Homework 3

Introduction

- Homework-3 is about the hidden terminal problem

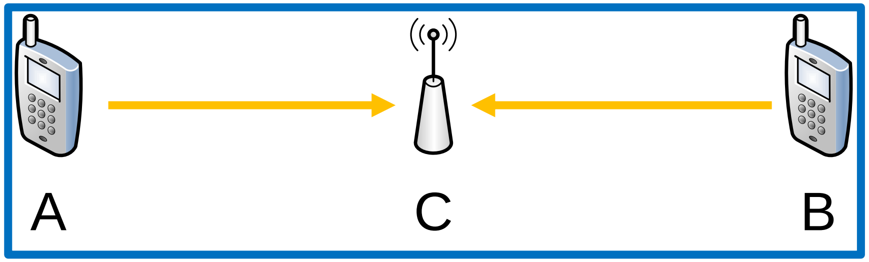
- In the lecture-9, we've learned that the hidden terminal problem may degrades the WLANs' performance

- In here, we will deep dive into the hidden terminal problem using the ns-3 simulator

- Capture the simulation results of all problems and explain the results sufficiently

☞ **This is not a programming homework**

Simulation setting



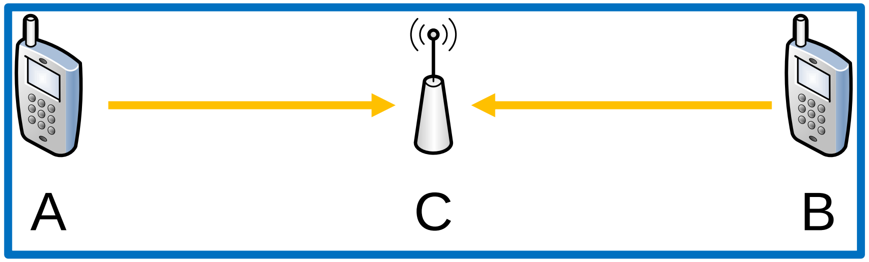
Simulation environment

- One AP and Two associated stations
- Data traffic rate: 70 Mbps
- Ideal rate control
- Dual slope path loss channel model
- Simulation time: 10 seconds
- Metric: Throughput (Mbps)

► Source code can be obtained from ETL

(Place the script file in the scratch directory)

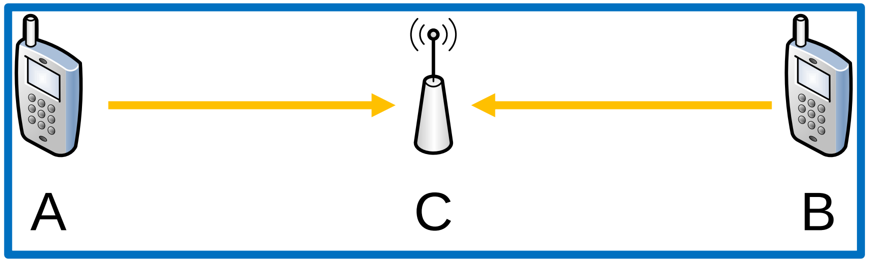
Report



Simulation environment

- Problem 1
  - Compare the throughput when RTS-CTS is enabled and disabled with default parameter setting
- Problem 2
  - Set tx power of sta A as 7 dBm, then compare the throughput with / without RTS-CTS
- Problem 3
  - Set tx power of both STA A and B as 7 dBM, then compare the throughput with / without RTS-CTS

Report



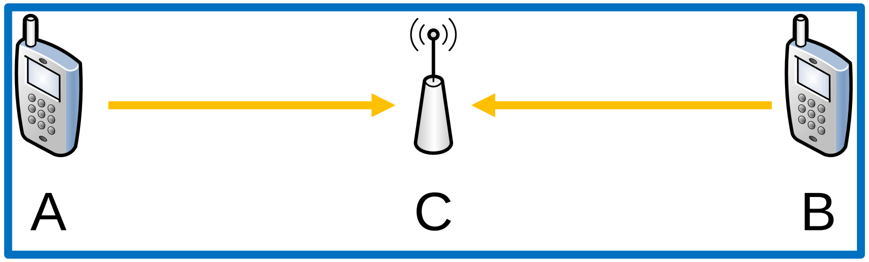
Simulation environment

- Problem 4
  - Set tx power of both STA A and B as 10 dBM, then compare the throughput with / without RTS-CTS

- Problem 5
  - Set CCA-ED threshold of both sta A and B as -70 dBm, then compare the throughput with / without RTS-CTS

- Problem 6
  - Set CCA-ED threshold of stations as -70 dBm, and set tx power of stations as 10 dBm, then compare the throughput with / without RTS-CTS

Report

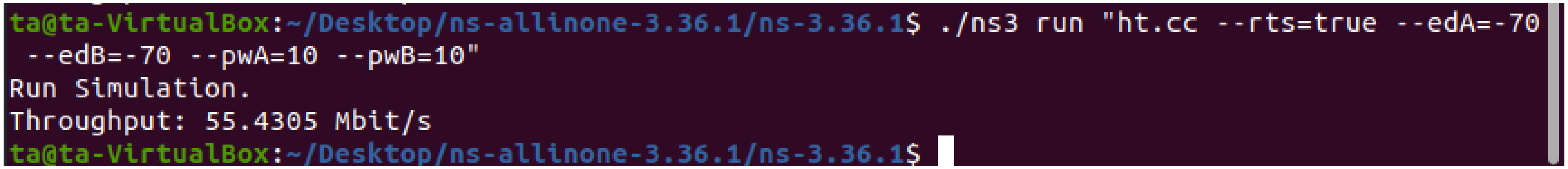


Simulation environment

- Problem 7
  - Nowadays, 802.11 (Wi-Fi) APs are densely deployed in congested areas such as schools and department stores to meet users' increasing demands. Do you think it will be helpful to enable RTS-CTS in such dense WLAN environments? Please fully explain your answer.

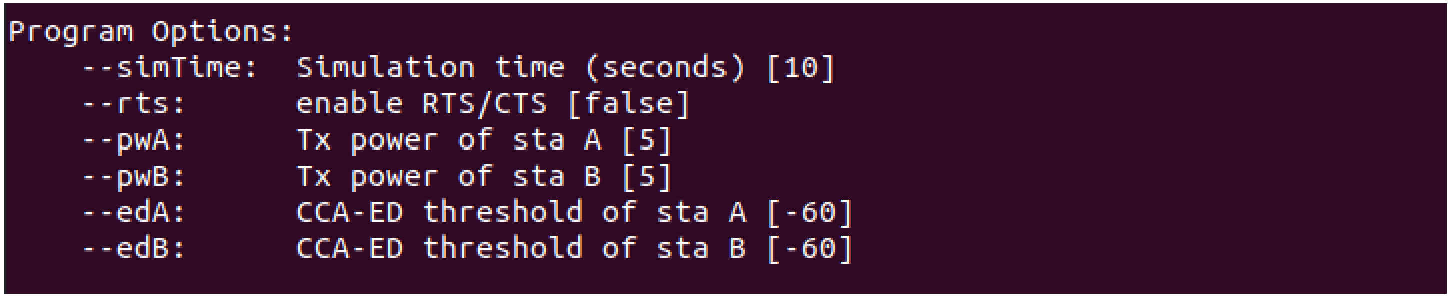
ht.cc

- Usage



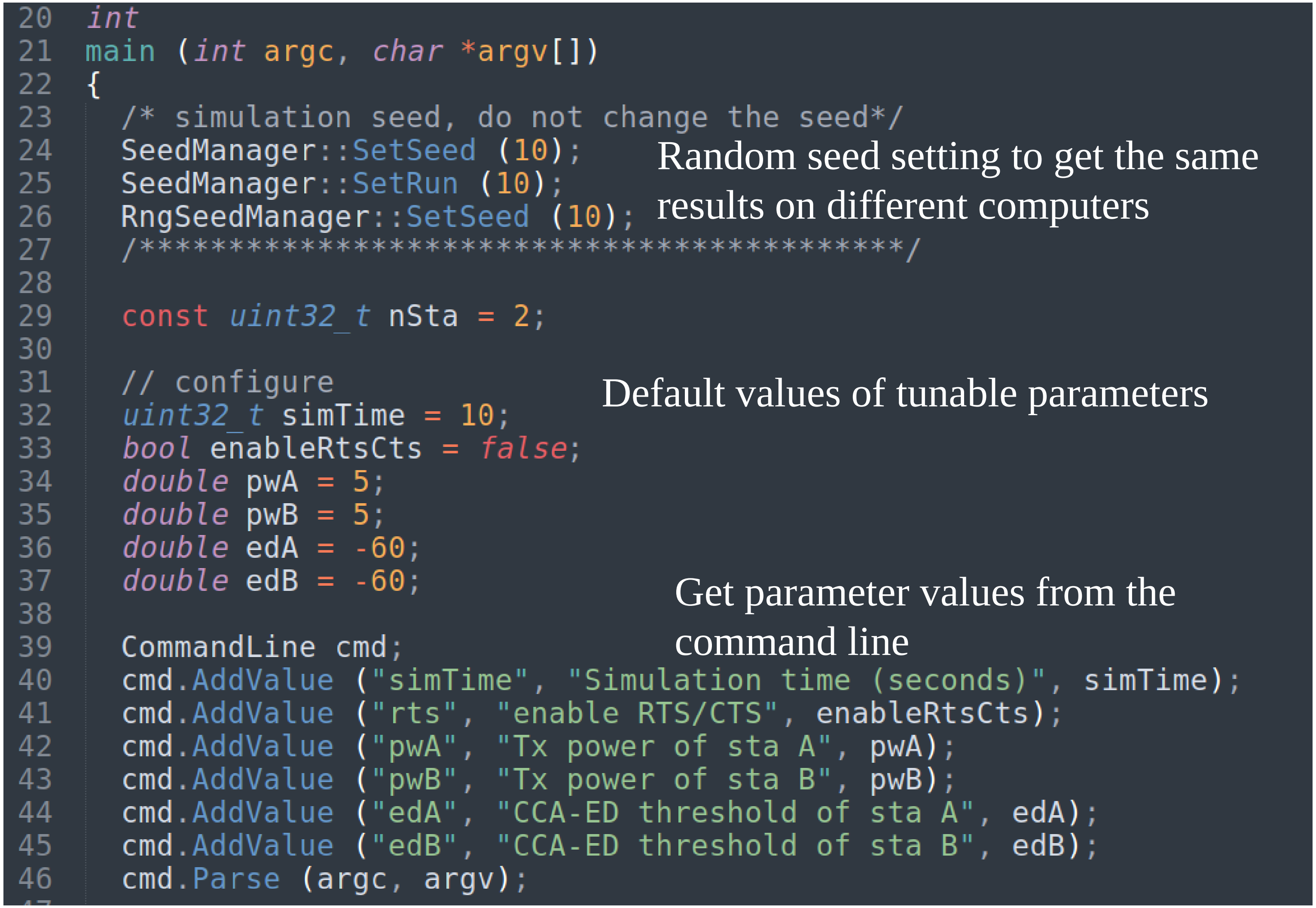
```
ta@ta-VirtualBox:~/Desktop/ns-allinone-3.36.1/ns-3.36.1$ ./ns3 run "ht.cc --rts=true --edA=-70
 --edB=-70 --pwA=10 --pwB=10"
Run Simulation.
Throughput: 55.4305 Mbit/s
ta@ta-VirtualBox:~/Desktop/ns-allinone-3.36.1/ns-3.36.1$
```

- Tunable parameters



```
Program Options:
    --simTime:  Simulation time (seconds) [10]
    --rts:      enable RTS/CTS [false]
    --pwA:      Tx power of sta A [5]
    --pwB:      Tx power of sta B [5]
    --edA:      CCA-ED threshold of sta A [-60]
    --edB:      CCA-ED threshold of sta B [-60]
```

ht.cc



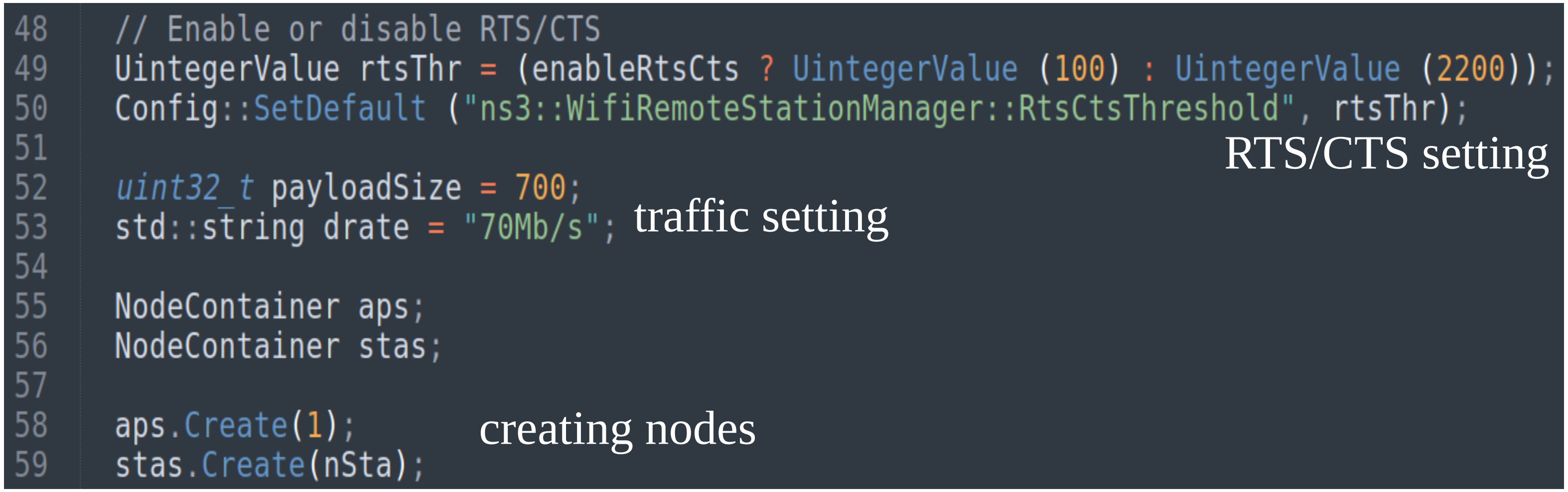
```
int
main (int argc, char *argv[])
{
  /* simulation seed, do not change the seed*/
  SeedManager::SetSeed (10);
  SeedManager::SetRun (10);
  RngSeedManager::SetSeed (10);
  /*******************************************/

  const uint32_t nSta = 2;

  // configure
  uint32_t simTime = 10;
  bool enableRtsCts = false;
  double pwA = 5;
  double pwB = 5;
  double edA = -60;
  double edB = -60;

  CommandLine cmd;
  cmd.AddValue ("simTime", "Simulation time (seconds)", simTime);
  cmd.AddValue ("rts", "enable RTS/CTS", enableRtsCts);
  cmd.AddValue ("pwA", "Tx power of sta A", pwA);
  cmd.AddValue ("pwB", "Tx power of sta B", pwB);
  cmd.AddValue ("edA", "CCA-ED threshold of sta A", edA);
  cmd.AddValue ("edB", "CCA-ED threshold of sta B", edB);
  cmd.Parse (argc, argv);
```

Random seed setting to get the same results on different computers

Default values of tunable parameters

Get parameter values from the command line

ht.cc



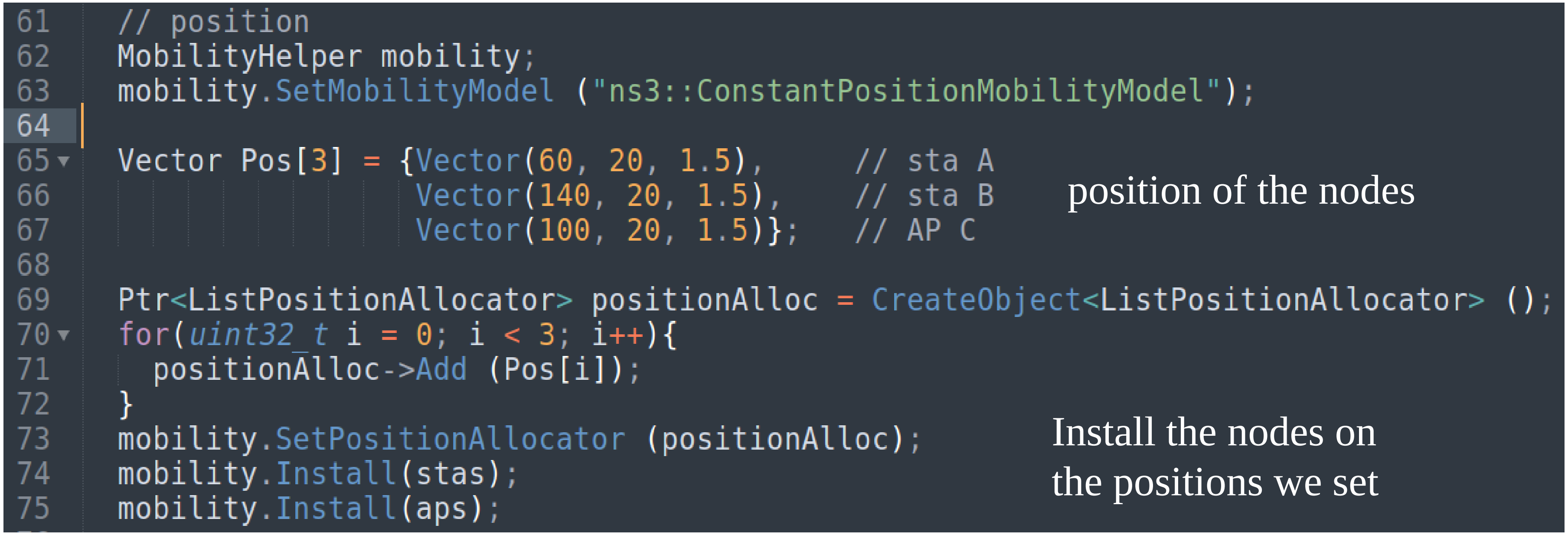
```
// Enable or disable RTS/CTS
UintegerValue rtsThr = (enableRtsCts ? UintegerValue (100) : UintegerValue (2200));
Config::SetDefault ("ns3::WifiRemoteStationManager::RtsCtsThreshold", rtsThr);

uint32_t payloadSize = 700;
std::string drate = "70Mb/s";

NodeContainer aps;
NodeContainer stas;

aps.Create(1);
stas.Create(nSta);
```

RTS/CTS setting

traffic setting

creating nodes

ht.cc

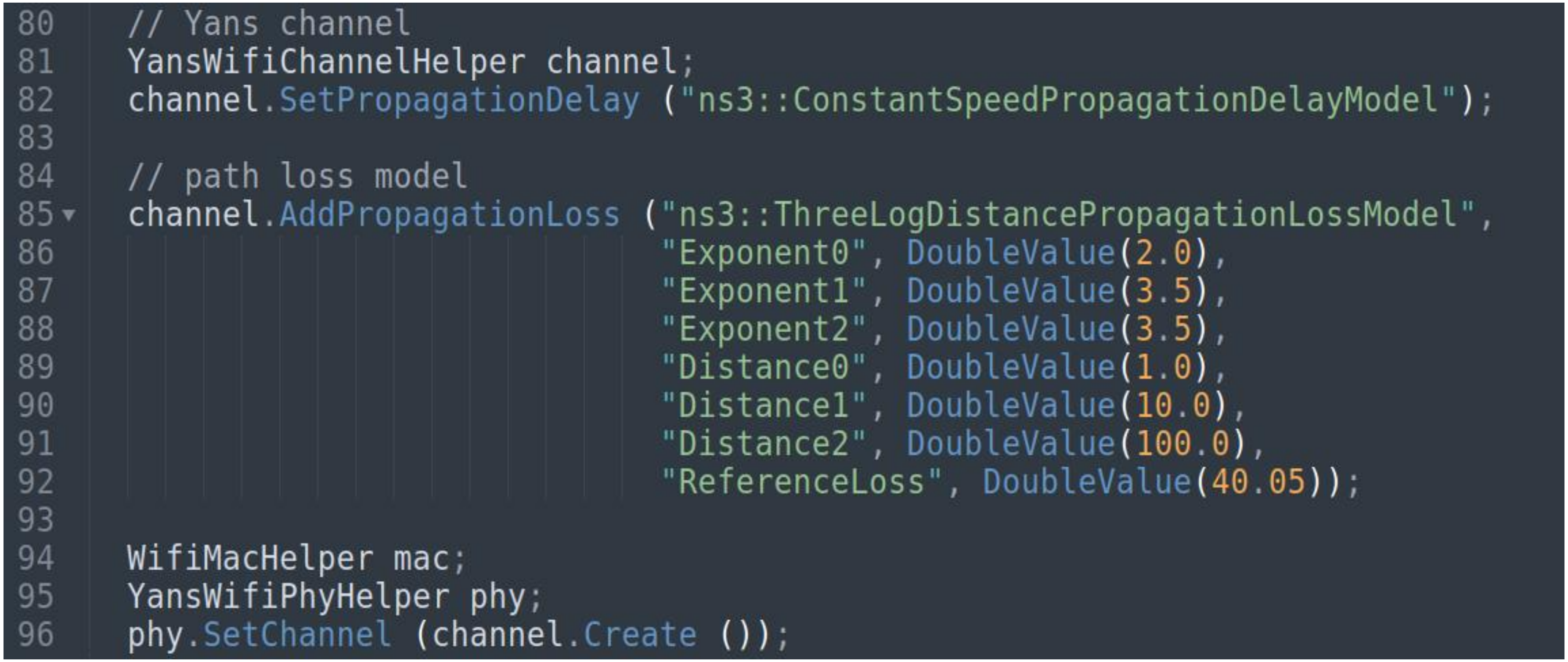


```
// position
MobilityHelper mobility;
mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");

Vector Pos[3] = {Vector(60, 20, 1.5),     // sta A
                 Vector(140, 20, 1.5),    // sta B
                 Vector(100, 20, 1.5)};   // AP C

Ptr<ListPositionAllocator> positionAlloc = CreateObject<ListPositionAllocator> ();
for(uint32_t i = 0; i < 3; i++){
  positionAlloc->Add (Pos[i]);
}
mobility.SetPositionAllocator (positionAlloc);
mobility.Install(stas);
mobility.Install(aps);
```

position of the nodes

Install the nodes on
the positions we set

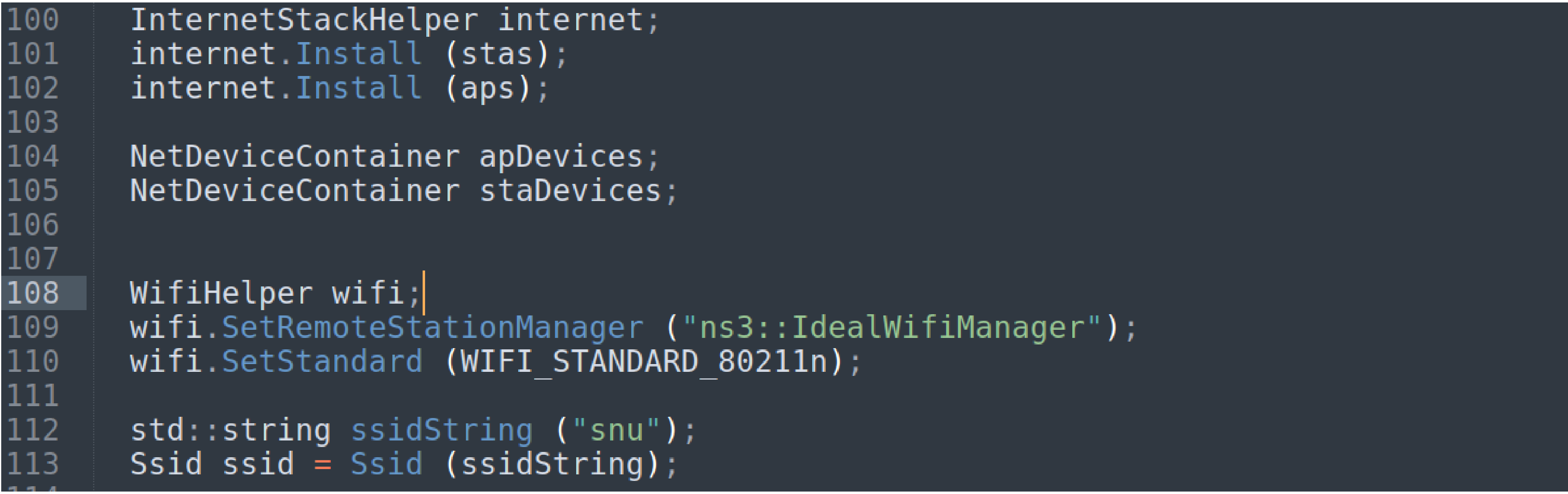
ht.cc



```
// Yans channel
YansWifiChannelHelper channel;
channel.SetPropagationDelay ("ns3::ConstantSpeedPropagationDelayModel");

// path loss model
channel.AddPropagationLoss ("ns3::ThreeLogDistancePropagationLossModel",
                            "Exponent0", DoubleValue(2.0),
                            "Exponent1", DoubleValue(3.5),
                            "Exponent2", DoubleValue(3.5),
                            "Distance0", DoubleValue(1.0),
                            "Distance1", DoubleValue(10.0),
                            "Distance2", DoubleValue(100.0),
                            "ReferenceLoss", DoubleValue(40.05));

WifiMacHelper mac;
YansWifiPhyHelper phy;
phy.SetChannel (channel.Create ());
```

- YansWifiChannelHelper helps you configure the channel model
- Propagation speed is the speed of the light and the path loss model has two slopes

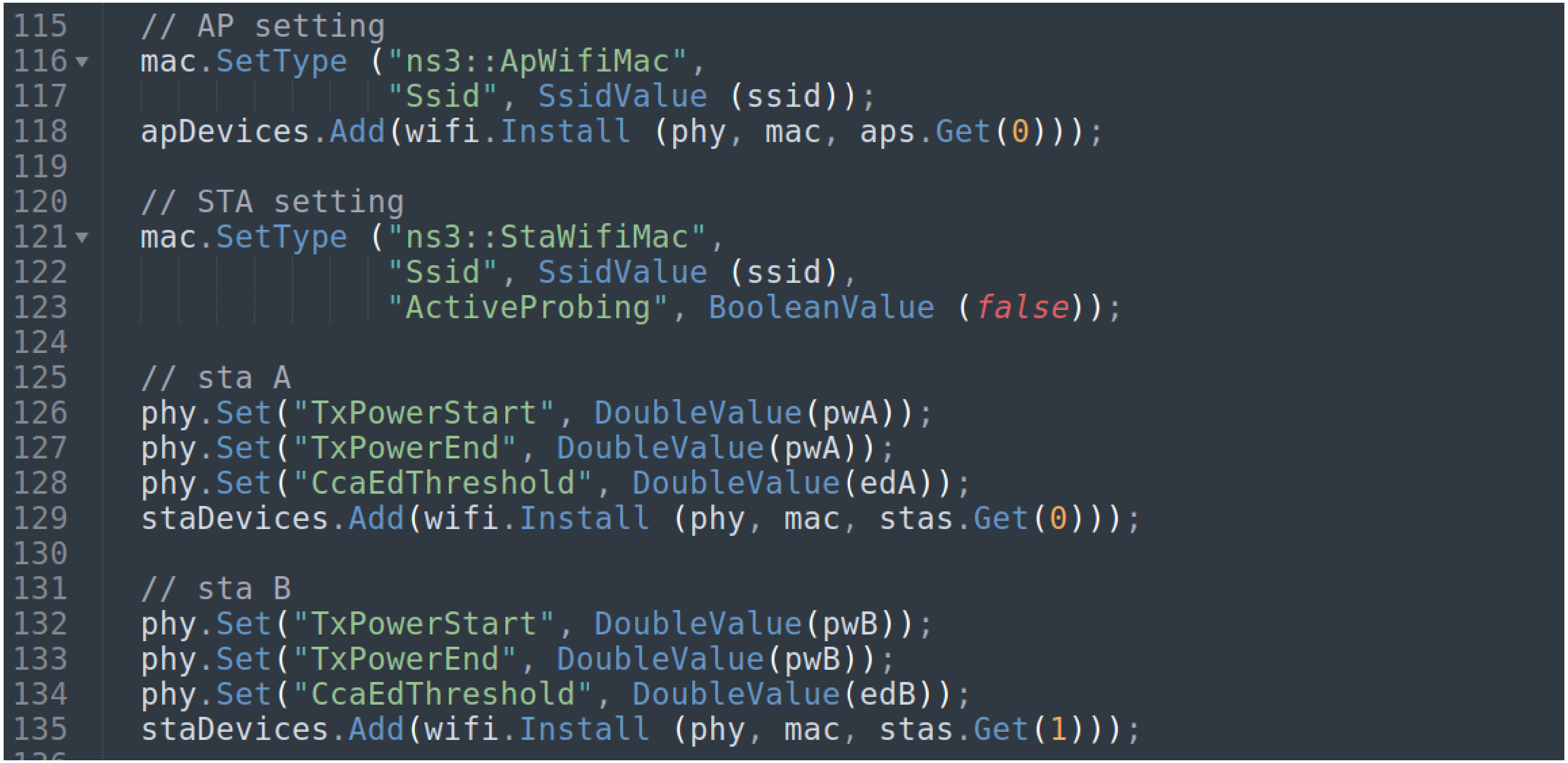
ht.cc



```
  InternetStackHelper internet;
  internet.Install (stas);
  internet.Install (aps);

  NetDeviceContainer apDevices;
  NetDeviceContainer staDevices;


  WifiHelper wifi;
  wifi.SetRemoteStationManager ("ns3::IdealWifiManager");
  wifi.SetStandard (WIFI_STANDARD_80211n);

  std::string ssidString ("snu");
  Ssid ssid = Ssid (ssidString);
```

- Install InternetProtocolStack on the nodes
- You can regard the NetDevice as a network interface card
- We use IdealWifiManager which adapt data rate ideally and follow 802.11n standard

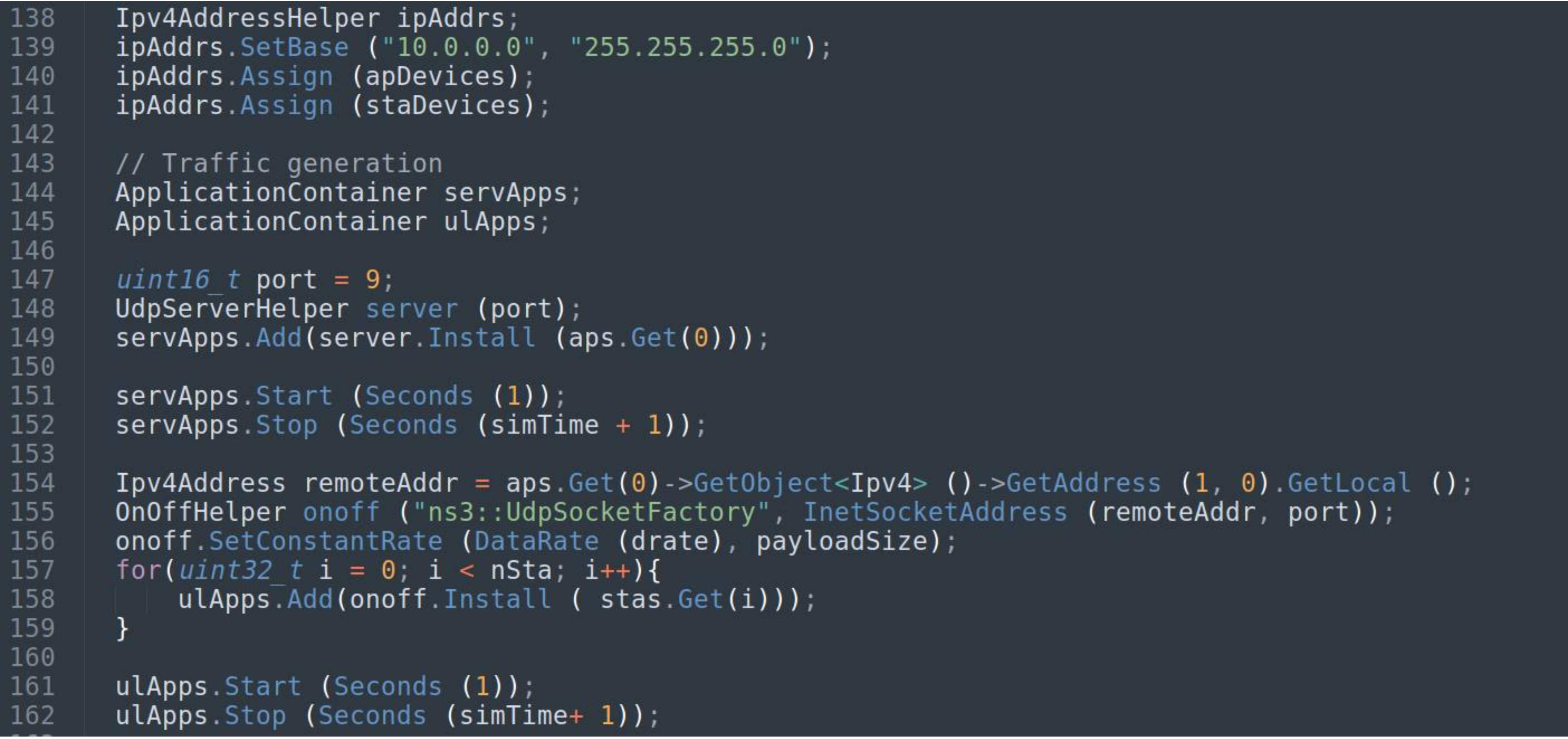
ht.cc



```
// AP setting
mac.SetType ("ns3::ApWifiMac",
             "Ssid", SsidValue (ssid));
apDevices.Add(wifi.Install (phy, mac, aps.Get(0)));

// STA setting
mac.SetType ("ns3::StaWifiMac",
             "Ssid", SsidValue (ssid),
             "ActiveProbing", BooleanValue (false));

// sta A
phy.Set("TxPowerStart", DoubleValue(pwA));
phy.Set("TxPowerEnd", DoubleValue(pwA));
phy.Set("CcaEdThreshold", DoubleValue(edA));
staDevices.Add(wifi.Install (phy, mac, stas.Get(0)));

// sta B
phy.Set("TxPowerStart", DoubleValue(pwB));
phy.Set("TxPowerEnd", DoubleValue(pwB));
phy.Set("CcaEdThreshold", DoubleValue(edB));
staDevices.Add(wifi.Install (phy, mac, stas.Get(1)));
```

- AP and station setting
- In here, we set the tx power and CCA-ED threshold of the stations

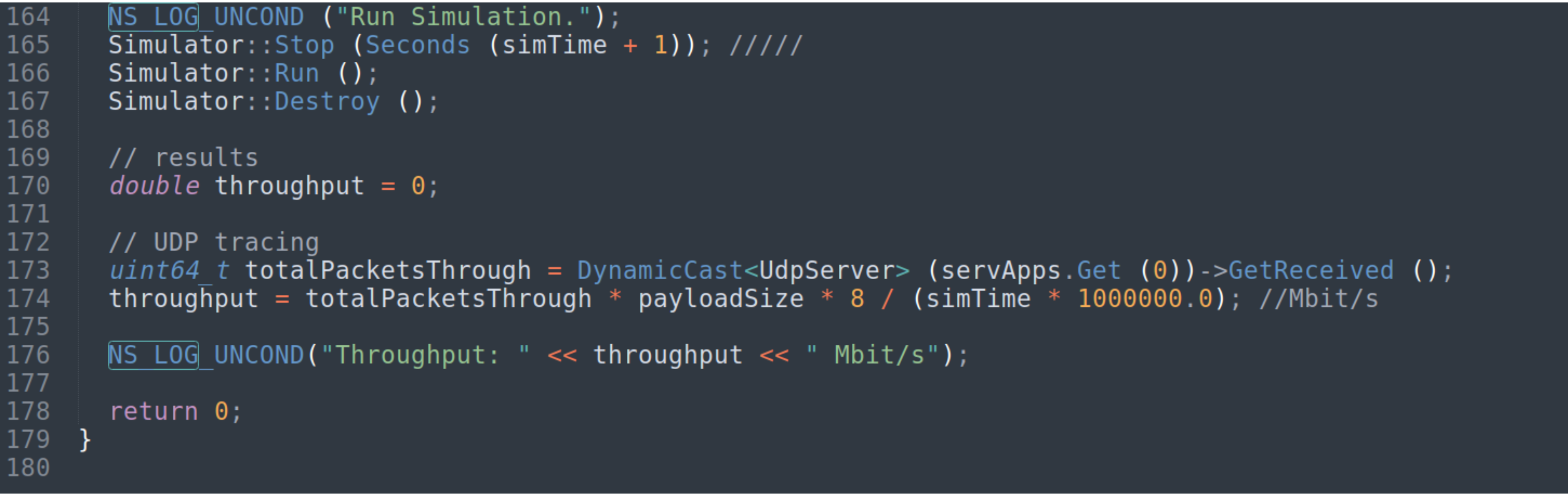
ht.cc



```
Ipv4AddressHelper ipAddrs;
ipAddrs.SetBase ("10.0.0.0", "255.255.255.0");
ipAddrs.Assign (apDevices);
ipAddrs.Assign (staDevices);

// Traffic generation
ApplicationContainer servApps;
ApplicationContainer ulApps;

uint16_t port = 9;
UdpServerHelper server (port);
servApps.Add(server.Install (aps.Get(0)));

servApps.Start (Seconds (1));
servApps.Stop (Seconds (simTime + 1));

Ipv4Address remoteAddr = aps.Get(0)->GetObject<Ipv4> ()->GetAddress (1, 0).GetLocal ();
OnOffHelper onoff ("ns3::UdpSocketFactory", InetSocketAddress (remoteAddr, port));
onoff.SetConstantRate (DataRate (drate), payloadSize);
for(uint32_t i = 0; i < nSta; i++){
    ulApps.Add(onoff.Install ( stas.Get(i)));
}

ulApps.Start (Seconds (1));
ulApps.Stop (Seconds (simTime+ 1));
```

- Set base of IP address and subnet mask
- In here, we install applications which generate data traffics

ht.cc



```
  NS_LOG_UNCOND ("Run Simulation.");
  Simulator::Stop (Seconds (simTime + 1)); /////
  Simulator::Run ();
  Simulator::Destroy ();

  // results
  double throughput = 0;

  // UDP tracing
  uint64_t totalPacketsThrough = DynamicCast<UdpServer> (servApps.Get (0))->GetReceived ();
  throughput = totalPacketsThrough * payloadSize * 8 / (simTime * 1000000.0); //Mbit/s

  NS_LOG_UNCOND("Throughput: " << throughput << " Mbit/s");

  return 0;
}
```

- Finally, we run the simulator and get the simulation result