```
/*Write a program to create 3 p2p duplex nodes. Set queue size, vary bandwidth and
find the number of packets dropped. */

#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"

// Network topology
//
//       10.1.1.0       10.1.2.0
// n0 -------------- n1..........n2
//    point-to-point
//

using namespace ns3;

int main(int argc, char* argv[])
{

    std::string socketType = "ns3::TcpSocketFactory";;

/*The above code creates a new string variable called socketType and initializes it
with the value "ns3::TcpSocketFactory". Basically socketType is now an alias for
"ns3::TcpSocketFactory".  This line can be avoided by usin"ns3::TcpSocketFactory"
wherever we have used socketType in the program.*/

    CommandLine cmd;
    cmd.Parse(argc, argv);
    /*This code snippet parses command-line arguments in a C++ program using the
    CommandLine class from the NS-3 network simulator.*/

    NodeContainer nodes;
    nodes.Create(3);
    // this code segment creates a container to hold nodes called node and creates 3
    //nodes in it.

    InternetStackHelper stack;
    stack.Install(nodes);
/* this code segment creates an object called stack of type InternetStackHelper
which is a class used to install Internet Protocals on objects. the IPs are being
installed on all the nodes held in the NodeContainer object. */

    PointToPointHelper p2p1;
    p2p1.SetDeviceAttribute("DataRate", StringValue("5Mbps"));
    p2p1.SetChannelAttribute("Delay", StringValue("1ms"));
    /* Creates a PointToPointHelper object called p2p1 and sets the attributes of the
     devices and the channel it will create. */

    Ipv4AddressHelper address;
    address.SetBase("10.1.1.0", "255.255.255.0");
 /* Creates an Ipv4AddressHelper object called address and sets the base address
 and mask address it will use to assign IP addresses to the devices it will create. */

    NetDeviceContainer devices;
    devices = p2p1.Install(nodes.Get(0), nodes.Get(1));
    Ipv4InterfaceContainer interfaces = address.Assign(devices);
 /* Creates a NetDeviceContainer object called devices and installs point-to-point
    devices on the first and second nodes in the nodes container using the p2p1
object. It then assigns IP addresses to the devices using the address object and
stores the interfaces in the interfaces container. */
```

```
    devices = p2p1.Install(nodes.Get(1), nodes.Get(2));
    address.SetBase("10.1.2.0", "255.255.255.0");
    interfaces = address.Assign(devices);
/*Installs point-to-point devices on the second and third nodes in the nodes container
using the p2p1 object, and assigns IP addresses to the devices using the address
object with a different base address and mask. The interfaces are stored in the
interfaces container.*/

    Ptr<RateErrorModel> em = CreateObject<RateErrorModel>();
/*This code creates an object of the RateErrorModel class and stores a smart pointer
to the object in the variable em.
The RateErrorModel class is a type of error model that can be used to simulate bit
errors in a communication link. It allows you to specify the bit error rate (BER)
for the link,which is the probability that a bit will be transmitted incorrectly.
The CreateObject<RateErrorModel> function is used to create a new object of the
RateErrorModel class and return a smart pointer to it.*/

     //Introduce error model to drop packets
    em->SetAttribute("ErrorRate", DoubleValue(0.00002));
    devices.Get(1)->SetAttribute("ReceiveErrorModel", PointerValue(em));
/*The SetAttribute function is a generic function that is used to set an attribute
of an object to a given value. The first argument is the name of the attribute to be
set,and the second argument is the value to be set.
In this case, the attribute being set is called "ReceiveErrorModel" and the value
being set is the smart pointer to the RateErrorModel object stored in the em variable.
*/

    Ipv4GlobalRoutingHelper::PopulateRoutingTables();
/*The Ipv4GlobalRoutingHelper class is a helper class that provides a convenient
interface for configuring and installing global routing protocols on a simulation
topology. In ns-3, global routing refers to the process of building routing tables
that allow packets to be forwarded between nodes in a network, regardless of their
location within the network. The PopulateRoutingTables method is a static method of
this class that creates and installs global routing tables on all nodes in the
simulation. */


    uint32_t payloadSize = 1448; // setting payload size or packet size.
    OnOffHelper onoff(socketType, Ipv4Address::GetAny());
/*The GetAny() method of the Ipv4Address class returns a special IP address that
represents any available IP address.
This code snippet creates an instance of the OnOffHelper class and initializes
it with a socket type and an IP address. The OnOffHelper class is a utility that
helps to set up On/Off traffic flows, which are  useful for simulating network
traffic with defined patterns of activity.
The socketType parameter specifies the type of socket to use for the On/Off traffic
flow.  */

    //Generate traffic
    onoff.SetAttribute("OnTime",
StringValue("ns3::ConstantRandomVariable[Constant=1]"));
    onoff.SetAttribute("OffTime",
StringValue("ns3::ConstantRandomVariable[Constant=0]"));
    onoff.SetAttribute("PacketSize", UintegerValue(payloadSize));
    onoff.SetAttribute("DataRate", StringValue("50Mbps")); //bit/s
    uint16_t port = 7;

    //1. Install receiver (for packetsink) on node 2
    Address localAddress1(InetSocketAddress(Ipv4Address::GetAny(), port));
    // InetSocketAddress class takes an Ipv4Address and port number. it is used to
    // represent an IP socket address.
    // the IP socket address is the address now stored in localAddress1.
```

```
    PacketSinkHelper packetSinkHelper1(socketType, localAddress1);
    // this creates a sink which uses tcp protocol in the node in the
    // localAddress1.
    ApplicationContainer sinkApp1 = packetSinkHelper1.Install(nodes.Get(2));
    //adds Packet Sink Application to node 2 and adds it to ApplicationContainer
sinkApp1
    sinkApp1.Start(Seconds(0.0));
    sinkApp1.Stop(Seconds(10));
/* The result of this code snippet is an ApplicationContainer object called "sinkApp1"
that holds the PacketSink application that was installed on the third node. You can
use the ApplicationContainer object to start and stop the PacketSink application.
This code snippet creates a packet sink application and installs it on a node in
a simulated network. A packet sink is a network application that receives packets
sent to it and does not transmit any packets. It is often used as a way to terminate
traffic flows in a simulated network. */

    //2. Install sender app on node 0
    ApplicationContainer apps;
    AddressValue remoteAddress(InetSocketAddress(interfaces.GetAddress(1), port));
 // Assigns the address of interface at node(1) and the port number to remoteAddress

    onoff.SetAttribute("Remote", remoteAddress);
    //The "remote" attribute of the OnOffHelper class specifies the address and
    //port of the destination to which the OnOffApplication will send packets.
    apps.Add(onoff.Install(nodes.Get(0)));

    // installs On/Off application on Node 0 and adds it to ApplicationContainer apps.
    apps.Start(Seconds(1.0));
    apps.Stop(Seconds(10));
    // The OnOffApplication is started at time 1.0 seconds.
    //The OnOffApplication is stopped at time 10 seconds.

    Simulator::Stop(Seconds(10));

    //Generate trace file
    AsciiTraceHelper ascii;
 /*The AsciiTraceHelper class is a helper class in the ns-3 networking simulation
 library that provides a convenient interface for enabling ASCII trace output
 in a simulation. ASCII trace output is a type of output that can be generated by ns-3
simulations, which consists of human-readable ASCII text that describes the events
that occur during a simulation. */

    p2p1.EnableAsciiAll(ascii.CreateFileStream("P2ptracefile.tr"));
 /*This code creates an AsciiTraceHelper object and enables ASCII trace output
 on all objects in the p2p1 container using the EnableAsciiAll method.
 The CreateFileStream method of the AsciiTraceHelper class is called to create
 a file stream to which the trace output will be written.
 The EnableAsciiAll method is used to enable ASCII trace output on all objects
 in a container. It takes as an argument a stream to which the trace output
 will be written. In this case, the CreateFileStream method is used to create a file
stream that writes to the file "P2ptracefile.tr".
The CreateFileStream method of the AsciiTraceHelper class is a method that
creates a stream to which ASCII trace output can be written.
The CreateFileStream method takes as an argument the name of the file to which the
trace output should be written, and returns a stream object that can be used to write
to that file. */

     //Run the simulator
    Simulator::Run();
    Simulator::Destroy();
    return 0;
}
```