Create graph to monitor buffer
changes
   QHelper.AddGraph(n.Get(0),
d0d1.Get(0)); //srcNode,
destinationAddress, BufferTitle


   //create QKD connection between nodes
1 and 2
   NetDeviceContainer qkdNetDevices12
= QHelper.InstallQKD (
      d1d2.Get(0), d1d2.Get(1),
      1048576,    //min
      11324620, //thr
      52428800,   //max
      52428800     //current    //20485770
   );

   //Create graph to monitor buffer
changes
   QHelper.AddGraph(n.Get(1),
d0d1.Get(0)); //srcNode,
destinationAddress, BufferTitle

   NS_LOG_INFO ("Create
Applications.");

   std::cout << "Source IP address: " <<
i0i1.GetAddress(0) << std::endl;
   std::cout << "Destination IP address: "
<< i1i2.GetAddress(1) << std::endl;

   /* QKD APPs for charing  */
   QKDAppChargingHelper
qkdChargingApp("ns3::TcpSocketFactory
", i0i1.GetAddress(0),
i0i1.GetAddress(1), 3072000);
   ApplicationContainer qkdChrgApps =
qkdChargingApp.Install ( d0d1.Get(0),
d0d1.Get(1) );
   qkdChrgApps.Start (Seconds (5.));
   qkdChrgApps.Stop (Seconds (1500.));
QKDAppChargingHelper
qkdChargingApp12("ns3::TcpSocketFacto
ry", i1i2.GetAddress(0),
i1i2.GetAddress(1), 3072000);
```

```
   ApplicationContainer qkdChrgApps12
= qkdChargingApp12.Install (
d1d2.Get(0), d1d2.Get(1) );
   qkdChrgApps12.Start (Seconds (5.));
   qkdChrgApps12.Stop (Seconds
(1500.));



   /* Create user's traffic between v0 and
v1 */
   /* Create sink app */
   uint16_t sinkPort = 8080;
   QKDSinkAppHelper packetSinkHelper
("ns3::UdpSocketFactory",
InetSocketAddress (Ipv4Address::GetAny
(), sinkPort));
   ApplicationContainer sinkApps =
packetSinkHelper.Install (n.Get (2));
   sinkApps.Start (Seconds (25.));
   sinkApps.Stop (Seconds (300.));

   /* Create source app  */
   Address sinkAddress
(InetSocketAddress (i1i2.GetAddress(1),
sinkPort));
   Address sourceAddress
(InetSocketAddress (i0i1.GetAddress(0),
sinkPort));
   Ptr<Socket> socket =
Socket::CreateSocket (n.Get (0),
UdpSocketFactory::GetTypeId ());

   Ptr<QKDSend> app =
CreateObject<QKDSend> ();
   app->Setup (socket, sourceAddress,
sinkAddress, 640, 5, DataRate
("160kbps"));
   n.Get (0)->AddApplication (app);
   app->SetStartTime (Seconds (25.));
   app->SetStopTime (Seconds (300.));
   //////////////////////////////////////
   ////        STATISTICS
   //////////////////////////////////////

   //if we need we can create pcap files
   p2p.EnablePcapAll
("QKD_channel_test");

Config::Connect("/NodeList/*/Applicatio
nList/*/$ns3::QKDSend/Tx",
MakeCallback(&SentPacket));

Config::Connect("/NodeList/*/Applicatio
nList/*/$ns3::QKDSink/Rx",
MakeCallback(&ReceivedPacket));

   AnimationInterface anim
("testcase1.xml");
   anim.SetConstantPosition(n.Get(0), 0.0,
200.0);
   anim.SetConstantPosition(n.Get(1),
200.0, 200.0);
   anim.SetConstantPosition(n.Get(2),
400.0, 200.0);

   Simulator::Stop (Seconds (50));
   Simulator::Run ();

   Ratio(app->sendDataStats(), app-
>sendPacketStats());

   //Finally print the graphs
   QHelper.PrintGraphs();
   Simulator::Destroy ();
}
```

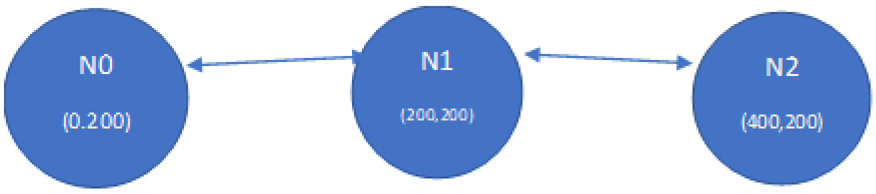
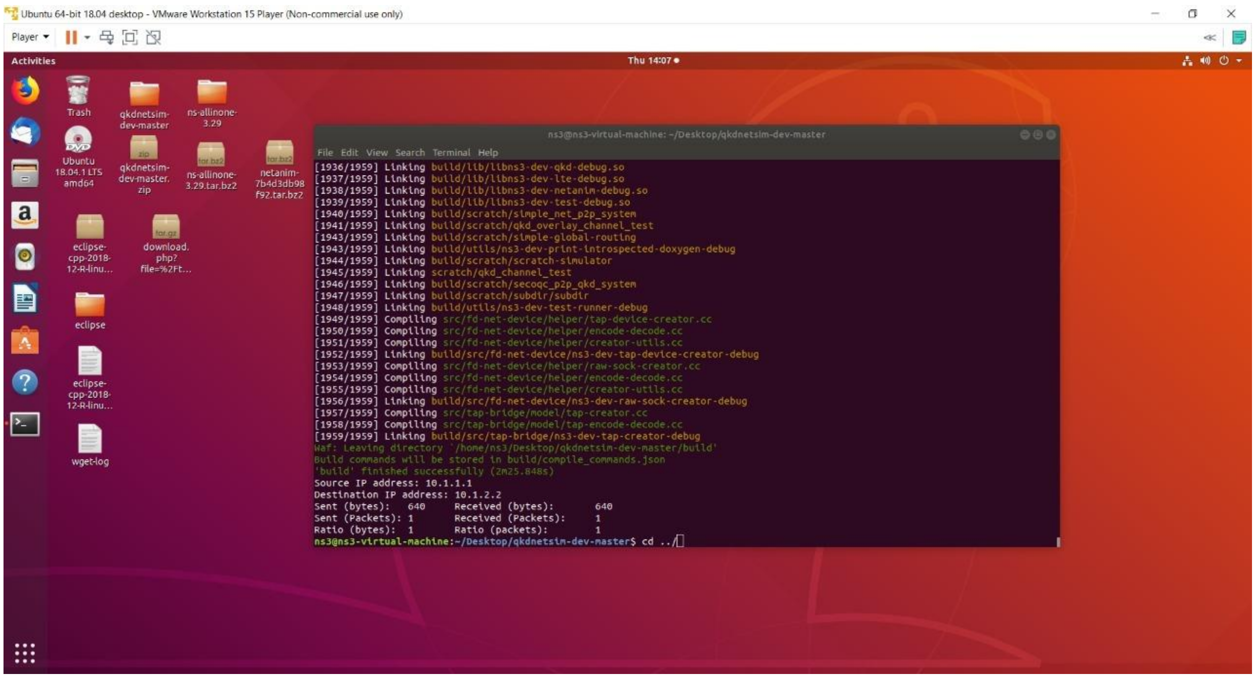


Fig.9: qkd_channel_test.cc network

- <u>Output</u>


Ubuntu 64-bit 18.04 desktop - VMware Workstation 15 Player (Non-commercial use only)
Player
Activities
Thu 14:07
ns3@ns3-virtual-machine: ~/Desktop/qkdnetsim-dev-master
File Edit View Search Terminal Help

```
[1936/1959] Linking build/lib/libns3-dev-qkd-debug.so
[1937/1959] Linking build/lib/libns3-dev-lte-debug.so
[1938/1959] Linking build/lib/libns3-dev-netanim-debug.so
[1939/1959] Linking build/lib/libns3-dev-test-debug.so
[1940/1959] Linking build/scratch/simple_net_p2p_system
[1941/1959] Linking build/scratch/qkd_overlay_channel_test
[1943/1959] Linking build/scratch/simple-global-routing
[1943/1959] Linking build/utils/ns3-dev-print-introspected-doxygen-debug
[1944/1959] Linking build/scratch/scratch-simulator
[1945/1959] Linking scratch/qkd_channel_test
[1946/1959] Linking build/scratch/secoqc_p2p_qkd_system
[1947/1959] Linking build/scratch/subdir/subdir
[1948/1959] Linking build/utils/ns3-dev-test-runner-debug
[1949/1959] Compiling src/fd-net-device/helper/tap-device-creator.cc
[1950/1959] Compiling src/fd-net-device/helper/encode-decode.cc
[1951/1959] Compiling src/fd-net-device/helper/creator-utils.cc
[1952/1959] Linking build/src/fd-net-device/ns3-dev-tap-device-creator-debug
[1953/1959] Compiling src/fd-net-device/helper/raw-sock-creator.cc
[1954/1959] Compiling src/fd-net-device/helper/encode-decode.cc
[1955/1959] Compiling src/fd-net-device/helper/creator-utils.cc
[1956/1959] Linking build/src/fd-net-device/ns3-dev-raw-sock-creator-debug
[1957/1959] Compiling src/tap-bridge/model/tap-creator.cc
[1958/1959] Compiling src/tap-bridge/model/tap-encode-decode.cc
[1959/1959] Linking build/src/tap-bridge/ns3-dev-tap-creator-debug
Waf: Leaving directory `/home/ns3/Desktop/qkdnetsim-dev-master/build'
Build commands will be stored in build/compile_commands.json
'build' finished successfully (2m25.848s)
Source IP address: 10.1.1.1
Destination IP address: 10.1.2.2
Sent (bytes):   640     Received (bytes):       640
Sent (Packets): 1       Received (Packets):     1
Ratio (bytes):  1       Ratio (packets):        1
ns3@ns3-virtual-machine:~/Desktop/qkdnetsim-dev-master$ cd ../
```

Fig.10: Output for qkd_channel_test.cc

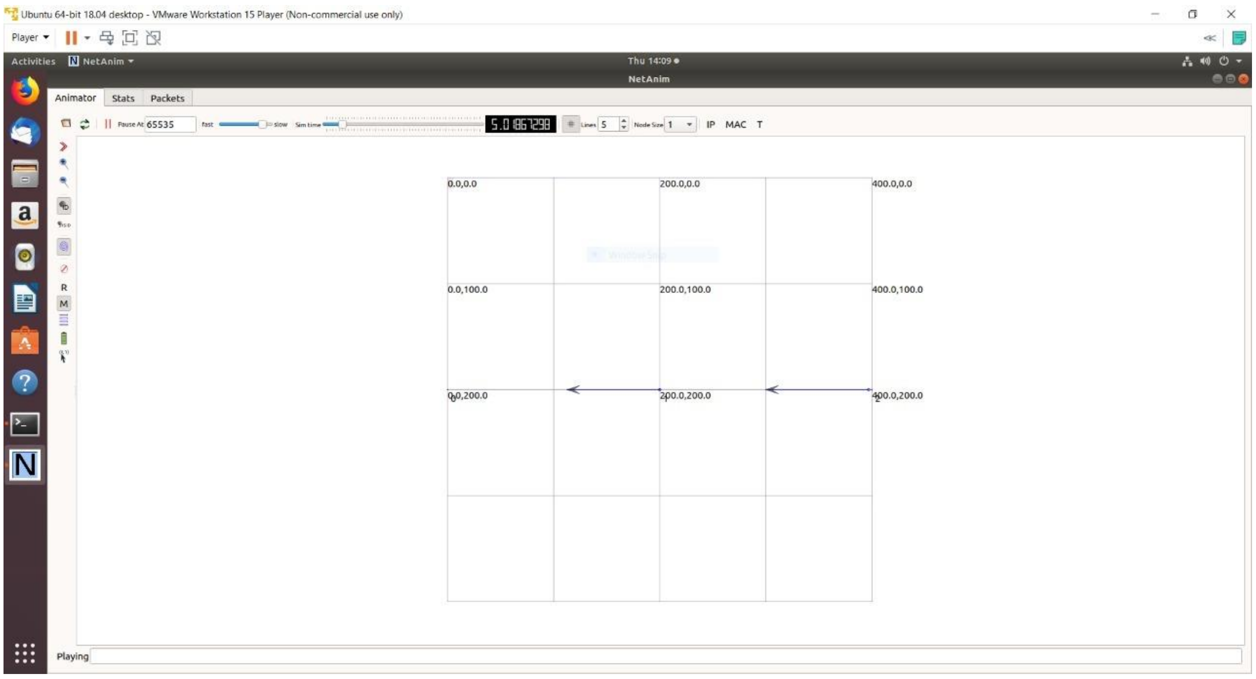


Fig.11: qkd_channel_test.cc network animation.

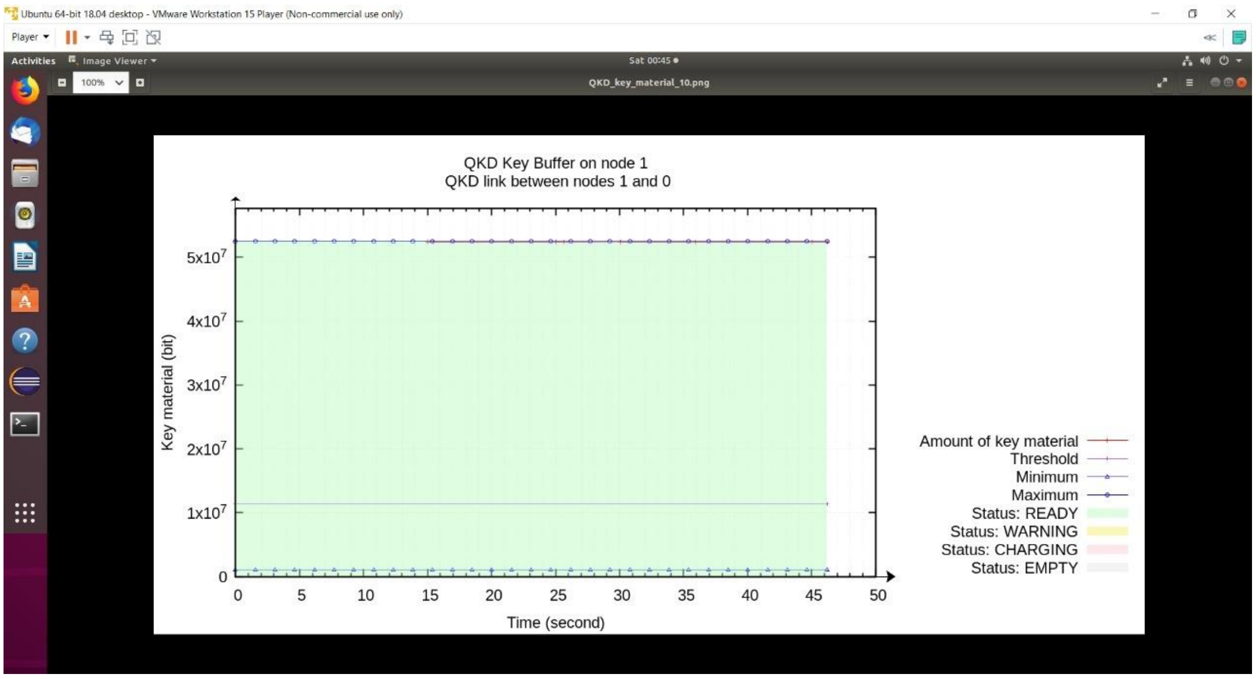


Fig.12: QKD buffer capacity between nodes 1&0

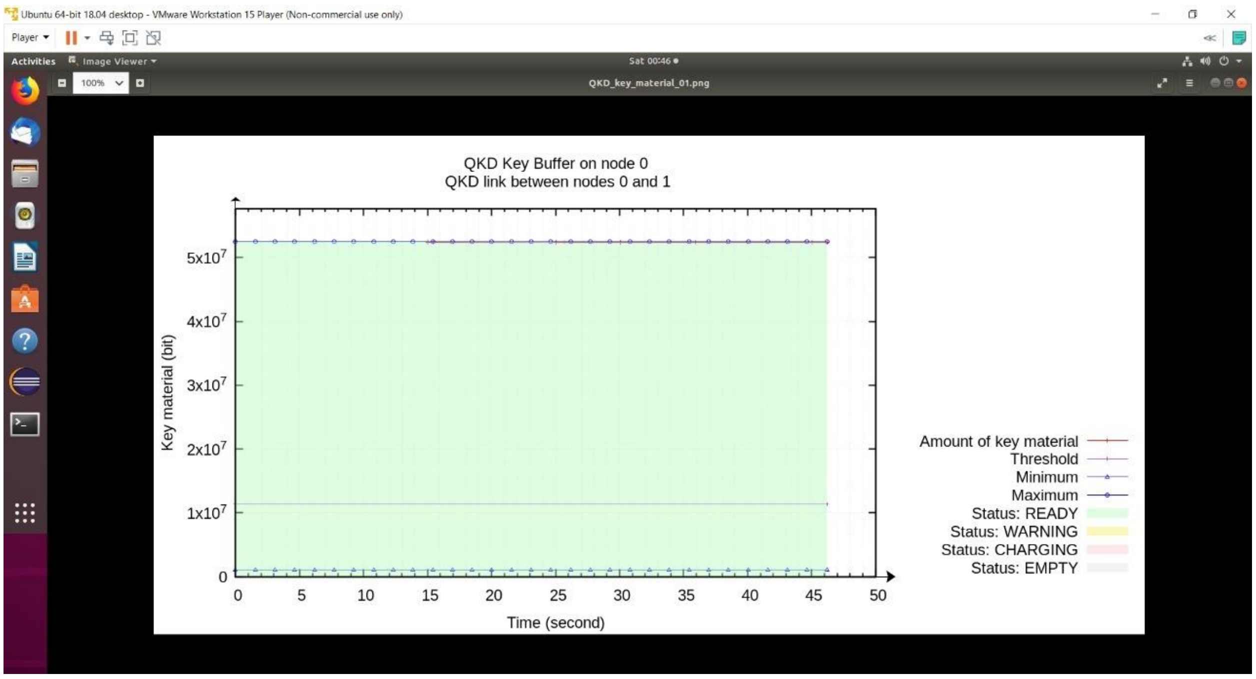


Fig.13: QKD buffer capacity between nodes 0&1

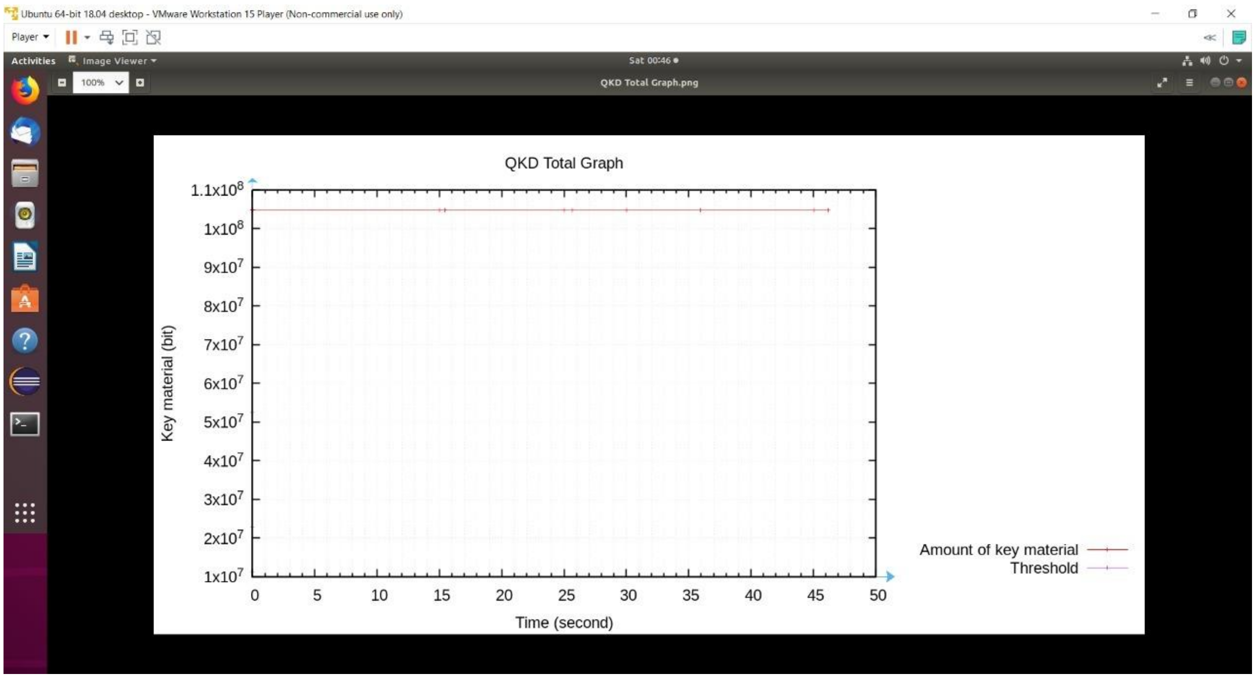


Fig.14: Total QKD buffer capacity

2. Quantum channel implementation for overlay network (qkd_overlay_channel_test.cc)

- Source code

```
/ Network topology
//
//      n0 ---p2p-- n1 --p2p-- n2
//       |---------qkd---------|
//
// - udp flows from n0 to n2
#include <fstream>
#include "ns3/core-module.h"
#include "ns3/qkd-helper.h"
#include "ns3/qkd-app-charging-
helper.h"
#include "ns3/qkd-graph-manager.h"
#include "ns3/applications-module.h"
#include "ns3/internet-module.h"
#include "ns3/flow-monitor-
module.h"
#include "ns3/mobility-module.h"
#include "ns3/point-to-point-
module.h"
#include "ns3/gnuplot.h"
#include "ns3/qkd-send.h"

#include "ns3/aodv-module.h"
#include "ns3/olsr-module.h"
#include "ns3/dsdv-module.h"
#include "ns3/dsr-module.h"

#include "ns3/aodvq-module.h"
#include "ns3/dsdvq-module.h"

using namespace ns3;

NS_LOG_COMPONENT_DEFINE
("QKD_CHANNEL_TEST");


uint32_t m_bytes_total = 0;
uint32_t m_bytes_received = 0;
uint32_t m_bytes_sent = 0;
uint32_t m_packets_received = 0;
double m_time = 0;

void
SentPacket(std::string context,
```

```
   m_bytes_sent += p->GetSize();
}

void
ReceivedPacket(std::string context,
Ptr<const Packet> p, const Address&
addr){

   m_bytes_received += p-
>GetSize();
   m_bytes_total += p->GetSize();
   m_packets_received++;

}

void
Ratio(uint32_t m_bytes_sent,
uint32_t m_packets_sent ){
   std::cout << "Sent (bytes):\t" <<
m_bytes_sent
   << "\tReceived (bytes):\t" <<
m_bytes_received
   << "\nSent (Packets):\t" <<
m_packets_sent
   << "\tReceived (Packets):\t" <<
m_packets_received

   << "\nRatio (bytes):\t" <<
(float)m_bytes_received/(float)m_byt
es_sent
   << "\tRatio (packets):\t" <<
(float)m_packets_received/(float)m_p
ackets_sent << "\n";
}
int main (int argc, char *argv[])
{
   Packet::EnablePrinting();
   PacketMetadata::Enable ();
   //
   // Explicitly create the nodes
required by the topology (shown
above).
```

```
NFO ("Create nodes.");

    NS_LOG_I    NodeContainer n;
      n.Create (5);

      NodeContainer n0n1 =
    NodeContainer (n.Get(0), n.Get (1));
      NodeContainer n1n2 =
    NodeContainer (n.Get(1), n.Get (2));
      NodeContainer n2n3 =
    NodeContainer (n.Get(2), n.Get (3));
      NodeContainer n3n4 =
    NodeContainer (n.Get(3), n.Get (4));
      NodeContainer qkdNodes =
    NodeContainer (
        n.Get (0),
        n.Get (2),
        n.Get (4)
      );

      //Underlay network - set routing
    protocol (if any)

      //Enable OLSR
      //OlsrHelper routingProtocol;
      //DsdvHelper routingProtocol;

      InternetStackHelper internet;
      //internet.SetRoutingHelper
    (routingProtocol);
      internet.Install (n);

      // Set Mobility for all nodes
      MobilityHelper mobility;
      Ptr<ListPositionAllocator>
    positionAlloc = CreateObject
    <ListPositionAllocator>();
      positionAlloc ->Add(Vector(0,
    200, 0)); // node0
      positionAlloc ->Add(Vector(200,
    200, 0)); // node1
      positionAlloc ->Add(Vector(400,
    200, 0)); // node2
      positionAlloc ->Add(Vector(600,
    200, 0)); // node3
      positionAlloc ->Add(Vector(800,
    200, 0)); // node4
```

```
    mobility.SetPositionAllocator(positio
    nAlloc);

    mobility.SetMobilityModel("ns3::Co
    nstantPositionMobilityModel");
      mobility.Install(n);

      // We create the channels first
    without any IP addressing
    information
      NS_LOG_INFO ("Create
    channels.");
      PointToPointHelper p2p;
      p2p.SetDeviceAttribute
    ("DataRate", StringValue ("1Gbps"));
      //p2p.SetChannelAttribute
    ("Delay", StringValue ("100ps"));

      NetDeviceContainer d0d1 =
    p2p.Install (n0n1);
      NetDeviceContainer d1d2 =
    p2p.Install (n1n2);
      NetDeviceContainer d2d3 =
    p2p.Install (n2n3);
      NetDeviceContainer d3d4 =
    p2p.Install (n3n4);

      //
      // We've got the "hardware" in
    place.  Now we need to add IP
    addresses.
      //
      NS_LOG_INFO ("Assign IP
    Addresses.");
      Ipv4AddressHelper ipv4;

      ipv4.SetBase ("10.1.1.0",
    "255.255.255.0");
      Ipv4InterfaceContainer i0i1 =
    ipv4.Assign (d0d1);

      ipv4.SetBase ("10.1.2.0",
    "255.255.255.0");
      Ipv4InterfaceContainer i1i2 =
    ipv4.Assign (d1d2);
```

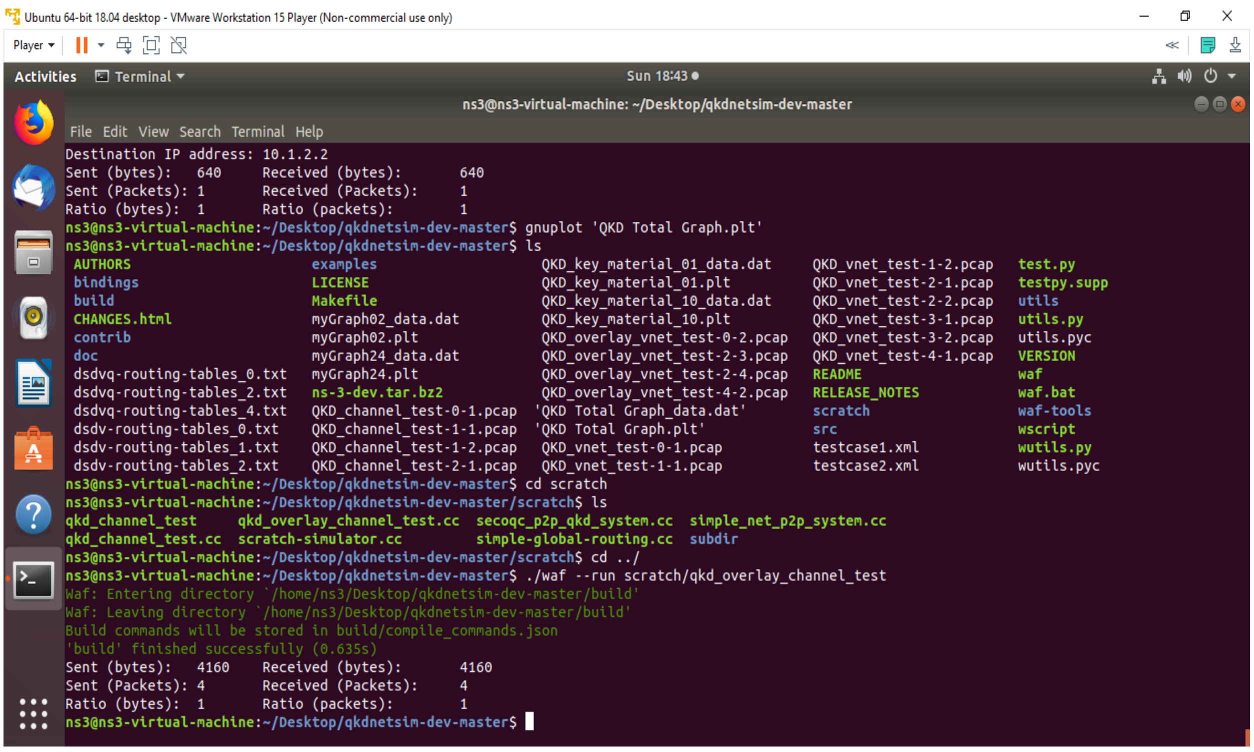
```
   ipv4.SetBase ("10.1.3.0",
"255.255.255.0");
   Ipv4InterfaceContainer i2i3 =
ipv4.Assign (d2d3);

   ipv4.SetBase ("10.1.4.0",
"255.255.255.0");
   Ipv4InterfaceContainer i3i4 =
ipv4.Assign (d3d4);

   // Create router nodes, initialize
routing database and set up the
routing
   // tables in the nodes.

Ipv4GlobalRoutingHelper::PopulateR
outingTables ();

 //Overlay network - set routing
protocol

   //Enable Overlay Routing
   //AodvqHelper
routingOverlayProtocol;
   DsdvqHelper
routingOverlayProtocol;
   //
   // Explicitly create the channels
required by the topology (shown
above).
   //
   QKDHelper QHelper;

   //install QKD Managers on the
nodes
   QHelper.SetRoutingHelper
(routingOverlayProtocol);
   QHelper.InstallQKDManager
(qkdNodes);
//Create QKDNetDevices and create
QKDbuffers
   Ipv4InterfaceAddress va02_0
(Ipv4Address ("11.0.0.1"), Ipv4Mask
("255.255.255.0"));
   Ipv4InterfaceAddress va02_2
(Ipv4Address ("11.0.0.2"), Ipv4Mask
("255.255.255.0"));
```

```
   Ipv4InterfaceAddress va24_2
(Ipv4Address ("11.0.0.3"), Ipv4Mask
("255.255.255.0"));
   Ipv4InterfaceAddress va24_4
(Ipv4Address ("11.0.0.4"), Ipv4Mask
("255.255.255.0"));
   //create QKD connection between
nodes 0 and 2
   NetDeviceContainer
qkdNetDevices02 =
QHelper.InstallOverlayQKD (
       d0d1.Get(0), d1d2.Get(1),
       va02_0, va02_2,
       108576,     //min
       0,          //thr - will be set
automatically
       1085760,    //max
       1085760     //current
//20485770
   );
   //Create graph to monitor buffer
changes

QHelper.AddGraph(qkdNodes.Get(0)
, d1d2.Get (0), "myGraph02");
//srcNode, destinationAddress,
BufferTitle

   //create QKD connection between
nodes 0 and 2
   NetDeviceContainer
qkdNetDevices24 =
QHelper.InstallOverlayQKD (
       d2d3.Get(0), d3d4.Get(1),
       va24_2, va24_4,
       108576,     //min
       0,          //thr - will be set
automatically
       1085760,    //max
       1085760     //current    //88576
   );
   //Create graph to monitor buffer
changes

QHelper.AddGraph(qkdNodes.Get(1)
, d3d4.Get (0), "myGraph24");
//srcNode, destinationAddress,
BufferTitle
```

```
NS_LOG_INFO ("Create
Applications.");

   /* QKD APPs for charing */
   //Config::SetDefault
("ns3::TcpSocket::SegmentSize",
UintegerValue (2536));
   QKDAppChargingHelper
qkdChargingApp02("ns3::VirtualTcp
SocketFactory", va02_0.GetLocal (),
va02_2.GetLocal (), 200000);
   ApplicationContainer
qkdChrgApps02 =
qkdChargingApp02.Install (
qkdNetDevices02.Get (0),
qkdNetDevices02.Get (1) );
   qkdChrgApps02.Start (Seconds
(10.));
   qkdChrgApps02.Stop (Seconds
(120.));
   QKDAppChargingHelper
qkdChargingApp24("ns3::VirtualTcp
SocketFactory", va24_2.GetLocal (),
va24_4.GetLocal (), 200000);
   ApplicationContainer
qkdChrgApps24 =
qkdChargingApp24.Install (
qkdNetDevices24.Get (0),
qkdNetDevices24.Get (1) );
   qkdChrgApps24.Start (Seconds
(10.));
   qkdChrgApps24.Stop (Seconds
(120.));

   /* Create user's traffic between v0
and v1 */
   /* Create sink app */
   uint16_t sinkPort = 8080;
   QKDSinkAppHelper
packetSinkHelper
("ns3::VirtualUdpSocketFactory",
InetSocketAddress
(Ipv4Address::GetAny (), sinkPort));
   ApplicationContainer sinkApps =
packetSinkHelper.Install
(qkdNodes.Get (2));
   sinkApps.Start (Seconds (25.));
   sinkApps.Stop (Seconds (170.));
```

```
/* Create source app  */
   Address sinkAddress
(InetSocketAddress
(va24_4.GetLocal (), sinkPort));
   Address sourceAddress
(InetSocketAddress
(va02_0.GetLocal (), sinkPort));
   Ptr<Socket> overlaySocket =
Socket::CreateSocket (qkdNodes.Get
(0),
VirtualUdpSocketFactory::GetTypeId
());
    Ptr<QKDSend> app =
CreateObject<QKDSend> ();
   app->Setup (overlaySocket,
sourceAddress, sinkAddress, 1040, 0,
DataRate ("5kbps"));
   qkdNodes.Get (0)-
>AddApplication (app);
   app->SetStartTime (Seconds (25.));
   app->SetStopTime (Seconds
(170.));
   //////////////////////////////////////
   ////        STATISTICS
   //////////////////////////////////////
   //if we need we can create pcap
files
   p2p.EnablePcapAll
("QKD_vnet_test");
   QHelper.EnablePcapAll
("QKD_overlay_vnet_test");
Config::Connect("/NodeList/*/Applic
ationList/*/$ns3::QKDSend/Tx",
MakeCallback(&SentPacket));

Config::Connect("/NodeList/*/Applic
ationList/*/$ns3::QKDSink/Rx",
MakeCallback(&ReceivedPacket));
   Simulator::Stop ( Seconds (30) );
   Simulator::Run ();
   Ratio(app->sendDataStats(), app-
>sendPacketStats());

   //Finally print the graphs
   QHelper.PrintGraphs();
   Simulator::Destroy ();
}
```

- <u>Output</u>



Ubuntu 64-bit 18.04 desktop - VMware Workstation 15 Player (Non-commercial use only)
Player
Activities
Terminal
Sun 18:43
ns3@ns3-virtual-machine: ~/Desktop/qkdnetsim-dev-master
File Edit View Search Terminal Help

```
Destination IP address: 10.1.2.2
Sent (bytes):   640     Received (bytes):       640
Sent (Packets): 1       Received (Packets):     1
Ratio (bytes):  1       Ratio (packets):        1
ns3@ns3-virtual-machine:~/Desktop/qkdnetsim-dev-master$ gnuplot 'QKD Total Graph.plt'
ns3@ns3-virtual-machine:~/Desktop/qkdnetsim-dev-master$ ls
 AUTHORS                     examples                    QKD_key_material_01_data.dat   QKD_vnet_test-1-2.pcap   test.py
 bindings                    LICENSE                     QKD_key_material_01.plt        QKD_vnet_test-2-1.pcap   testpy.supp
 build                       Makefile                    QKD_key_material_10_data.dat   QKD_vnet_test-2-2.pcap   utils
 CHANGES.html                myGraph02_data.dat          QKD_key_material_10.plt        QKD_vnet_test-3-1.pcap   utils.py
 contrib                     myGraph02.plt               QKD_overlay_vnet_test-0-2.pcap QKD_vnet_test-3-2.pcap   utils.pyc
 doc                         myGraph24_data.dat          QKD_overlay_vnet_test-2-3.pcap QKD_vnet_test-4-1.pcap   VERSION
 dsdvq-routing-tables_0.txt  myGraph24.plt               QKD_overlay_vnet_test-2-4.pcap README                   waf
 dsdvq-routing-tables_2.txt  ns-3-dev.tar.bz2            QKD_overlay_vnet_test-4-2.pcap RELEASE_NOTES            waf.bat
 dsdvq-routing-tables_4.txt  QKD_channel_test-0-1.pcap  'QKD Total Graph_data.dat'      scratch                  waf-tools
 dsdv-routing-tables_0.txt   QKD_channel_test-1-1.pcap  'QKD Total Graph.plt'           src                      wscript
 dsdv-routing-tables_1.txt   QKD_channel_test-1-2.pcap   QKD_vnet_test-0-1.pcap         testcase1.xml            wutils.py
 dsdv-routing-tables_2.txt   QKD_channel_test-2-1.pcap   QKD_vnet_test-1-1.pcap         testcase2.xml            wutils.pyc
ns3@ns3-virtual-machine:~/Desktop/qkdnetsim-dev-master$ cd scratch
ns3@ns3-virtual-machine:~/Desktop/qkdnetsim-dev-master/scratch$ ls
qkd_channel_test     qkd_overlay_channel_test.cc  secoqc_p2p_qkd_system.cc  simple_net_p2p_system.cc
qkd_channel_test.cc  scratch-simulator.cc         simple-global-routing.cc  subdir
ns3@ns3-virtual-machine:~/Desktop/qkdnetsim-dev-master/scratch$ cd ../
ns3@ns3-virtual-machine:~/Desktop/qkdnetsim-dev-master$ ./waf --run scratch/qkd_overlay_channel_test
Waf: Entering directory `/home/ns3/Desktop/qkdnetsim-dev-master/build'
Waf: Leaving directory `/home/ns3/Desktop/qkdnetsim-dev-master/build'
Build commands will be stored in build/compile_commands.json
'build' finished successfully (0.635s)
Sent (bytes):   4160    Received (bytes):       4160
Sent (Packets): 4       Received (Packets):     4
Ratio (bytes):  1       Ratio (packets):        1
ns3@ns3-virtual-machine:~/Desktop/qkdnetsim-dev-master$ 
```

Fig.15: Output for qkd_overlay_channel_test.cc

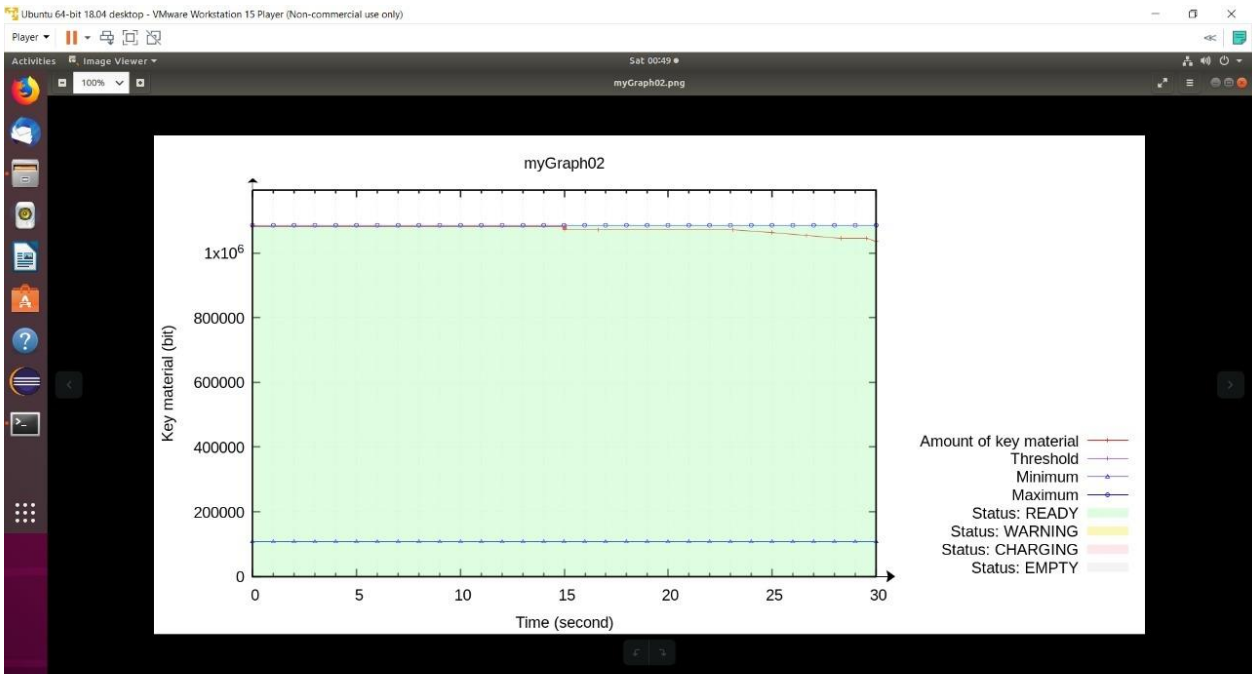


Fig.16: QKD buffer capacity between nodes 0&2

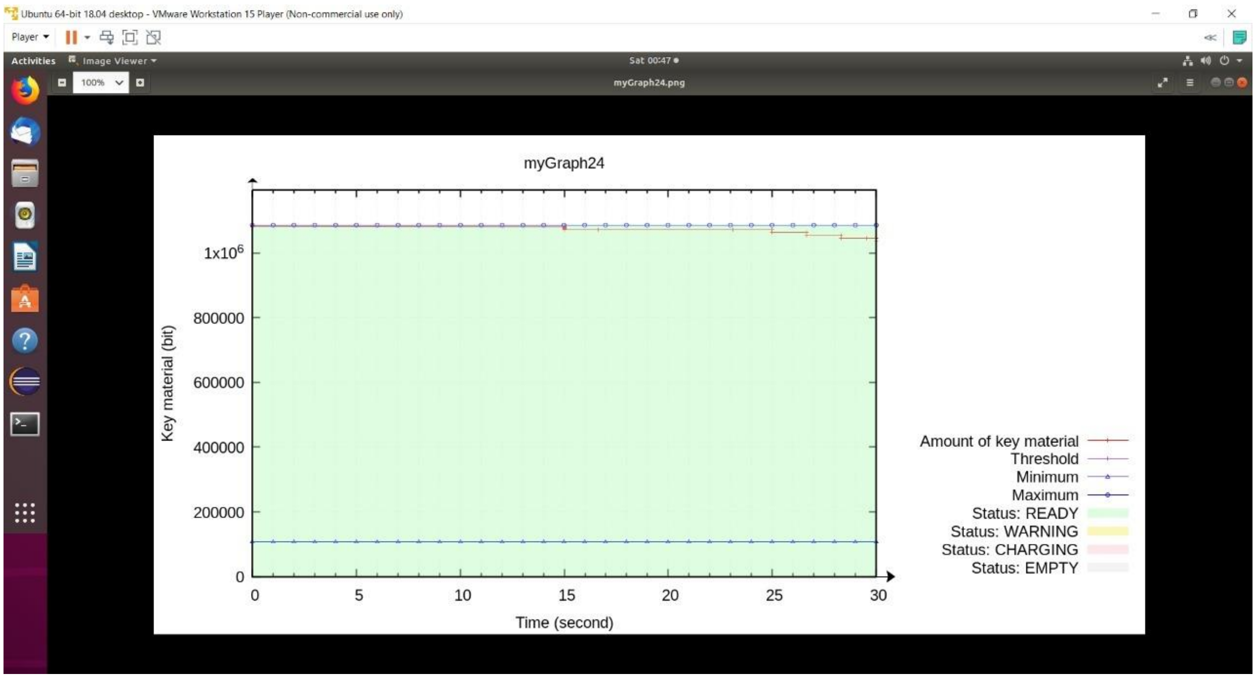


Fig.17: QKD buffer capacity between nodes 2&4

3. Quantum channel implementation for peer to peer network (secoqc_p2p_qkd_system.cc)

- Source Code

```
// Network topology
// Simulation of SECOQC QKD Network
//                        2Mbps          15Mbps
//                  n3(SIE) -- 2kbps -- n4(ERD) --- 17kbps n5(FRM)
//                   |           |
//                   5.7kbps (5.5Mbps) 8kbps (7.2Mbps)
//                   |           |
//      n0(STP) --0.5kbps-- n1(BRT) --1kbps --- n2(GUD)
//             1Mbps          1Mbps
// - All links are wired point-to-point
// - TCP flow from n0 to n5
#include <fstream>
#include "ns3/core-module.h"
#include "ns3/applications-module.h"
#include "ns3/internet-module.h"
#include "ns3/flow-monitor-module.h"
#include "ns3/mobility-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/gnuplot.h"
#include "ns3/qkd-helper.h"
#include "ns3/qkd-app-charging-helper.h"
#include "ns3/qkd-send.h
#include "ns3/aodv-module.h"
#include "ns3/olsr-module.h"
#include "ns3/dsdv-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-apps-module.h"
```

```
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
NS_LOG_COMPONENT_DEFINE
("SECOQC");
using namespace ns3;
uint32_t m_bytes_total = 0;
uint32_t m_bytes_received = 0;
uint32_t m_bytes_sent = 0;
uint32_t m_packets_received = 0;
double m_time = 0;
void
SentPacket(std::string context,
Ptr<const Packet> p){
   m_bytes_sent += p->GetSize();
}
void
ReceivedPacket(std::string context,
Ptr<const Packet> p, const Address&
addr){
   m_bytes_received += p-
>GetSize();
   m_bytes_total += p->GetSize();
   m_packets_received++;
}
void
Ratio(uint32_t m_bytes_sent,
uint32_t m_packets_sent ){
   std::cout << "Sent (bytes):\t" <<
m_bytes_sent
   << "\tReceived (bytes):\t" <<
m_bytes_received
   << "\nSent (Packets):\t" <<
m_packets_sent
   << "\tReceived (Packets):\t" <<
m_packets_received
   << "\nRatio (bytes):\t" <<
(float)m_bytes_received/(float)m_byt
es_sent
   << "\tRatio (packets):\t" <<
(float)m_packets_received/(float)m_p
ackets_sent << "\n";
}
int main (int argc, char *argv[]){
Packet::EnablePrinting();

PacketMetadata::Enable ();
```

```
bool enableFlowMonitor = false;
   bool enableApplication = true;
   double simulationTime = 300;
   bool useSeparetedIPAddresses =
false;
   std::string lat = "2ms";
   std::string rate = "10Mb/s"; // P2P
link
   CommandLine cmd;
   cmd.AddValue ("EnableMonitor",
"Enable Flow Monitor",
enableFlowMonitor);
   cmd.AddValue ("simulationTime",
"simulationTime", simulationTime);
   cmd.AddValue
("enableApplication",
"enableApplication",
enableApplication);
   cmd.Parse (argc, argv);
   NS_LOG_INFO ("Create QKD
system.");
   // Explicitly create the nodes
required by the topology (shown
above).
   NS_LOG_INFO ("Create nodes.");
   NodeContainer nodes; // ALL
Nodes
   nodes.Create(6);
   //Enable OLSR
   //OlsrHelper routingProtocol;
   //AodvHelper routingProtocol;
   DsdvHelper routingProtocol;
   // Install Internet Stack
   InternetStackHelper internet;
   internet.SetRoutingHelper
(routingProtocol);
   internet.Install (nodes);
   // Set up Addresses
   NS_LOG_INFO ("Create
channels.");
   NS_LOG_INFO ("Assign IP
Addresses.");
   Ipv4AddressHelper ipv4;
   // Explicitly create the channels
required by the topology (shown
above).
   PointToPointHelper p2p;
```

```
p2p.SetDeviceAttribute ("DataRate",
StringValue (rate));
   p2p.SetChannelAttribute ("Delay",
StringValue (lat));
   //Nodes 0 & 1
   NodeContainer link_0_1;
   link_0_1.Add(nodes.Get(0));
   link_0_1.Add(nodes.Get(1));
   NetDeviceContainer devices_0_1 =
p2p.Install (link_0_1);
   ipv4.SetBase ("10.1.1.0",
"255.255.255.0");
   Ipv4InterfaceContainer ifconf_0_1
= ipv4.Assign (devices_0_1);
   //Nodes 1 & 2
   NodeContainer link_1_2;
   link_1_2.Add(nodes.Get(1));
   link_1_2.Add(nodes.Get(2));
NetDeviceContainer devices_1_2 =
p2p.Install (link_1_2);
   if(useSeparetedIPAddresses)
      ipv4.SetBase ("10.1.2.0",
"255.255.255.0");
   Ipv4InterfaceContainer ifconf_1_2
= ipv4.Assign (devices_1_2);
   //Nodes 1 & 2
   NodeContainer link_1_3;
   link_1_3.Add(nodes.Get(1));
   link_1_3.Add(nodes.Get(3));
   NetDeviceContainer devices_1_3 =
p2p.Install (link_1_3);
   if(useSeparetedIPAddresses)
      ipv4.SetBase ("10.1.3.0",
"255.255.255.0");
Ipv4InterfaceContainer ifconf_1_3 =
ipv4.Assign (devices_1_3);
   //Nodes 3 & 4
   NodeContainer link_3_4;
   link_3_4.Add(nodes.Get(3));
   link_3_4.Add(nodes.Get(4));
   NetDeviceContainer devices_3_4 =
p2p.Install (link_3_4);
   if(useSeparetedIPAddresses)
      ipv4.SetBase ("10.1.4.0",
"255.255.255.0");
   Ipv4InterfaceContainer ifconf_3_4
= ipv4.Assign (devices_3_4);
```

```
   NodeContainer link_2_4;
   link_2_4.Add(nodes.Get(2));
   link_2_4.Add(nodes.Get(4));
   NetDeviceContainer devices_2_4 =
p2p.Install (link_2_4);
   if(useSeparetedIPAddresses)
      ipv4.SetBase ("10.1.5.0",
"255.255.255.0");
   Ipv4InterfaceContainer ifconf_2_4
= ipv4.Assign (devices_2_4);
   NodeContainer link_4_5;
   link_4_5.Add(nodes.Get(4));
   link_4_5.Add(nodes.Get(5));
   NetDeviceContainer devices_4_5 =
p2p.Install (link_4_5);
   if(useSeparetedIPAddresses)
      ipv4.SetBase ("10.1.6.0",
"255.255.255.0");
   Ipv4InterfaceContainer ifconf_4_5
= ipv4.Assign (devices_4_5);
   QKDHelper QHelper;
   QHelper.InstallQKDManager
(nodes);
NetDeviceContainer
qkdNetDevices_0_1 =
QHelper.InstallQKD (
      devices_0_1.Get(0),
devices_0_1.Get(1),
      1048576,    //min
      11324620, //thr
      52428800,   //max
      52428800     //current
   );
QHelper.AddGraph(nodes.Get(0),
devices_0_1.Get(0)); //srcNode,
destinationAddress, BufferTitle
   NetDeviceContainer
qkdNetDevices_1_2 =
QHelper.InstallQKD (
      devices_1_2.Get(0),
devices_1_2.Get(1),
      1548576,    //min
      11324620, //thr
      52428800,   //max
      52428800     //current
//20485770
   );
```

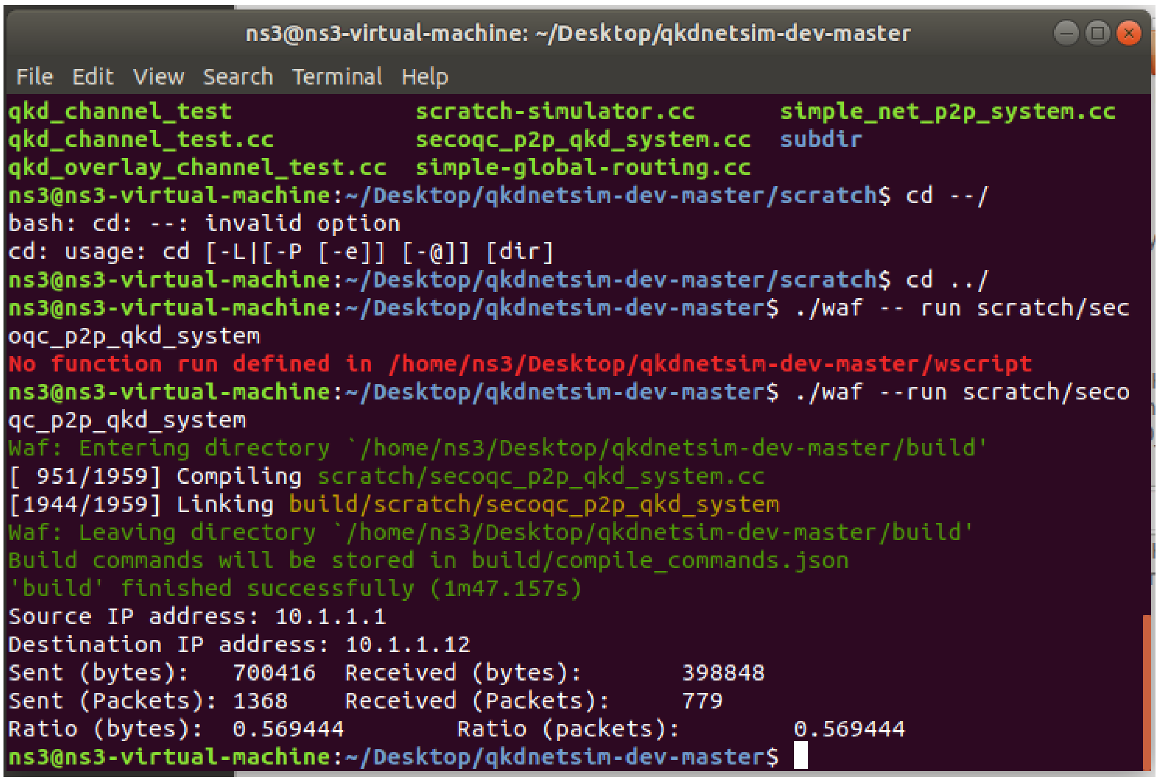
```
QHelper.AddGraph(nodes.Get(1),
devices_1_2.Get(1)); //srcNode,
destinationAddress, BufferTitle
   NetDeviceContainer
qkdNetDevices_2_4 =
QHelper.InstallQKD (
      devices_2_4.Get(0),
devices_2_4.Get(1),
      1048576,    //min
      11324620, //thr
      52428800,   //max
      10485760     //current
//10485760
   );
   QHelper.AddGraph(nodes.Get(2),
devices_2_4.Get(1)); //srcNode,
destinationAddress, BufferTitle
   NetDeviceContainer
qkdNetDevices_1_3 =
QHelper.InstallQKD (
      devices_1_3.Get(0),
devices_1_3.Get(1),
      1048576,    //min
      11324620, //thr
      52428800,   //max
      52428800     //current
//20485770
   );
QHelper.AddGraph(nodes.Get(1),
devices_1_3.Get(1)); //srcNode,
destihnationAddress, BufferTitle
   NetDeviceContainer
qkdNetDevices_3_4 =
QHelper.InstallQKD (
      devices_3_4.Get(0),
devices_3_4.Get(1),
      1048576,    //min
      11324620, //thr
      52428800,   //max
      12485760     //current
//12485760
   );
QHelper.AddGraph(nodes.Get(3),
devices_3_4.Get(1)); //srcNode,
destinationAddress, BufferTitle
NetDeviceContainer
qkdNetDevices_4_5 =
```

```
QHelper.InstallQKD (
      devices_4_5.Get(0),
devices_4_5.Get(1),
      1048576,    //min
      11324620, //thr
      52428800,   //max
      52428800     //current
//20485770
   );
   QHelper.AddGraph(nodes.Get(4),
devices_4_5.Get(1)); //srcNode,
destinationAddress, BufferTitle
  MobilityHelper mobility;
   Ptr<ListPositionAllocator>
positionAlloc = CreateObject
<ListPositionAllocator>();
   positionAlloc ->Add(Vector(0,
200, 0)); // node0
   positionAlloc ->Add(Vector(200,
200, 0)); // node1
   positionAlloc ->Add(Vector(400,
60, 0)); // node2
   positionAlloc ->Add(Vector(400,
350, 0)); // node3
   positionAlloc ->Add(Vector(600,
200, 0)); // node4
   positionAlloc ->Add(Vector(700,
200, 0)); // node5
mobility.SetPositionAllocator(positio
nAlloc);
mobility.SetMobilityModel("ns3::Co
nstantPositionMobilityModel");
   mobility.Install(nodes);
   NS_LOG_INFO ("Create
Applications.");
   std::cout << "Source IP address: "
<< ifconf_0_1.GetAddress(0) <<
std::endl;
   std::cout << "Destination IP
address: " <<
ifconf_4_5.GetAddress(1) <<
std::endl;
QKDAppChargingHelper
qkdChargingApp_0_1("ns3::TcpSock
etFactory",
ifconf_0_1.GetAddress(0),
ifconf_0_1.GetAddress(1), 3072000);
//102400 * 30seconds
```

```
ApplicationContainer
qkdChrgApps_0_1 =
qkdChargingApp_0_1.Install (
devices_0_1.Get(0),
devices_0_1.Get(1) );
   qkdChrgApps_0_1.Start (Seconds
(5.));
   qkdChrgApps_0_1.Stop (Seconds
(500.));
   QKDAppChargingHelper
qkdChargingApp_1_2("ns3::TcpSock
etFactory",
ifconf_1_2.GetAddress(0),
ifconf_1_2.GetAddress(1), 3072000);
//102400 * 30seconds
   ApplicationContainer
qkdChrgApps_1_2 =
qkdChargingApp_1_2.Install (
devices_1_2.Get(0),
devices_1_2.Get(1) );
   qkdChrgApps_1_2.Start (Seconds
(5.));
   qkdChrgApps_1_2.Stop (Seconds
(500.));
  QKDAppChargingHelper
qkdChargingApp_2_4("ns3::TcpSock
etFactory",
ifconf_2_4.GetAddress(0),
ifconf_2_4.GetAddress(1), 3072000);
//102400 * 30seconds
   ApplicationContainer
qkdChrgApps_2_4 =
qkdChargingApp_2_4.Install (
devices_2_4.Get(0),
devices_2_4.Get(1) );
   qkdChrgApps_2_4.Start (Seconds
(5.));
   qkdChrgApps_2_4.Stop (Seconds
(500.));
QKDAppChargingHelper
qkdChargingApp_1_3("ns3::TcpSock
etFactory",
ifconf_1_3.GetAddress(0),
ifconf_1_3.GetAddress(1), 3072000);
//102400 * 30seconds
   ApplicationContainer
qkdChrgApps_1_3 =
qkdChargingApp_1_3.Install (
```

```
devices_3_4.Get(0),
devices_3_4.Get(1) );
   qkdChrgApps_3_4.Start (Seconds
(5.));
   qkdChrgApps_3_4.Stop (Seconds
(500.));
QKDAppChargingHelper
qkdChargingApp_4_5("ns3::TcpSock
etFactory",
ifconf_4_5.GetAddress(0),
ifconf_4_5.GetAddress(1), 3072000);
//102400 * 30seconds
   ApplicationContainer
qkdChrgApps_4_5 =
qkdChargingApp_4_5.Install (
devices_4_5.Get(0),
devices_4_5.Get(1) );
   qkdChrgApps_4_5.Start (Seconds
(5.));
   qkdChrgApps_4_5.Stop (Seconds
(500.));
Ptr<QKDSend> app =
CreateObject<QKDSend> ();
   if(enableApplication){
      /* Create user's traffic between
v0 and v1 */
      /* Create sink app */
      uint16_t sinkPort = 8080;
      QKDSinkAppHelper
packetSinkHelper
("ns3::UdpSocketFactory",
InetSocketAddress
(Ipv4Address::GetAny (), sinkPort));
      ApplicationContainer sinkApps
= packetSinkHelper.Install
(nodes.Get (5));
      sinkApps.Start (Seconds (15.));
      sinkApps.Stop (Seconds (500.));
Address sinkAddress
(InetSocketAddress
(ifconf_4_5.GetAddress(1),
sinkPort));
      Address sourceAddress
(InetSocketAddress
(ifconf_0_1.GetAddress(0),
sinkPort));
      Ptr<Socket> socket =
Socket::CreateSocket (nodes.Get (0),
 UdpSocketFactory::GetTypeId ());
     app->Setup (socket, sourceAddress, sinkAddress, 512, 0, DataRate ("160kbps"));
     nodes.Get (0)->AddApplication (app);
     app->SetStartTime (Seconds (15.));
     app->SetStopTime (Seconds (500.));
   }
p2p.EnablePcapAll ("QKD_netsim_test");
   //QHelper.EnablePcapAll ("QKD_netsim_test_Qhelper");
Config::Connect("/NodeList/*/ApplicationList/*/$ns3::QKDSink/Rx",
MakeCallback(&ReceivedPacket));
   NS_LOG_INFO ("Run Simulation.");
Simulator::Stop (Seconds(simulationTime));
   Simulator::Run ();
   if(enableApplication)
      Ratio(app->sendDataStats(), app->sendPacketStats());
   //Finally print the graphs
   QHelper.PrintGraphs();
   Simulator::Destroy ();
}
```

- Output


ns3@ns3-virtual-machine: ~/Desktop/qkdnetsim-dev-master
File Edit View Search Terminal Help

```
qkd_channel_test              scratch-simulator.cc      simple_net_p2p_system.cc
qkd_channel_test.cc           secoqc_p2p_qkd_system.cc  subdir
qkd_overlay_channel_test.cc   simple-global-routing.cc
ns3@ns3-virtual-machine:~/Desktop/qkdnetsim-dev-master/scratch$ cd --/
bash: cd: --: invalid option
cd: usage: cd [-L|[-P [-e]] [-@]] [dir]
ns3@ns3-virtual-machine:~/Desktop/qkdnetsim-dev-master/scratch$ cd ../
ns3@ns3-virtual-machine:~/Desktop/qkdnetsim-dev-master$ ./waf -- run scratch/sec
oqc_p2p_qkd_system
No function run defined in /home/ns3/Desktop/qkdnetsim-dev-master/wscript
ns3@ns3-virtual-machine:~/Desktop/qkdnetsim-dev-master$ ./waf --run scratch/seco
qc_p2p_qkd_system
Waf: Entering directory `/home/ns3/Desktop/qkdnetsim-dev-master/build'
[ 951/1959] Compiling scratch/secoqc_p2p_qkd_system.cc
[1944/1959] Linking build/scratch/secoqc_p2p_qkd_system
Waf: Leaving directory `/home/ns3/Desktop/qkdnetsim-dev-master/build'
Build commands will be stored in build/compile_commands.json
'build' finished successfully (1m47.157s)
Source IP address: 10.1.1.1
Destination IP address: 10.1.1.12
Sent (bytes):    700416  Received (bytes):       398848
Sent (Packets): 1368     Received (Packets):     779
Ratio (bytes):   0.569444        Ratio (packets):        0.569444
ns3@ns3-virtual-machine:~/Desktop/qkdnetsim-dev-master$
```

Fig.18: Output for secoqc_p2p_qkd_system.cc

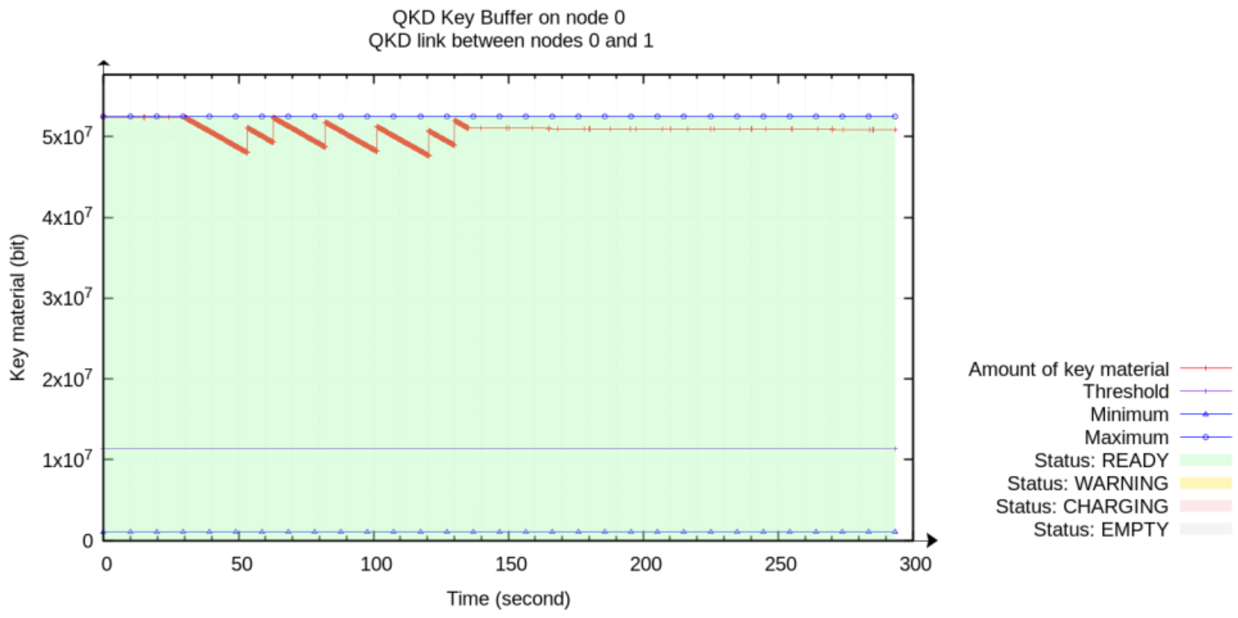


Fig.19: QKD buffer capacity between nodes 0&1.

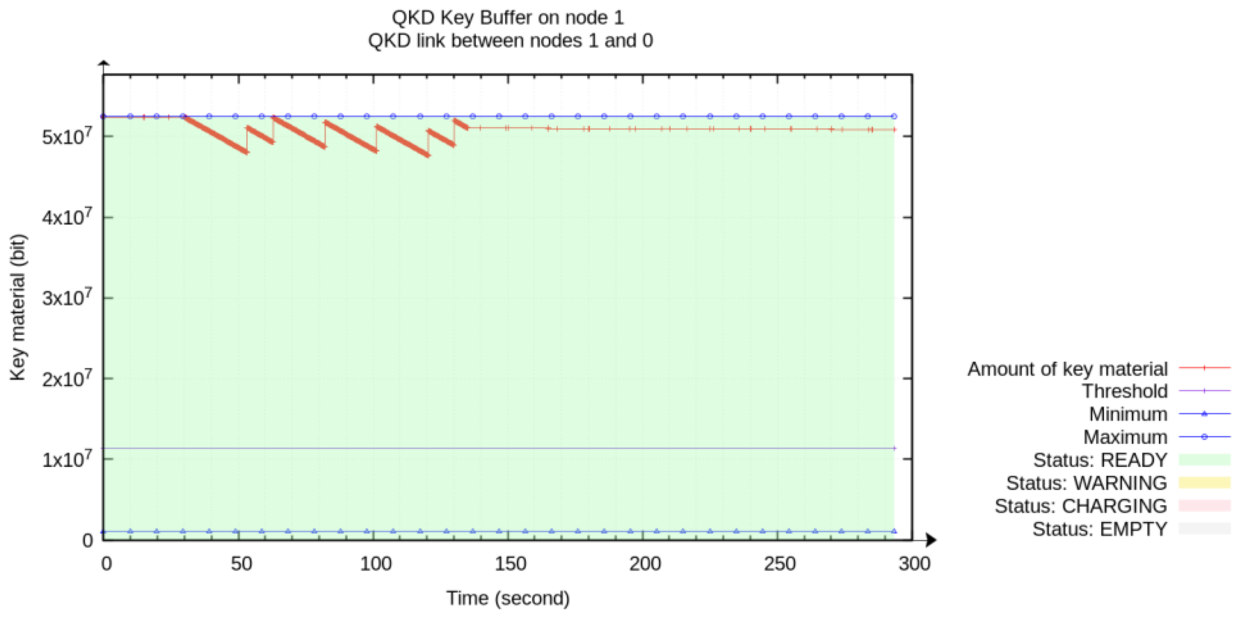


Fig.20: QKD buffer capacity between nodes 1&0.

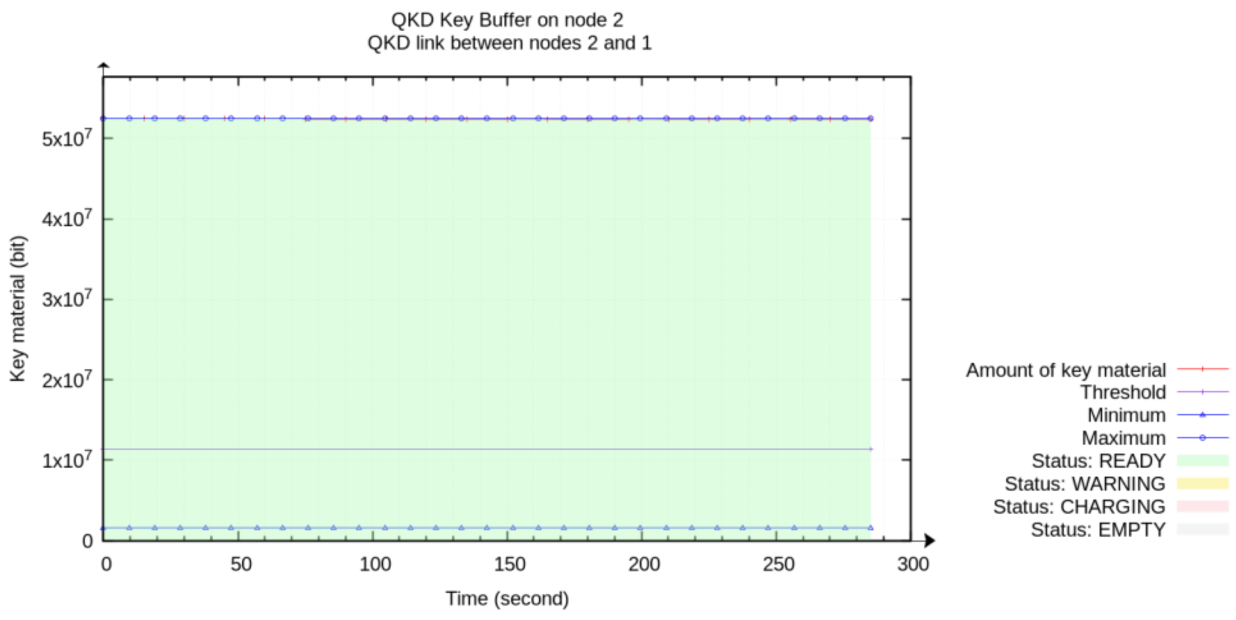


Fig.19: QKD buffer capacity between nodes 2&1.

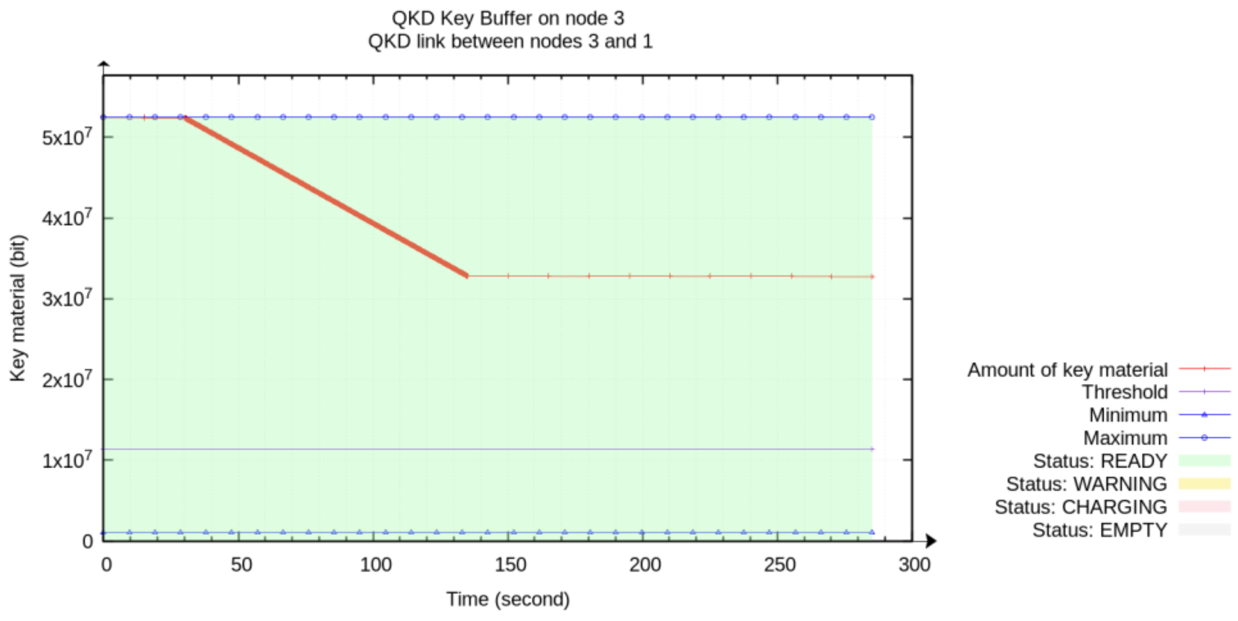


Fig.21: QKD buffer capacity between nodes 3&1.

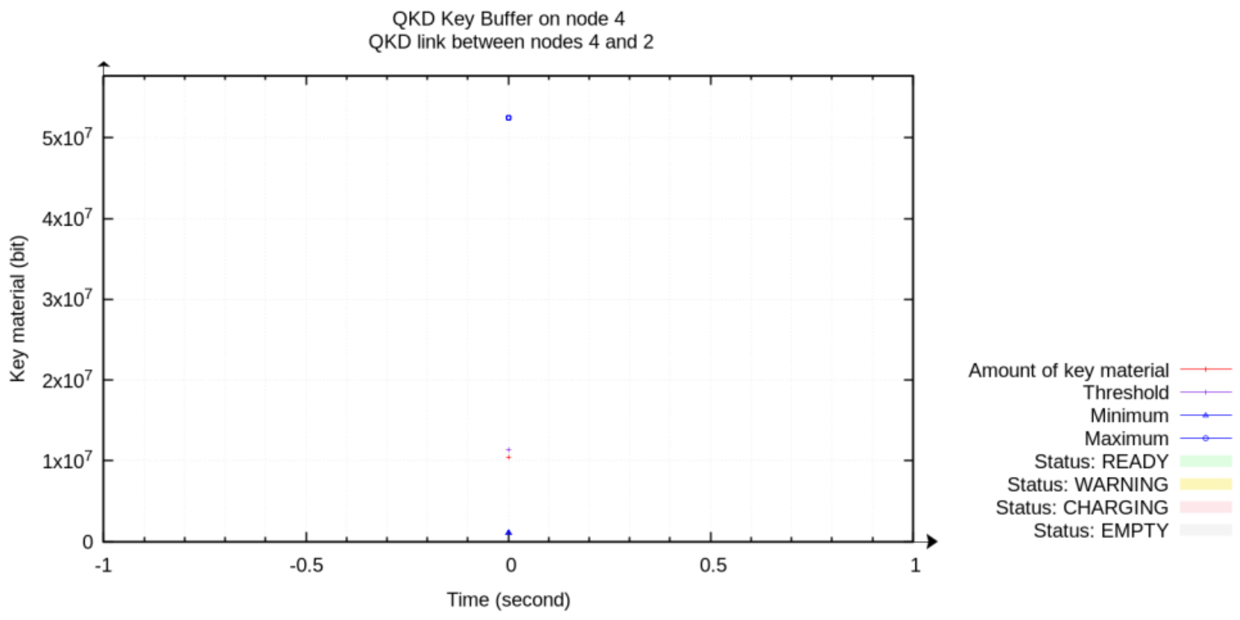


Fig.22: QKD buffer capacity between nodes 4&2.

4. Five node mesh network overlay channel implementation for peer to peer network using ipv4 (sixmesh.cc)

- Source Code

```
#include <fstream>
#include "ns3/core-module.h"
#include "ns3/applications-module.h"
#include "ns3/internet-module.h"
#include "ns3/flow-monitor-module.h"
#include "ns3/mobility-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/gnuplot.h"

#include "ns3/qkd-helper.h"
#include "ns3/qkd-app-charging-helper.h"
#include "ns3/qkd-send.h"

#include "ns3/aodv-module.h"
#include "ns3/olsr-module.h"
#include "ns3/dsdv-module.h"

#include "ns3/network-module.h"
#include "ns3/internet-apps-module.h"
#include "ns3/netanim-module.h"

#include <iostream>
#include <fstream>
#include <vector>
#include <string>

NS_LOG_COMPONENT_DEFINE ("SECOQC");

using namespace ns3;
uint32_t m_bytes_total = 0;
uint32_t m_bytes_received = 0;
uint32_t m_bytes_sent = 0;
uint32_t m_packets_received = 0;
double m_time = 0;

void
SentPacket(std::string context, Ptr<const
Packet> p){

  m_bytes_sent += p->GetSize();
}
void
ReceivedPacket(std::string context, Ptr<const
Packet> p, const Address& addr){

  m_bytes_received += p->GetSize();
  m_bytes_total += p->GetSize();
```

```
void
Ratio(uint32_t m_bytes_sent, uint32_t
m_packets_sent ){
  std::cout << "Sent (bytes):\t" <<
m_bytes_sent
  << "\tReceived (bytes):\t" <<
m_bytes_received
  << "\nSent (Packets):\t" <<
m_packets_sent
  << "\tReceived (Packets):\t" <<
m_packets_received

  << "\nRatio (bytes):\t" <<
(float)m_bytes_received/(float)m_bytes_sent
  << "\tRatio (packets):\t" <<
(float)m_packets_received/(float)m_packets_
sent << "\n";
}
int main (int argc, char *argv[])
{
  Packet::EnablePrinting();
  PacketMetadata::Enable ();
  bool enableFlowMonitor = false;
  bool enableApplication = true;
  double simulationTime = 300;
  bool useSeparetedIPAddresses = false;
  std::string lat = "2ms";
  std::string rate = "10Mb/s"; // P2P link
  CommandLine cmd;
  cmd.AddValue ("EnableMonitor", "Enable
Flow Monitor", enableFlowMonitor);
  cmd.AddValue ("simulationTime",
"simulationTime", simulationTime);
  cmd.AddValue ("enableApplication",
"enableApplication", enableApplication);
  cmd.Parse (argc, argv);

  NS_LOG_INFO ("Create QKD system.");
  // Explicitly create the nodes required by
the topology (shown above).
  NS_LOG_INFO ("Create nodes.");
  NodeContainer nodes; // ALL Nodes
  nodes.Create(6);
  //Enable OLSR
  //OlsrHelper routingProtocol;
  //AodvHelper routingProtocol;
  DsdvHelper routingProtocol;
  InternetStackHelper internet;
   internet.SetRoutingHelper
(routingProtocol);
   internet.Install (nodes);
   // Set up Addresses
   NS_LOG_INFO ("Create channels.");
   NS_LOG_INFO ("Assign IP Addresses.");
   Ipv4AddressHelper ipv4;
   // Explicitly create the channels required by
the topology (shown above).
   PointToPointHelper p2p;
   p2p.SetDeviceAttribute ("DataRate",
StringValue (rate));
   p2p.SetChannelAttribute ("Delay",
StringValue (lat));
   //Nodes 0 & 1
   NodeContainer link_0_1;
   link_0_1.Add(nodes.Get(0));
   link_0_1.Add(nodes.Get(1));
   NetDeviceContainer devices_0_1 =
p2p.Install (link_0_1);
   ipv4.SetBase ("10.1.1.0", "255.255.255.0");
   Ipv4InterfaceContainer ifconf_0_1 =
ipv4.Assign (devices_0_1);
   //Nodes 0 & 2
   NodeContainer link_0_2;
   link_0_2.Add(nodes.Get(0));
   link_0_2.Add(nodes.Get(2));
NetDeviceContainer devices_0_2 = p2p.Install
(link_0_2);

   if(useSeparetedIPAddresses)
      ipv4.SetBase ("10.1.2.0",
"255.255.255.0");
   Ipv4InterfaceContainer ifconf_0_2 =
ipv4.Assign (devices_0_2);
   //Nodes 0 & 3
   NodeContainer link_0_3;
   link_0_3.Add(nodes.Get(0));
   link_0_3.Add(nodes.Get(3));
   NetDeviceContainer devices_0_3 =
p2p.Install (link_0_3);
   if(useSeparetedIPAddresses)
      ipv4.SetBase ("10.1.3.0",
"255.255.255.0");
   Ipv4InterfaceContainer ifconf_0_3 =
ipv4.Assign (devices_0_3);
   //Nodes 0 & 4
   NodeContainer link_0_4;
   link_0_4.Add(nodes.Get(0));
   link_0_4.Add(nodes.Get(4));
```

```
NetDeviceContainer devices_0_4 = p2p.Install
(link_0_4);

   if(useSeparetedIPAddresses)
      ipv4.SetBase ("10.1.4.0",
"255.255.255.0");
   Ipv4InterfaceContainer ifconf_0_4 =
ipv4.Assign (devices_0_4);
   //Nodes 0 & 5
   NodeContainer link_0_5;
   link_0_5.Add(nodes.Get(0));
   link_0_5.Add(nodes.Get(5));

   NetDeviceContainer devices_0_5 =
p2p.Install (link_0_5);

   if(useSeparetedIPAddresses)
      ipv4.SetBase ("10.1.5.0",
"255.255.255.0");

   Ipv4InterfaceContainer ifconf_0_5 =
ipv4.Assign (devices_0_5);

   //Nodes 1 & 2
   NodeContainer link_1_2;
   link_1_2.Add(nodes.Get(1));
   link_1_2.Add(nodes.Get(2));

   NetDeviceContainer devices_1_2 =
p2p.Install (link_1_2);
   if(useSeparetedIPAddresses)
      ipv4.SetBase ("10.1.6.0",
"255.255.255.0");
   Ipv4InterfaceContainer ifconf_1_2 =
ipv4.Assign (devices_1_2);

//Nodes 1 & 3
   NodeContainer link_1_3;
   link_1_3.Add(nodes.Get(1));
   link_1_3.Add(nodes.Get(3));

   NetDeviceContainer devices_1_3 =
p2p.Install (link_1_3);
   ipv4.SetBase ("10.1.7.0", "255.255.255.0");
   Ipv4InterfaceContainer ifconf_1_3 =
ipv4.Assign (devices_1_3);

//Nodes 1 &4
   NodeContainer link_1_4;
   link_1_4.Add(nodes.Get(1));
   link_1_4.Add(nodes.Get(4));
```

```
NetDeviceContainer devices_1_4 = p2p.Install
(link_1_4);
   ipv4.SetBase ("10.1.8.0", "255.255.255.0");
   Ipv4InterfaceContainer ifconf_1_4 =
ipv4.Assign (devices_1_4);

//Nodes 1 & 5
   NodeContainer link_1_5;
   link_1_5.Add(nodes.Get(1));
   link_1_5.Add(nodes.Get(5));

   NetDeviceContainer devices_1_5 =
p2p.Install (link_1_5);
   ipv4.SetBase ("10.1.9.0", "255.255.255.0");
   Ipv4InterfaceContainer ifconf_1_5 =
ipv4.Assign (devices_1_5);

//Nodes 2 & 3
   NodeContainer link_2_3;
   link_2_3.Add(nodes.Get(2));
   link_2_3.Add(nodes.Get(3));

   NetDeviceContainer devices_2_3 =
p2p.Install (link_2_3);
   ipv4.SetBase ("10.1.10.0",
"255.255.255.0");
   Ipv4InterfaceContainer ifconf_2_3 =
ipv4.Assign (devices_2_3);

//Nodes 2 & 4
   NodeContainer link_2_4;
   link_2_4.Add(nodes.Get(2));
   link_2_4.Add(nodes.Get(4));
   NetDeviceContainer devices_2_4 =
p2p.Install (link_2_4);
   ipv4.SetBase ("10.1.11.0",
"255.255.255.0");
   Ipv4InterfaceContainer ifconf_2_4 =
ipv4.Assign (devices_2_4);

//Nodes 2 & 5
   NodeContainer link_2_5;
   link_2_5.Add(nodes.Get(2));
   link_2_5.Add(nodes.Get(5));

   NetDeviceContainer devices_2_5 =
p2p.Install (link_2_5);
   ipv4.SetBase ("10.1.12.0",
"255.255.255.0");
   Ipv4InterfaceContainer ifconf_2_5 =
ipv4.Assign (devices_2_5);
```

```
//Nodes 3 & 4
   NodeContainer link_3_4;
   link_3_4.Add(nodes.Get(3));
   link_3_4.Add(nodes.Get(4));

   NetDeviceContainer devices_3_4 =
p2p.Install (link_3_4);
   ipv4.SetBase ("10.1.13.0",
"255.255.255.0");
   Ipv4InterfaceContainer ifconf_3_4 =
ipv4.Assign (devices_3_4);

//Nodes 3 & 5
   NodeContainer link_3_5;
   link_3_5.Add(nodes.Get(3));
   link_3_5.Add(nodes.Get(5));

   NetDeviceContainer devices_3_5 =
p2p.Install (link_3_5);
   ipv4.SetBase ("10.1.14.0",
"255.255.255.0");
   Ipv4InterfaceContainer ifconf_3_5 =
ipv4.Assign (devices_3_5);

//Nodes 4 & 5
   NodeContainer link_4_5;
   link_4_5.Add(nodes.Get(4));
   link_4_5.Add(nodes.Get(5));

   NetDeviceContainer devices_4_5 =
p2p.Install (link_4_5);
   ipv4.SetBase ("10.1.15.0",
"255.255.255.0");
   Ipv4InterfaceContainer ifconf_4_5 =
ipv4.Assign (devices_4_5);



   // Create router nodes, initialize routing
database and set up the routing
   // tables in the nodes.

//Ipv4GlobalRoutingHelper::PopulateRouting
Tables ();
   //routingProtocol.Set
("LocationServiceName", StringValue
("GOD"));
   //routingProtocol.Install ();
   //  install QKD Managers on the nodes
   QKDHelper QHelper;
```

```
QHelper.InstallQKDManager (nodes);

   //create QKD connection between nodes 0
and 1
   NetDeviceContainer qkdNetDevices_0_1 =
QHelper.InstallQKD (
       devices_0_1.Get(0), devices_0_1.Get(1),
       1048576,    //min
       11324620, //thr
       52428800,   //max
       52428800     //current    //20485770
   );
   //Create graph to monitor buffer changes
   QHelper.AddGraph(nodes.Get(0),
devices_0_1.Get(0)); //srcNode,
destinationAddress, BufferTitle

   //create QKD connection between nodes 0
and 2
   NetDeviceContainer qkdNetDevices_0_2 =
QHelper.InstallQKD (
       devices_0_2.Get(0), devices_0_2.Get(1),
       1548576,    //min
       11324620, //thr
       52428800,   //max
       52428800     //current    //20485770
   );
   //Create graph to monitor buffer changes
   QHelper.AddGraph(nodes.Get(0),
devices_0_2.Get(1)); //srcNode,
destinationAddress, BufferTitle

//create QKD connection between nodes 0
and 3
   NetDeviceContainer qkdNetDevices_0_3 =
QHelper.InstallQKD (
       devices_0_3.Get(0), devices_0_3.Get(1),
       1048576,    //min
       11324620, //thr
       52428800,   //max
       10485760     //current    //10485760
);
   //Create graph to monitor buffer changes
   QHelper.AddGraph(nodes.Get(0),
devices_0_3.Get(1)); //srcNode,
destinationAddress, BufferTitle

//create QKD connection between nodes 0
and 4
   NetDeviceContainer qkdNetDevices_0_4 =
QHelper.InstallQKD (
devices_0_4.Get(0), devices_0_4.Get(1),
       1048576,    //min
       11324620, //thr
       52428800,   //max
       10485760     //current    //10485760
   );
   //Create graph to monitor buffer changes
   QHelper.AddGraph(nodes.Get(0),
devices_0_4.Get(1)); //srcNode,
destinationAddress, BufferTitle

//create QKD connection between nodes 0
and 5
   NetDeviceContainer qkdNetDevices_0_5 =
QHelper.InstallQKD (
       devices_0_5.Get(0), devices_0_5.Get(1),
       1048576,    //min
       11324620, //thr
       52428800,   //max
       10485760     //current    //10485760
   );
   //Create graph to monitor buffer changes
   QHelper.AddGraph(nodes.Get(0),
devices_0_5.Get(1)); //srcNode,
destinationAddress, BufferTitle

//create QKD connection between nodes 1
and 2
   NetDeviceContainer qkdNetDevices_1_2 =
QHelper.InstallQKD (
       devices_1_2.Get(0), devices_1_2.Get(1),
       1048576,    //min
       11324620, //thr
       52428800,   //max
       10485760     //current    //10485760
   );
   //Create graph to monitor buffer changes
   QHelper.AddGraph(nodes.Get(1),
devices_1_2.Get(1)); //srcNode,
destinationAddress, BufferTitle
//create QKD connection between nodes 1
and 3
   NetDeviceContainer qkdNetDevices_1_3 =
QHelper.InstallQKD (
       devices_1_3.Get(0), devices_1_3.Get(1),
       1048576,    //min
       11324620, //thr
       52428800,   //max
       10485760     //current    //10485760
);
   //Create graph to monitor buffer changes
```

```
//Create graph to monitor buffer changes
   QHelper.AddGraph(nodes.Get(3),
devices_3_5.Get(1)); //srcNode,
destinationAddress, BufferTitle
//create QKD connection between nodes 4
and 5
   NetDeviceContainer qkdNetDevices_4_5 =
QHelper.InstallQKD (
      devices_4_5.Get(0), devices_4_5.Get(1),
      1048576,    //min
      11324620, //thr
      52428800,   //max
      52428800     //current    //20485770
   );
   //Create graph to monitor buffer changes
   QHelper.AddGraph(nodes.Get(4),
devices_4_5.Get(1)); //srcNode,
destinationAddress, BufferTitle
   // Set Mobility for all nodes
   MobilityHelper mobility;
   Ptr<ListPositionAllocator> positionAlloc =
CreateObject <ListPositionAllocator>();
   positionAlloc ->Add(Vector(0, 200, 0)); //
node0
   positionAlloc ->Add(Vector(100, 0, 0)); //
node1
   positionAlloc ->Add(Vector(300, 0, 0)); //
node2
   positionAlloc ->Add(Vector(500, 200, 0)); //
node3
   positionAlloc ->Add(Vector(300, 400, 0)); //
node4
   positionAlloc ->Add(Vector(100, 400, 0)); //
node5
  mobility.SetPositionAllocator(positionAlloc);
mobility.SetMobilityModel("ns3::ConstantPos
itionMobilityModel");
   mobility.Install(nodes);
   NS_LOG_INFO ("Create Applications.");
   std::cout << "Source IP address: " <<
ifconf_0_1.GetAddress(0) << std::endl;
   std::cout << "Destination IP address: " <<
ifconf_4_5.GetAddress(1) << std::endl;

   /* QKD APPs for charing */
   //QKD LINK 0_1
   QKDAppChargingHelper
qkdChargingApp_0_1("ns3::TcpSocketFactory
", ifconf_0_1.GetAddress(0),
ifconf_0_1.GetAddress(1), 3072000);
//102400 * 30seconds
```

```
ApplicationContainer qkdChrgApps_0_1 =
qkdChargingApp_0_1.Install (
devices_0_1.Get(0), devices_0_1.Get(1) );
   qkdChrgApps_0_1.Start (Seconds (5.));
   qkdChrgApps_0_1.Stop (Seconds (500.));
   //QKD LINK 0_2
   QKDAppChargingHelper
qkdChargingApp_0_2("ns3::TcpSocketFactory
", ifconf_0_2.GetAddress(0),
ifconf_0_2.GetAddress(1), 3072000);
//102400 * 30seconds
   ApplicationContainer qkdChrgApps_0_2 =
qkdChargingApp_0_2.Install (
devices_0_2.Get(0), devices_0_2.Get(1) );
   qkdChrgApps_0_2.Start (Seconds (5.));
   qkdChrgApps_0_2.Stop (Seconds (500.));
   //QKD LINK 0_3
   QKDAppChargingHelper
qkdChargingApp_0_3("ns3::TcpSocketFactory
", ifconf_0_3.GetAddress(0),
ifconf_0_3.GetAddress(1), 3072000);
//102400 * 30seconds
   ApplicationContainer qkdChrgApps_0_3 =
qkdChargingApp_0_3.Install (
devices_0_3.Get(0), devices_0_3.Get(1) );
   qkdChrgApps_0_3.Start (Seconds (5.));
   qkdChrgApps_0_3.Stop (Seconds (500.));

   //QKD LINK 0_4
   QKDAppChargingHelper
qkdChargingApp_0_4("ns3::TcpSocketFactory
", ifconf_0_4.GetAddress(0),
ifconf_0_4.GetAddress(1), 3072000);
//102400 * 30seconds
   ApplicationContainer qkdChrgApps_0_4 =
qkdChargingApp_0_4.Install (
devices_0_4.Get(0), devices_0_4.Get(1) );
   qkdChrgApps_0_4.Start (Seconds (5.));
   qkdChrgApps_0_4.Stop (Seconds (500.));

   //QKD LINK 0_5
   QKDAppChargingHelper
qkdChargingApp_0_5("ns3::TcpSocketFactory
", ifconf_0_5.GetAddress(0),
ifconf_0_5.GetAddress(1), 3072000);
//102400 * 30seconds
   ApplicationContainer qkdChrgApps_0_5 =
qkdChargingApp_0_5.Install (
devices_0_5.Get(0), devices_0_5.Get(1) );
   qkdChrgApps_0_5.Start (Seconds (5.));
   qkdChrgApps_0_5.Stop (Seconds (500.));
```

```
//QKD LINK 1_2
   QKDAppChargingHelper
qkdChargingApp_1_2("ns3::TcpSocketFactory
", ifconf_1_2.GetAddress(0),
ifconf_1_2.GetAddress(1), 3072000);
//102400 * 30seconds
   ApplicationContainer qkdChrgApps_1_2 =
qkdChargingApp_1_2.Install (
devices_1_2.Get(0), devices_1_2.Get(1) );
   qkdChrgApps_1_2.Start (Seconds (5.));
   qkdChrgApps_1_2.Stop (Seconds (500.));
   //QKD LINK 1_3
   QKDAppChargingHelper
qkdChargingApp_1_3("ns3::TcpSocketFactory
", ifconf_1_3.GetAddress(0),
ifconf_1_3.GetAddress(1), 3072000);
//102400 * 30seconds
   ApplicationContainer qkdChrgApps_1_3 =
qkdChargingApp_1_3.Install (
devices_1_3.Get(0), devices_1_3.Get(1) );
   qkdChrgApps_1_3.Start (Seconds (5.));
   qkdChrgApps_1_3.Stop (Seconds (500.));
   //QKD LINK 1_4
   QKDAppChargingHelper
qkdChargingApp_1_4("ns3::TcpSocketFactory
", ifconf_1_4.GetAddress(0),
ifconf_1_4.GetAddress(1), 3072000);
//102400 * 30seconds
   ApplicationContainer qkdChrgApps_1_4 =
qkdChargingApp_1_4.Install (
devices_1_4.Get(0), devices_1_4.Get(1) );
   qkdChrgApps_1_4.Start (Seconds (5.));
   qkdChrgApps_1_4.Stop (Seconds (500.));
   //QKD LINK 1_5
   QKDAppChargingHelper
qkdChargingApp_1_5("ns3::TcpSocketFactory
", ifconf_1_5.GetAddress(0),
ifconf_1_5.GetAddress(1), 3072000);
//102400 * 30seconds
   ApplicationContainer qkdChrgApps_1_5 =
qkdChargingApp_1_5.Install (
devices_1_5.Get(0), devices_1_5.Get(1) );
   qkdChrgApps_1_5.Start (Seconds (5.));
   qkdChrgApps_1_5.Stop (Seconds (500.));

   //QKD LINK 2_3
   QKDAppChargingHelper
qkdChargingApp_2_3("ns3::TcpSocketFactory
", ifconf_2_3.GetAddress(0),
ifconf_2_3.GetAddress(1), 3072000);
//102400 * 30seconds
ApplicationContainer qkdChrgApps_2_3 =
qkdChargingApp_2_3.Install (
devices_2_3.Get(0), devices_2_3.Get(1) );
   qkdChrgApps_2_3.Start (Seconds (5.));
   qkdChrgApps_2_3.Stop (Seconds (500.));
   //QKD LINK 2_4
   QKDAppChargingHelper
qkdChargingApp_2_4("ns3::TcpSocketFactory
", ifconf_2_4.GetAddress(0),
ifconf_2_4.GetAddress(1), 3072000);
//102400 * 30seconds
   ApplicationContainer qkdChrgApps_2_4 =
qkdChargingApp_2_4.Install (
devices_2_4.Get(0), devices_2_4.Get(1) );
   qkdChrgApps_2_4.Start (Seconds (5.));
   qkdChrgApps_2_4.Stop (Seconds (500.));
   //QKD LINK 2_5
   QKDAppChargingHelper
qkdChargingApp_2_5("ns3::TcpSocketFactory
", ifconf_2_5.GetAddress(0),
ifconf_2_5.GetAddress(1), 3072000);
//102400 * 30seconds
   ApplicationContainer qkdChrgApps_2_5 =
qkdChargingApp_2_5.Install (
devices_2_5.Get(0), devices_2_5.Get(1) );
   qkdChrgApps_2_5.Start (Seconds (5.));
   qkdChrgApps_2_5.Stop (Seconds (500.));

   //QKD LINK 3_4
   QKDAppChargingHelper
qkdChargingApp_3_4("ns3::TcpSocketFactory
", ifconf_3_4.GetAddress(0),
ifconf_3_4.GetAddress(1), 3072000);
//102400 * 30seconds
   ApplicationContainer qkdChrgApps_3_4 =
qkdChargingApp_3_4.Install (
devices_3_4.Get(0), devices_3_4.Get(1) );
   qkdChrgApps_3_4.Start (Seconds (5.));
   qkdChrgApps_3_4.Stop (Seconds (500.));

   //QKD LINK 3_5
   QKDAppChargingHelper
qkdChargingApp_3_5("ns3::TcpSocketFactory
", ifconf_3_5.GetAddress(0),
ifconf_3_5.GetAddress(1), 3072000);
//102400 * 30seconds
   ApplicationContainer qkdChrgApps_3_5 =
qkdChargingApp_3_5.Install (
devices_3_5.Get(0), devices_3_5.Get(1) );
   qkdChrgApps_3_5.Start (Seconds (5.));
   qkdChrgApps_3_5.Stop (Seconds (500.));
```

```
//QKD LINK 4_5
   QKDAppChargingHelper
qkdChargingApp_4_5("ns3::TcpSocketFactory
", ifconf_4_5.GetAddress(0),
ifconf_4_5.GetAddress(1), 3072000);
//102400 * 30seconds
   ApplicationContainer qkdChrgApps_4_5 =
qkdChargingApp_4_5.Install (
devices_4_5.Get(0), devices_4_5.Get(1) );
   qkdChrgApps_4_5.Start (Seconds (5.));
   qkdChrgApps_4_5.Stop (Seconds (500.));


   Ptr<QKDSend> app =
CreateObject<QKDSend> ();
   if(enableApplication){
      /* Create user's traffic between v0 and
v1 */
      /* Create sink app */
      uint16_t sinkPort = 8080;
      QKDSinkAppHelper packetSinkHelper
("ns3::UdpSocketFactory", InetSocketAddress
(Ipv4Address::GetAny (), sinkPort));
      ApplicationContainer sinkApps =
packetSinkHelper.Install (nodes.Get (5));
      sinkApps.Start (Seconds (15.));
      sinkApps.Stop (Seconds (500.));

      /* Create source app */
      Address sinkAddress (InetSocketAddress
(ifconf_0_5.GetAddress(1), sinkPort));
      Address sourceAddress
(InetSocketAddress
(ifconf_0_1.GetAddress(0), sinkPort));
      Ptr<Socket> socket =
Socket::CreateSocket (nodes.Get (0),
UdpSocketFactory::GetTypeId ());
app->Setup (socket, sourceAddress,
sinkAddress, 512, 0, DataRate ("160kbps"));
      nodes.Get (0)->AddApplication (app);
      app->SetStartTime (Seconds (15.));
      app->SetStopTime (Seconds (500.));
   }
//if we need we can create pcap files
   p2p.EnablePcapAll ("QKD_netsim_test");
   //QHelper.EnablePcapAll
("QKD_netsim_test_Qhelper");

Config::Connect("/NodeList/*/ApplicationList
/*/$ns3::QKDSink/Rx",
MakeCallback(&ReceivedPacket));

   //
   // Now, do the actual simulation.
   //
   NS_LOG_INFO ("Run Simulation.");

   //AnimationInterface
anim("secoqc_olsr_wifi.xml");

AnimationInterface anim
("animsixmesh.xml");
  anim.SetConstantPosition(nodes.Get(0), 0.0,
200.0, 0.0);
  anim.SetConstantPosition(nodes.Get(1),
100.0, 0.0, 0.0);
  anim.SetConstantPosition(nodes.Get(2),
300.0, 0.0, 0.0);
  anim.SetConstantPosition(nodes.Get(3),
500.0, 200.0, 0.0);
  anim.SetConstantPosition(nodes.Get(4),
300.0, 400.0, 0.0);
  anim.SetConstantPosition(nodes.Get(5),
100.0, 400.0, 0.0);
   Simulator::Stop (Seconds(simulationTime));
   Simulator::Run ();

   if(enableApplication)
      Ratio(app->sendDataStats(), app-
>sendPacketStats());

   //Finally print the graphs
   QHelper.PrintGraphs();
   Simulator::Destroy ();
}
```

- <u>Output</u>

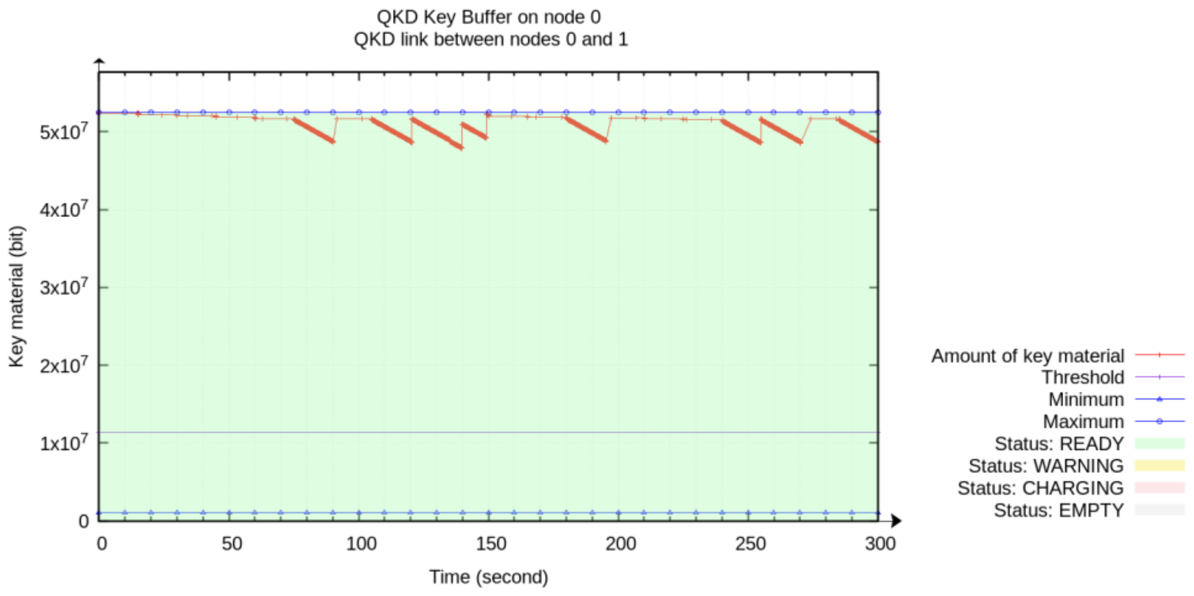


Fig.23: QKD buffer capacity between nodes 0&1

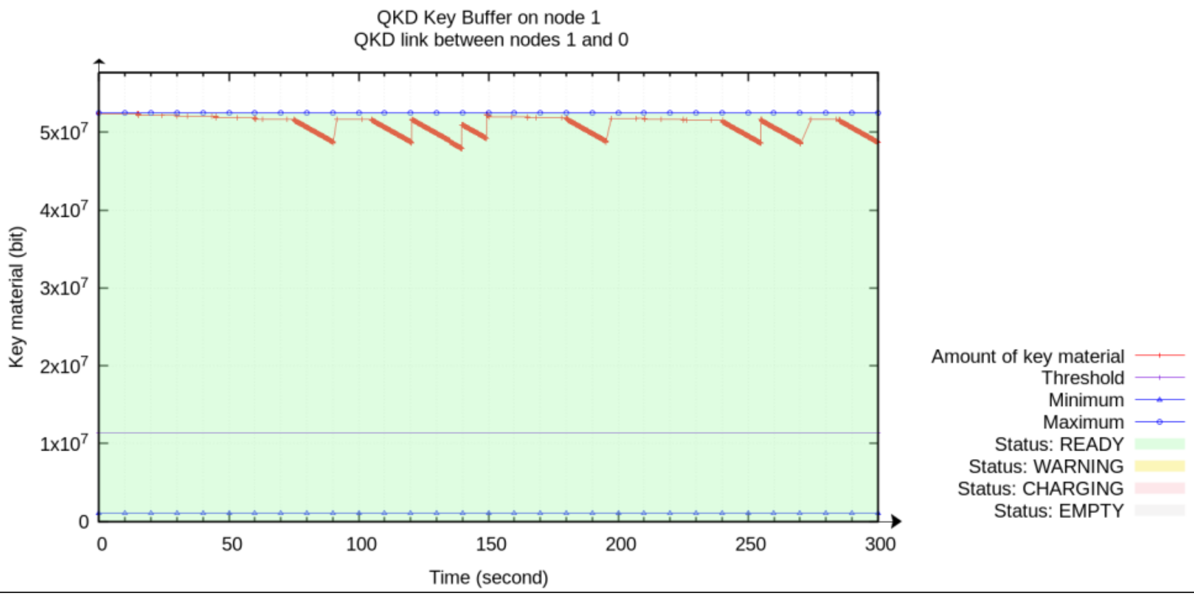


Fig.24: QKD buffer capacity between nodes 0&1

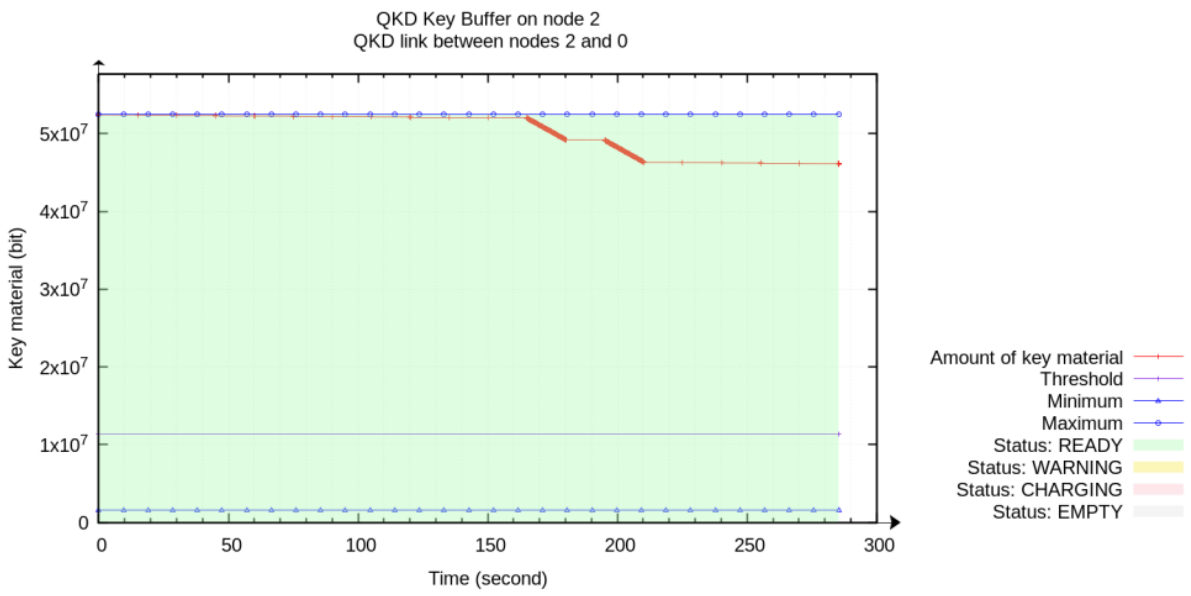


Fig.24: QKD buffer capacity between nodes 2&0

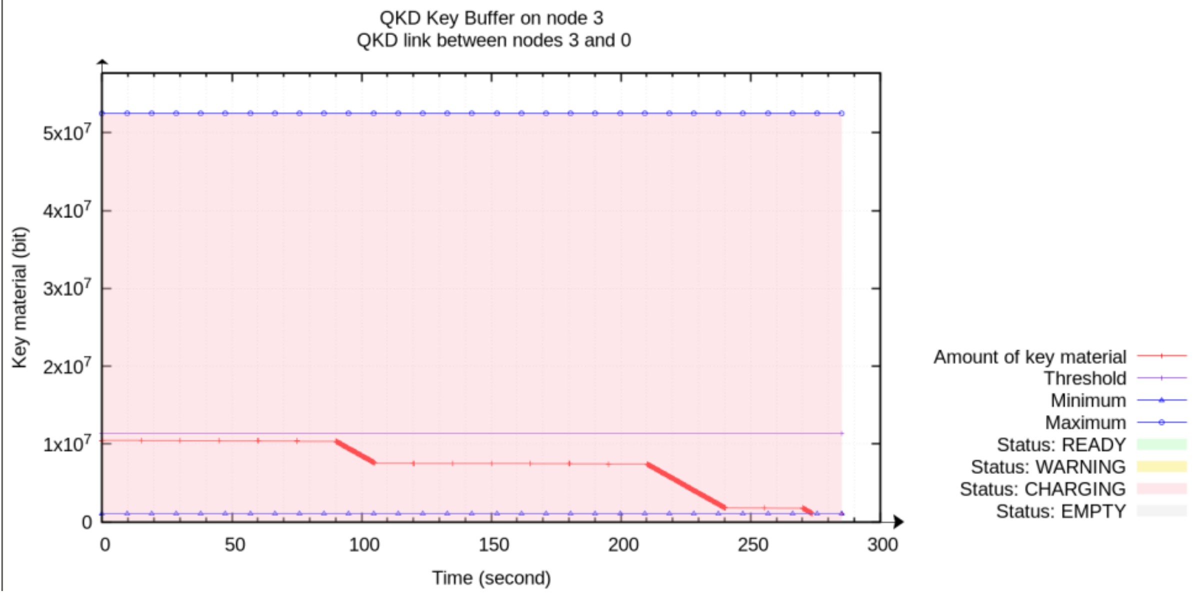


Fig.24: QKD buffer capacity between nodes 3&0

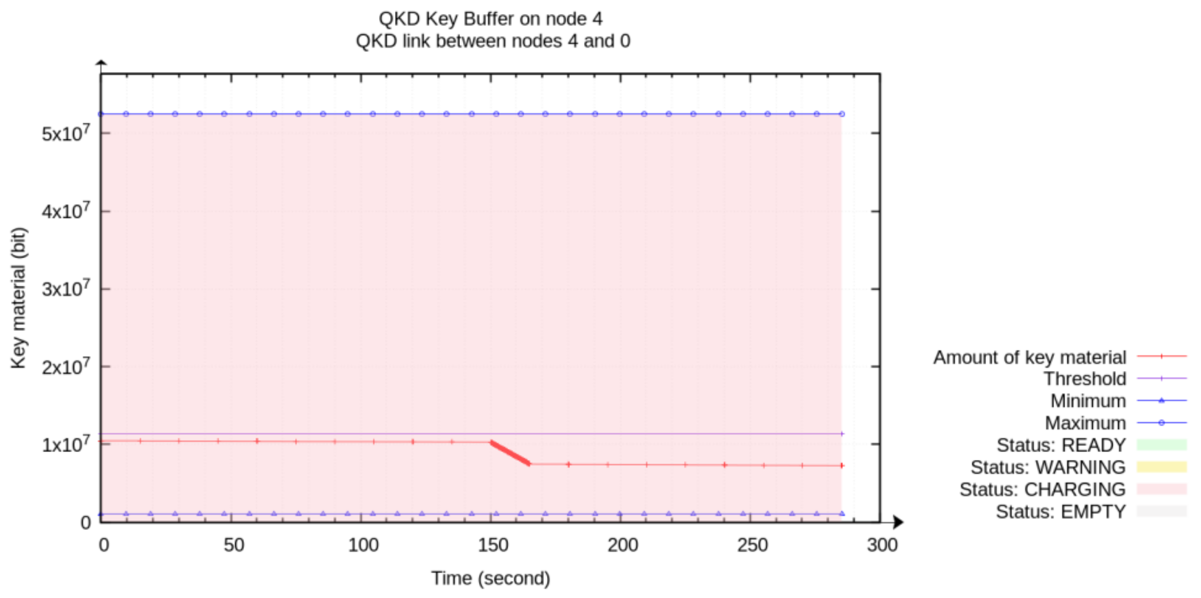


Fig.25: QKD buffer capacity between nodes 4&0

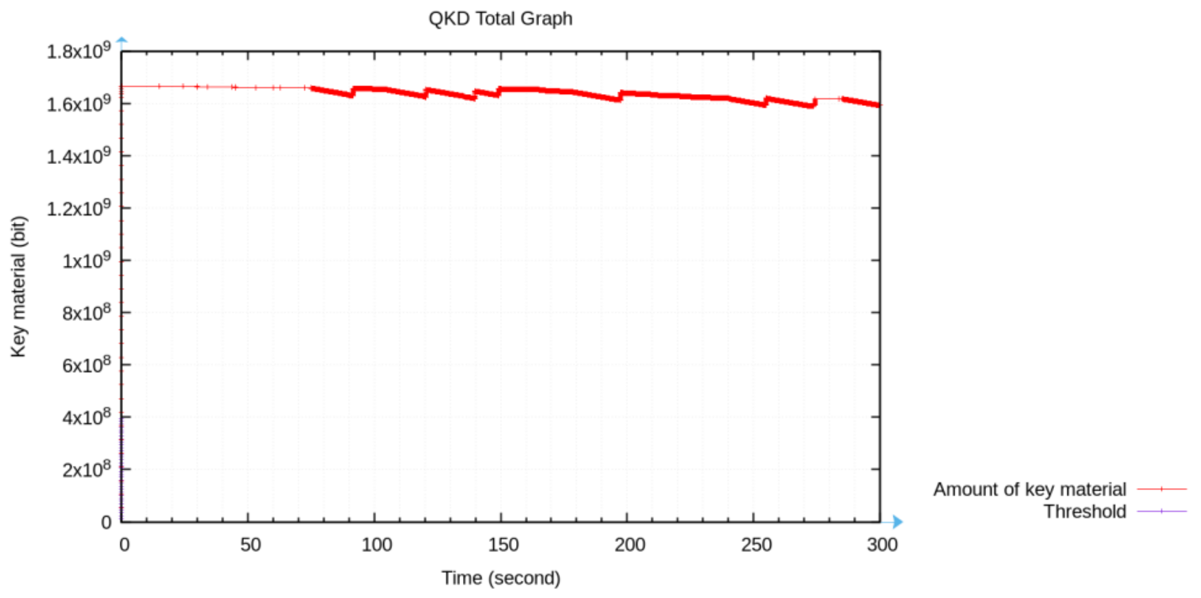


Fig.26: Total graph for QKD buffer capacity

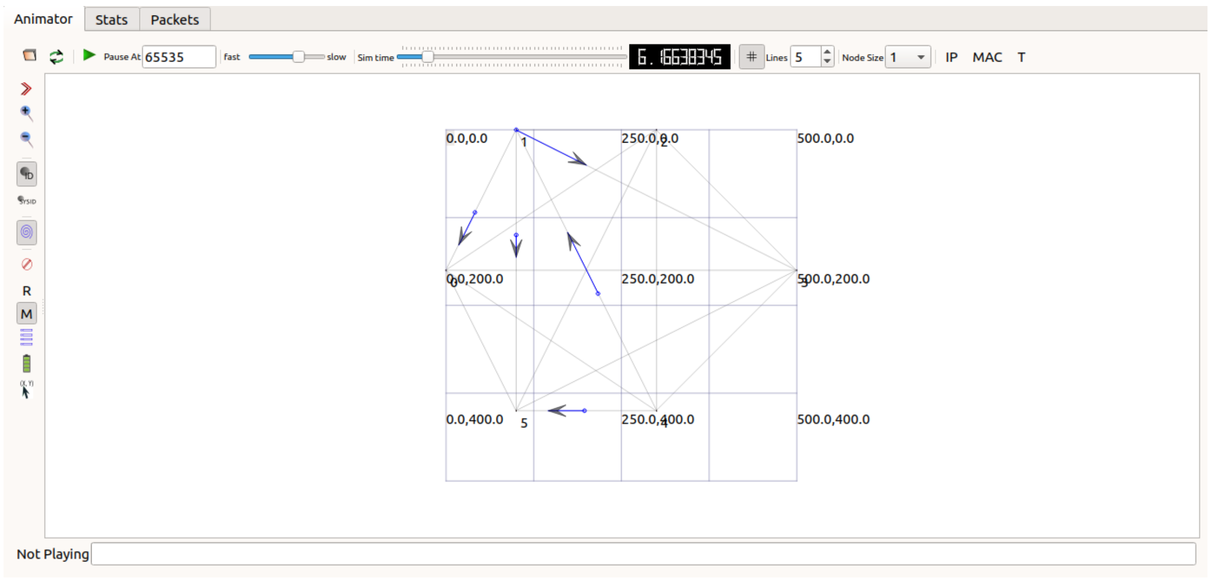
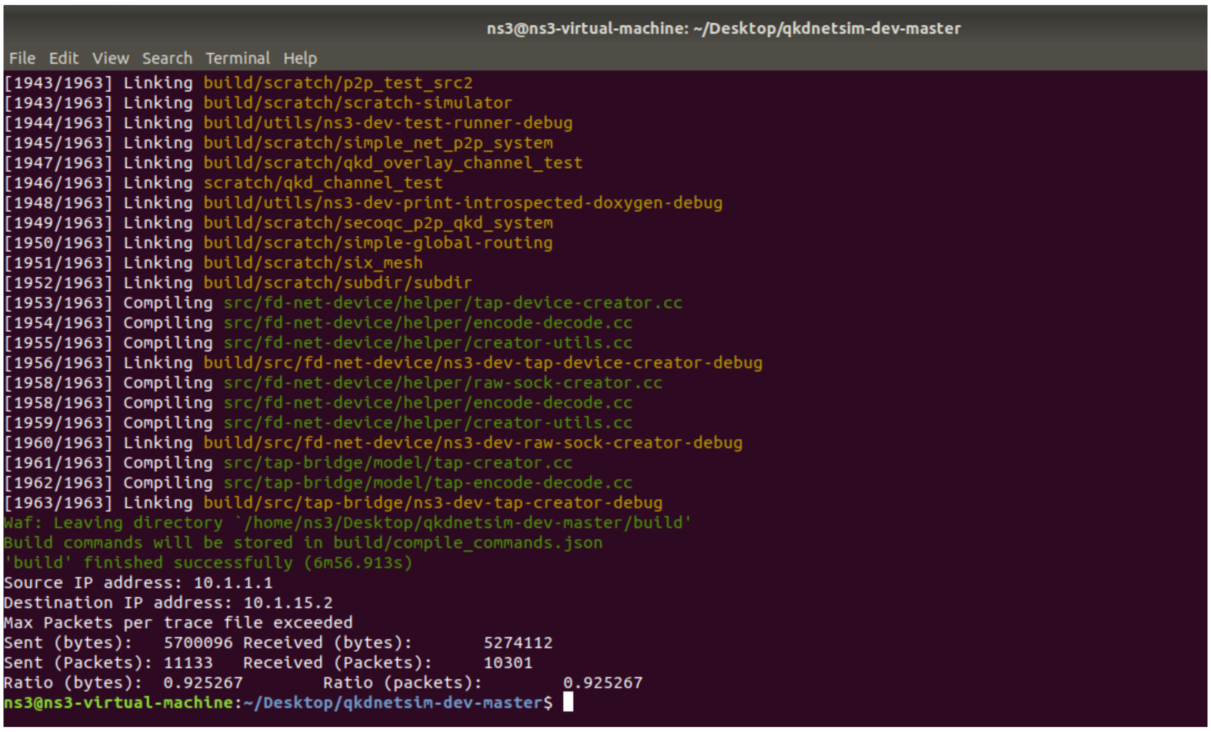
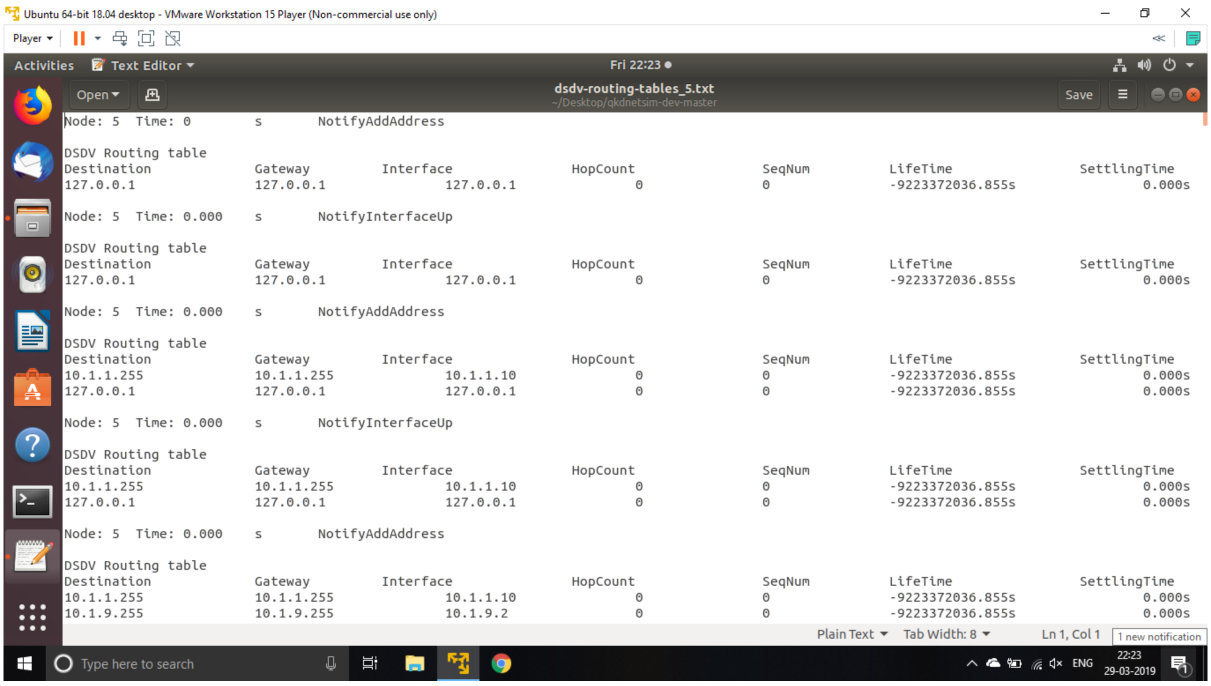


Fig.27: Network Topology



```
ns3@ns3-virtual-machine: ~/Desktop/qkdnetsim-dev-master
File  Edit  View  Search  Terminal  Help
[1943/1963] Linking build/scratch/p2p_test_src2
[1943/1963] Linking build/scratch/scratch-simulator
[1944/1963] Linking build/utils/ns3-dev-test-runner-debug
[1945/1963] Linking build/scratch/simple_net_p2p_system
[1947/1963] Linking build/scratch/qkd_overlay_channel_test
[1946/1963] Linking scratch/qkd_channel_test
[1948/1963] Linking build/utils/ns3-dev-print-introspected-doxygen-debug
[1949/1963] Linking build/scratch/secoqc_p2p_qkd_system
[1950/1963] Linking build/scratch/simple-global-routing
[1951/1963] Linking build/scratch/six_mesh
[1952/1963] Linking build/scratch/subdir/subdir
[1953/1963] Compiling src/fd-net-device/helper/tap-device-creator.cc
[1954/1963] Compiling src/fd-net-device/helper/encode-decode.cc
[1955/1963] Compiling src/fd-net-device/helper/creator-utils.cc
[1956/1963] Linking build/src/fd-net-device/ns3-dev-tap-device-creator-debug
[1958/1963] Compiling src/fd-net-device/helper/raw-sock-creator.cc
[1958/1963] Compiling src/fd-net-device/helper/encode-decode.cc
[1959/1963] Compiling src/fd-net-device/helper/creator-utils.cc
[1960/1963] Linking build/src/fd-net-device/ns3-dev-raw-sock-creator-debug
[1961/1963] Compiling src/tap-bridge/model/tap-creator.cc
[1962/1963] Compiling src/tap-bridge/model/tap-encode-decode.cc
[1963/1963] Linking build/src/tap-bridge/ns3-dev-tap-creator-debug
Waf: Leaving directory `/home/ns3/Desktop/qkdnetsim-dev-master/build'
Build commands will be stored in build/compile_commands.json
'build' finished successfully (6m56.913s)
Source IP address: 10.1.1.1
Destination IP address: 10.1.15.2
Max Packets per trace file exceeded
Sent (bytes):   5700096 Received (bytes):       5274112
Sent (Packets): 11133   Received (Packets):     10301
Ratio (bytes):  0.925267        Ratio (packets):        0.925267
ns3@ns3-virtual-machine:~/Desktop/qkdnetsim-dev-master$ 
```

Fig.28: Network data transfer reading


Ubuntu 64-bit 18.04 desktop - VMware Workstation 15 Player (Non-commercial use only)
Player
Activities
Text Editor
Fri 22:23
dsdv-routing-tables_5.txt
~/Desktop/qkdnetsim-dev-master
Open
Save
Plain Text
Tab Width: 8
Ln 1, Col 1
1 new notification
Type here to search
ENG
22:23
29-03-2019

```
Node: 5  Time: 0        s       NotifyAddAddress

DSDV Routing table
Destination             Gateway         Interface               HopCount                SeqNum          LifeTime                SettlingTime
127.0.0.1               127.0.0.1               127.0.0.1               0               0               -9223372036.855s                0.000s

Node: 5  Time: 0.000    s       NotifyInterfaceUp

DSDV Routing table
Destination             Gateway         Interface               HopCount                SeqNum          LifeTime                SettlingTime
127.0.0.1               127.0.0.1               127.0.0.1               0               0               -9223372036.855s                0.000s

Node: 5  Time: 0.000    s       NotifyAddAddress

DSDV Routing table
Destination             Gateway         Interface               HopCount                SeqNum          LifeTime                SettlingTime
10.1.1.255              10.1.1.255              10.1.1.10               0               0               -9223372036.855s                0.000s
127.0.0.1               127.0.0.1               127.0.0.1               0               0               -9223372036.855s                0.000s

Node: 5  Time: 0.000    s       NotifyInterfaceUp

DSDV Routing table
Destination             Gateway         Interface               HopCount                SeqNum          LifeTime                SettlingTime
10.1.1.255              10.1.1.255              10.1.1.10               0               0               -9223372036.855s                0.000s
127.0.0.1               127.0.0.1               127.0.0.1               0               0               -9223372036.855s                0.000s

Node: 5  Time: 0.000    s       NotifyAddAddress

DSDV Routing table
Destination             Gateway         Interface               HopCount                SeqNum          LifeTime                SettlingTime
10.1.1.255              10.1.1.255              10.1.1.10               0               0               -9223372036.855s                0.000s
10.1.9.255              10.1.9.255              10.1.9.2                0               0               -9223372036.855s                0.000s
```

Fig.29: Node 5 routing table