APPENDIX A

- - -

Simulation Code Listing

```
/* -*-  Mode: C++; c-file-style: "gnu"; indent-tabs-mode:nil; -*- */
/*
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation;
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 *
 * N. Wooster
 * Simon Fraser University
 * 2017
 */

#include <cmath>
#include <ctime>
#include <iomanip>
#include <ios>
#include <ns3/core-module.h>
#include <ns3/network-module.h>
#include <ns3/mobility-module.h>
#include <ns3/internet-module.h>
#include <ns3/lte-module.h>
#include <ns3/config-store-module.h>
#include <ns3/buildings-module.h>
#include <ns3/point-to-point-helper.h>
#include <ns3/applications-module.h>
#include <ns3/log.h>
#include <sstream>
#include <string>
#include <vector>

#define pi 3.14159265359

using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("HetNetHandover");

//////////////////////////

// allocate city blocks of user defined size.
class BlockAllocator
{
public:
  BlockAllocator (Box sim_area, double x_dim, double y_dim, double zMin, double zMax, double st_dim, uint32_t blks_X,
uint32_t blks_Y);
  void Create (uint32_t n);
  void Create ();

private:
  bool LocationOccupied (Box);
  bool OverlapDetector (Box, Box);
  Box topoBounds;
  std::list<Box> prevBlks;
  double blkX_dim;
  double blkY_dim;
  double blkZmin;
  double blkZmax;
  double stWidth;
  uint32_t blkX_cnt;
  uint32_t blkY_cnt;
  Ptr<UniformRandomVariable> blkX;
  Ptr<UniformRandomVariable> blkY;
  Ptr<UniformRandomVariable> blkZ_elev;
};

BlockAllocator::BlockAllocator (Box sim_area, double x_dim, double y_dim, double zMin, double zMax, double st_dim,
uint32_t blks_X, uint32_t blks_Y)
  : topoBounds (sim_area),
    blkX_dim (x_dim),
    blkY_dim (y_dim),
    blkZmin (zMin),
    blkZmax (zMax),
    stWidth (st_dim),
    blkX_cnt (blks_X),
```

```
    blkY_cnt (blks_Y)
{
  blkX = CreateObject<UniformRandomVariable> ();
  blkX->SetAttribute ("Min", DoubleValue (0));
  blkX->SetAttribute ("Max", DoubleValue (blkX_cnt));
  blkY = CreateObject<UniformRandomVariable> ();
  blkY->SetAttribute ("Min", DoubleValue (0));
  blkY->SetAttribute ("Max", DoubleValue (blkY_cnt));
  blkZ_elev = CreateObject<UniformRandomVariable> ();
  blkZ_elev->SetAttribute ("Min", DoubleValue (blkZmin));
  blkZ_elev->SetAttribute ("Max", DoubleValue (blkZmax));
}

void
BlockAllocator::Create (uint32_t n)
{
  for (uint32_t i = 0; i < n; ++i)
    {
      Create ();
    }
}

void
BlockAllocator::Create ()
{
  Box newBlk;
  double xMinGrid;
  double yMinGrid;
  double blkHeight = blkZ_elev->GetValue ();
  do
    {
      xMinGrid = ((topoBounds.xMax - topoBounds.xMin - (blkX_dim * blkX_cnt)
        - (stWidth * (blkX_cnt - 1))) / 2) + ((blkX_dim + stWidth) * floor (blkX->GetValue ()));
      yMinGrid = ((topoBounds.yMax - topoBounds.yMin - (blkY_dim * blkY_cnt)
        - (stWidth * (blkY_cnt - 1))) / 2) + ((blkY_dim + stWidth) * floor (blkY->GetValue ()));
      newBlk.xMin = xMinGrid;
      newBlk.xMax = newBlk.xMin + blkX_dim;
      newBlk.yMin = yMinGrid;
      newBlk.yMax = newBlk.yMin + blkY_dim;
    }
  while (LocationOccupied (newBlk));
  prevBlks.push_back (newBlk);
  Ptr<GridBuildingAllocator>  gridBuildingAllocator;
  gridBuildingAllocator = CreateObject<GridBuildingAllocator> ();
  gridBuildingAllocator->SetAttribute ("GridWidth", UintegerValue (1));
  gridBuildingAllocator->SetAttribute ("LengthX", DoubleValue (blkX_dim));
  gridBuildingAllocator->SetAttribute ("LengthY", DoubleValue (blkY_dim));
  gridBuildingAllocator->SetAttribute ("Height", DoubleValue (blkHeight));
  gridBuildingAllocator->SetAttribute ("MinX", DoubleValue (newBlk.xMin));
  gridBuildingAllocator->SetAttribute ("MinY", DoubleValue (newBlk.yMin));
  gridBuildingAllocator->Create (1);
}

bool
BlockAllocator::OverlapDetector (Box a, Box b)
{
  return !((a.xMin >= b.xMax) || (b.xMin >= a.xMax) || (a.yMin >= b.yMax) || (b.yMin >= a.yMax));
}

bool
BlockAllocator::LocationOccupied (Box newBlk)
{
  for (std::list<Box>::iterator idx = prevBlks.begin (); idx != prevBlks.end (); ++idx)
    {
      if (OverlapDetector (*idx, newBlk))
        {
          return true;
        }
    }
  return false;
}

//////////////////////////

// capture handover control messages between UEs and eNBs.
class HandoverCapture : public SimpleRefCount <HandoverCapture>
{
public:
  HandoverCapture (std::string);
  void PrintToFile (std::string, uint64_t);
  static void ueConnectionEstablishedCallback (Ptr <HandoverCapture>, uint64_t, uint16_t, uint16_t);
  void ueConnectionEstablished (uint64_t, uint16_t, uint16_t);
  static void enbConnectionEstablishedCallback (Ptr <HandoverCapture>, uint64_t, uint16_t, uint16_t);
  void enbConnectionEstablished (uint64_t, uint16_t, uint16_t);
  static void ueHandoverInitiatedCallback (Ptr <HandoverCapture>, uint64_t, uint16_t, uint16_t, uint16_t);
  void ueHandoverInitiated (uint64_t, uint16_t, uint16_t, uint16_t);
  static void enbHandoverInitiatedCallback (Ptr <HandoverCapture>, uint64_t, uint16_t, uint16_t, uint16_t);
  void enbHandoverInitiated (uint64_t, uint16_t, uint16_t, uint16_t);
  static void ueHandoverCompleteCallback (Ptr <HandoverCapture>, uint64_t, uint16_t, uint16_t);
  void ueHandoverComplete (uint64_t, uint16_t, uint16_t);
  static void enbHandoverCompleteCallback (Ptr <HandoverCapture>, uint64_t, uint16_t, uint16_t);
  void enbHandoverComplete (uint64_t, uint16_t, uint16_t);
```

```
private:
  bool firstWrite;
  std::string filename;
};

HandoverCapture::HandoverCapture (std::string filename_in)
  : filename (filename_in)
{
  firstWrite = true;
}

void
HandoverCapture::PrintToFile (std::string newLine, uint64_t imsi)
{
  Vector pos;
  std::ofstream outFile;
  for (NodeList::Iterator it = NodeList::Begin (); it != NodeList::End (); ++it)
    {
      Ptr<Node> node = *it;
      int nDevs = node->GetNDevices ();
      for (int j = 0; j < nDevs; j++)
        {
          Ptr<LteUeNetDevice> uedev = node->GetDevice (j)->GetObject <LteUeNetDevice> ();
          if (uedev)
            {
              if (uedev->GetImsi () == imsi)
                {
                  pos = node->GetObject<MobilityModel> ()->GetPosition ();
                }
            }
        }
    }
  if (firstWrite == true)
    {
      outFile.open (filename.c_str (), std::ios_base::out | std::ios_base::trunc);
      if (!outFile.is_open ())
        {
          NS_LOG_ERROR ("Can't open file " << filename);
          return;
        }
      firstWrite = false;
      outFile << "time\tdeviceType\tstatus\tIMSI\tsourceCellID\ttargetCellID\tRNTI\tueCoordinates";
      outFile << std::endl;
    }
  else
    {
      outFile.open (filename.c_str (), std::ios_base::out | std::ios_base::app);
      if (!outFile.is_open ())
        {
          NS_LOG_ERROR ("Can't open file " << filename);
          return;
        }
    }
  outFile << newLine  << "\t" << "(" << pos.x << ", " << pos.y << ")" << std::endl;
  outFile.close ();
}

// UE connection established
void
HandoverCapture::ueConnectionEstablishedCallback (Ptr <HandoverCapture> instance, uint64_t imsi, uint16_t cellId,
uint16_t rnti)
{
  instance->ueConnectionEstablished (imsi, cellId, rnti);
}

void
HandoverCapture::ueConnectionEstablished (uint64_t imsi, uint16_t cellId, uint16_t rnti)
{
  std::ostringstream newLine;
  newLine << Simulator::Now ().GetSeconds () << "s\t" << "UE" << "\t" << "connection established"
          << "\t" << imsi << "\t" << cellId << "\t" << "" << "\t" << rnti;
  PrintToFile (newLine.str (), imsi);
}

// eNB connection established
void
HandoverCapture::enbConnectionEstablishedCallback (Ptr <HandoverCapture> instance, uint64_t imsi, uint16_t cellId,
uint16_t rnti)
{
  instance->enbConnectionEstablished (imsi, cellId, rnti);
}

void
HandoverCapture::enbConnectionEstablished (uint64_t imsi, uint16_t cellId, uint16_t rnti)
{
  std::ostringstream newLine;
  newLine << Simulator::Now ().GetSeconds () << "s\t" << "eNB" << "\t" << "connection established"
          << "\t" << imsi << "\t" << cellId << "\t" << "" << "\t" << rnti;
  PrintToFile (newLine.str (), imsi);
}
```

```
// UE handover initiated
void
HandoverCapture::ueHandoverInitiatedCallback (Ptr <HandoverCapture> instance, uint64_t imsi, uint16_t cellId, uint16_t
rnti, uint16_t targetCellId)
{
  instance->ueHandoverInitiated (imsi, cellId, rnti, targetCellId);
}

void
HandoverCapture::ueHandoverInitiated (uint64_t imsi, uint16_t cellId, uint16_t rnti, uint16_t targetCellId)
{
  std::ostringstream newLine;
  newLine << Simulator::Now ().GetSeconds () << "s\t" << "UE" << "\t" << "handover initiated"
          << "\t" << imsi << "\t" << cellId << "\t" << targetCellId << "\t" << rnti;
  PrintToFile (newLine.str (), imsi);
}

// eNB handover intiated
void
HandoverCapture::enbHandoverInitiatedCallback (Ptr <HandoverCapture> instance, uint64_t imsi, uint16_t cellId, uint16_t
rnti, uint16_t targetCellId)
{
  instance->enbHandoverInitiated (imsi, cellId, rnti, targetCellId);
}

void
HandoverCapture::enbHandoverInitiated (uint64_t imsi, uint16_t cellId, uint16_t rnti, uint16_t targetCellId)
{
  std::ostringstream newLine;
  newLine << Simulator::Now ().GetSeconds () << "s\t" << "eNB" << "\t" << "handover initiated"
          << "\t" << imsi << "\t" << cellId << "\t" << targetCellId << "\t" << rnti;
  PrintToFile (newLine.str (), imsi);
}

// UE handover complete
void
HandoverCapture::ueHandoverCompleteCallback (Ptr <HandoverCapture> instance, uint64_t imsi, uint16_t cellId, uint16_t
rnti)
{
  instance->ueHandoverComplete (imsi, cellId, rnti);
}

void
HandoverCapture::ueHandoverComplete (uint64_t imsi, uint16_t cellId, uint16_t rnti)
{
  std::ostringstream newLine;
  newLine << Simulator::Now ().GetSeconds () << "s\t" << "UE" << "\t" << "handover complete"
          << "\t" << imsi << "\t" << cellId << "\t" << "" << "\t" << rnti;
  PrintToFile (newLine.str (), imsi);
}

// eNB handover complete
void
HandoverCapture::enbHandoverCompleteCallback (Ptr <HandoverCapture> instance, uint64_t imsi, uint16_t cellId, uint16_t
rnti)
{
  instance->enbHandoverComplete (imsi, cellId, rnti);
}

void
HandoverCapture::enbHandoverComplete (uint64_t imsi, uint16_t cellId, uint16_t rnti)
{
  std::ostringstream newLine;
  newLine << Simulator::Now ().GetSeconds () << "s\t" << "eNB" << "\t" << "handover complete"
          << "\t" << imsi << "\t" << cellId << "\t" << "" << "\t" << rnti;
  PrintToFile (newLine.str (), imsi);
}

//////////////////////////

// capture course changes for mobile network nodes.
class MobilityCapture : public SimpleRefCount <MobilityCapture>
{
public:
  MobilityCapture (std::string);
  void PrintToFile (std::string, std::string, uint64_t);
  static void courseChangeCallback (Ptr <MobilityCapture>, Ptr <const MobilityModel>);
  void courseChange (Ptr <const MobilityModel>);

private:
  bool firstWrite;
  std::string filename;
};

MobilityCapture::MobilityCapture (std::string filename_in)
  : filename (filename_in)
{
  firstWrite = true;
}

void
MobilityCapture::PrintToFile (std::string time, std::string newLine, uint64_t nodeId)
```

```
{
  uint64_t imsi;
  std::ofstream outFile;
  for (NodeList::Iterator it = NodeList::Begin (); it != NodeList::End (); ++it)
    {
      Ptr<Node> node = *it;
      int nDevs = node->GetNDevices ();
      for (int j = 0; j < nDevs; j++)
        {
          Ptr<LteUeNetDevice> uedev = node->GetDevice (j)->GetObject <LteUeNetDevice> ();
          if (uedev)
            {
              if (node->GetId () == nodeId)
                {
                  imsi = uedev->GetImsi ();
                }
            }
        }
    }
  if (firstWrite == true)
    {
      outFile.open (filename.c_str (), std::ios_base::out | std::ios_base::trunc);
      if (!outFile.is_open ())
        {
          NS_LOG_ERROR ("Can't open file " << filename);
          return;
        }
      firstWrite = false;
      outFile << "time\tIMSI\tueCoordinates\tueVelocity";
      outFile << std::endl;
    }
  else
    {
      outFile.open (filename.c_str (), std::ios_base::out | std::ios_base::app);
      if (!outFile.is_open ())
        {
          NS_LOG_ERROR ("Can't open file " << filename);
          return;
        }
    }
  outFile << time << "s\t" << imsi << "\t" << newLine <<std::endl;
  outFile.close ();
}

void
MobilityCapture::courseChangeCallback (Ptr <MobilityCapture> instance, Ptr <const MobilityModel> mobility)
{
  instance->courseChange (mobility);
}

void
MobilityCapture::courseChange (Ptr <const MobilityModel> mobility)
{
  std::ostringstream time;
  std::ostringstream newLine;
  time << Simulator::Now ().GetSeconds ();
  Ptr<Node> node = mobility->GetObject<Node> ();
  Vector pos = mobility->GetPosition ();
  Vector vel = mobility->GetVelocity ();
  newLine << "(" << pos.x << ", " << pos.y << ")" << "\t"
          << "(" << vel.x << ", " << vel.y << ")";
  PrintToFile (time.str (), newLine.str (), node->GetId ());
}

//////////////////////////

// trigger custom traces with callbacks to handler classes.
class CustomTracesHelper
{
public:
  CustomTracesHelper ();
  void EnableHandoverTraces ();
  void EnableMobilityTraces ();

private:
  std::string handoverFilename;
  std::string mobilityFilename;
};

CustomTracesHelper::CustomTracesHelper ()
{
  handoverFilename = "HandoverTrace.txt";
  mobilityFilename = "MobilityTrace.txt";
}

void
CustomTracesHelper::EnableHandoverTraces ()
{
  Ptr <HandoverCapture> handoverTrace = Create <HandoverCapture> (handoverFilename);
  Config::ConnectWithoutContext ("/NodeList/*/DeviceList/*/LteUeRrc/ConnectionEstablished",
    MakeBoundCallback (&HandoverCapture::ueConnectionEstablishedCallback, handoverTrace));
  Config::ConnectWithoutContext ("/NodeList/*/DeviceList/*/LteEnbRrc/ConnectionEstablished",
```

```
    MakeBoundCallback (&HandoverCapture::enbConnectionEstablishedCallback, handoverTrace));
  Config::ConnectWithoutContext ("/NodeList/*/DeviceList/*/LteUeRrc/HandoverStart",
    MakeBoundCallback (&HandoverCapture::ueHandoverInitiatedCallback, handoverTrace));
  Config::ConnectWithoutContext ("/NodeList/*/DeviceList/*/LteEnbRrc/HandoverStart",
    MakeBoundCallback (&HandoverCapture::enbHandoverInitiatedCallback, handoverTrace));
  Config::ConnectWithoutContext ("/NodeList/*/DeviceList/*/LteUeRrc/HandoverEndOk",
    MakeBoundCallback (&HandoverCapture::ueHandoverCompleteCallback, handoverTrace));
  Config::ConnectWithoutContext ("/NodeList/*/DeviceList/*/LteEnbRrc/HandoverEndOk",
    MakeBoundCallback (&HandoverCapture::enbHandoverCompleteCallback, handoverTrace));
}

void
CustomTracesHelper::EnableMobilityTraces ()
{

  Ptr <MobilityCapture> mobilityTrace = Create <MobilityCapture> (mobilityFilename);
  Config::ConnectWithoutContext ("/NodeList/*/$ns3::MobilityModel/CourseChange",
    MakeBoundCallback (&MobilityCapture::courseChangeCallback, mobilityTrace));
}

////////////////////////

void
buildingsPrintListToFile (std::string filename)
{
  std::ofstream outFile;
  outFile.open (filename.c_str (), std::ios_base::out | std::ios_base::trunc);
  if (!outFile.is_open ())
    {
      NS_LOG_ERROR ("Can't open file " << filename);
      return;
    }
  uint32_t index = 0;
  for (BuildingList::Iterator it = BuildingList::Begin (); it != BuildingList::End (); ++it)
    {
      ++index;
      Box box = (*it)->GetBoundaries ();
      outFile << "set object " << index
              << " rect from " << box.xMin  << "," << box.yMin
              << " to "   << box.xMax  << "," << box.yMax
              << " front fs empty "
              << std::endl;
    }
}

void
eNBsPrintListToFile (std::string filename)
{
  std::ofstream outFile;
  outFile.open (filename.c_str (), std::ios_base::out | std::ios_base::trunc);
  if (!outFile.is_open ())
    {
      NS_LOG_ERROR ("Can't open file " << filename);
      return;
    }
  for (NodeList::Iterator it = NodeList::Begin (); it != NodeList::End (); ++it)
    {
      Ptr<Node> node = *it;
      int nDevs = node->GetNDevices ();
      for (int j = 0; j < nDevs; j++)
        {
          Ptr<LteEnbNetDevice> enbdev = node->GetDevice (j)->GetObject <LteEnbNetDevice> ();
          if (enbdev)
            {
              Vector pos = node->GetObject<MobilityModel> ()->GetPosition ();
              outFile << "set label \"" << enbdev->GetCellId ()
                      << "\" at "<< pos.x << "," << pos.y
                      << " left font \"Helvetica,4\" textcolor rgb \"white\""
                      << "front  point pt 2 ps 0.3 lc rgb \"white\" offset 0,0"
                      << std::endl;
            }
        }
    }
}

void
uesPrintListToFile (std::string filename)
{
  std::ofstream outFile;
  outFile.open (filename.c_str (), std::ios_base::out | std::ios_base::trunc);
  if (!outFile.is_open ())
    {
      NS_LOG_ERROR ("Can't open file " << filename);
      return;
    }
  for (NodeList::Iterator it = NodeList::Begin (); it != NodeList::End (); ++it)
    {
      Ptr<Node> node = *it;
      int nDevs = node->GetNDevices ();
      for (int j = 0; j < nDevs; j++)
        {
          Ptr<LteUeNetDevice> uedev = node->GetDevice (j)->GetObject <LteUeNetDevice> ();
```

```
          if (uedev)
            {
              Vector pos = node->GetObject<MobilityModel> ()->GetPosition ();
              outFile << "set label \"" << uedev->GetImsi ()
                      << "\" at "<< pos.x << "," << pos.y
                      << " left font \"Helvetica,4\" textcolor rgb \"grey\""
                      << " front point pt 1 ps 0.3 lc rgb \"grey\" offset 0,0"
                      << std::endl;
            }
        }
    }
}

//////////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////////////////////////////

int
main (int argc, char *argv[])
{
  time_t tic, toc;
  time(&tic);

  // configure default simulation parameters.
  Config::SetDefault ("ns3::UdpClient::Interval", TimeValue (MilliSeconds (1)));
  Config::SetDefault ("ns3::UdpClient::MaxPackets", UintegerValue (1000000));
  Config::SetDefault ("ns3::LteRlcUm::MaxTxBufferSize", UintegerValue (10 * 1024));
  Config::SetDefault ("ns3::LteSpectrumPhy::CtrlErrorModelEnabled", BooleanValue (true));
  Config::SetDefault ("ns3::LteSpectrumPhy::DataErrorModelEnabled", BooleanValue (true));

  // enable command line input to overwrite defaults.
  CommandLine cmd;
  cmd.Parse (argc, argv);
  ConfigStore inputConfig;
  inputConfig.ConfigureDefaults ();
  cmd.Parse (argc, argv);

//////////////////////////////////////////////////////////////////////////////////////////////////

           uint32_t macroCellsX_cnt = 1; // number of sites along the x-axis of the hex grid.
           uint32_t macroCellsY_cnt = 2; // number of sites along the y-axis of the hex grid.
                double macroCellsZ = 30; // elevation of macrocell air interface.
             double macroCells_spc = 500; // distance between macrocell sites.
           double gridMargin_fact = 0.25; // how much of the topology extends beyond the hex grid.
         double macroCellTxPowerDbm = 46.0; // TX power [dBm] used by macro eNBs.
          uint16_t macroCellDlEarfcn = 100; // DL EARFCN used by macro eNBs.
          uint16_t macroCellBandwidth = 25; // bandwidth [num RBs] used by macro eNBs.

           uint32_t smallSites_cnt = 1; // number of smallcell sites. 1 site consists of 3 nodes.
                double smallCellsZ = 6; // elevation of smallcell site air interface.
             double smallCells_spc = 20; // distance between any two smallcell sites.
         double smallCellTxPowerDbm = 20.0; // TX power [dBm] used by small eNBs.
          uint16_t smallCellDlEarfcn = 100; // DL EARFCN used by small eNBs.
          uint16_t smallCellBandwidth = 25; // bandwidth [num RBs] used by small eNBs.

              uint32_t ueNodes_cnt = 1; // number of deployed UEs.
                     double ueZ = 1.75; // UE height.
           uint16_t numBearersPerUe = 1; // number of bearers per UE.
                double ueMinSpeed = 0.5; // minimum speed value of UEs [m/s].
                double ueMaxSpeed = 3.5; // maximum speed value of UEs [m/s].
            double ueRangeMargin = 30.0; // how much the UE mobility range extends beyond the
                                         // centre of a smallcell site. requires UE range to be
                                         // constrained.
           bool ueRangeConstrain = true; // constrain range of UE mobility.

                double blkDensity = 0.2; // city block density with respect to hex grid.
                   double blkX_dim = 80; // city block x dimension.
                   double blkY_dim = 80; // city block y dimension.
                    double stWidth = 20; // city street width.
               double blkZmin_dim = 6.0; // minimum z dimension of blocks.
              double blkZmax_dim = 15.0; // maximum z dimension of blocks.

                 double simTime = 900.0; // total duration of the simulation [s].

                       bool epc = true; // setup the EPC. otherwise, only the LTE radio access
                                        // will be simulated with RLC SM.
                    bool epcDl = true; // activate data flows in the downlink. requires epc.
                    bool epcUl = true; // activate data flows in the uplink. requires epc.
                   bool useUdp = true; // use UdpClient application. otherwise, use BulkSend
                                       // application over a TCP connection. requires EPC.
           uint16_t srsPeriodicity = 80; // SRS periodicity. must be at least greater than the
                                         // number of UEs per eNB.

              double A3Hysteresis = 3.0; //
           double A3TimeToTrigger = 250; // time delay upon assertion of A3 condition.
         uint16_t A2A4SrvCellThresh = 30; // threshold rsrq table row.
          uint16_t A2A4NbrCellOffset = 1; // minimum offset between rsrq table rows.
  std::string handoverAlgorithm = "A2A4"; // A2A4 or A3.

               bool generateRem = false; // generate a REM and then abort the simulation.
                   int32_t remRbId = -1; // resource block id for which REM will be generated. if
                                         // -1, REM will be average of all RBs of control channel
```

```
////////////////////////////////////////////////////////////////////////////////////////////////

  Config::SetDefault ("ns3::LteEnbRrc::SrsPeriodicity", UintegerValue (srsPeriodicity));

  Ptr <LteHelper> lteHelper = CreateObject<LteHelper> ();
  Ptr<PointToPointEpcHelper> epcHelper;

  // configure MAC scheduler and handover algorithm.
  if (epc)
    {
      epcHelper = CreateObject<PointToPointEpcHelper> ();
      lteHelper->SetEpcHelper (epcHelper);
      lteHelper->SetSchedulerType ("ns3::PfFfMacScheduler");

      if (handoverAlgorithm == "A2A4")
        {
          lteHelper->SetHandoverAlgorithmType ("ns3::A2A4RsrqHandoverAlgorithm");
          lteHelper->SetHandoverAlgorithmAttribute ("ServingCellThreshold", UintegerValue (A2A4SrvCellThresh));
          lteHelper->SetHandoverAlgorithmAttribute ("NeighbourCellOffset", UintegerValue (A2A4NbrCellOffset));
        }
      else if (handoverAlgorithm == "A3")
        {
          lteHelper->SetHandoverAlgorithmType ("ns3::A3RsrpHandoverAlgorithm");
          lteHelper->SetHandoverAlgorithmAttribute ("Hysteresis", DoubleValue (A3Hysteresis));
          lteHelper->SetHandoverAlgorithmAttribute ("TimeToTrigger", TimeValue (MilliSeconds (A3TimeToTrigger)));
        }
    }
  NS_LOG_LOGIC ("using " << lteHelper->GetHandoverAlgorithmType () << " handover algorithm");

  // determine counts of physical network components to be created.
  uint32_t macroSites_cnt = macroCellsY_cnt * macroCellsX_cnt + (macroCellsY_cnt - (macroCellsY_cnt % 2)) / 2;
  uint32_t macroNodes_cnt = 3 * macroSites_cnt;
  uint32_t smallNodes_cnt = 3 * smallSites_cnt;

  // generate network topology boundaries.
  Box topoBounds;
  topoBounds = Box (0, (macroCellsX_cnt + (2 * gridMargin_fact)) * macroCells_spc,
                    0, (((macroCellsY_cnt - 1) * sqrt(3) / 2) + (2 * gridMargin_fact)) * macroCells_spc,
                    0, macroCellsZ);

  // create macro eNB, small eNB, and UE nodes.
  NodeContainer macroNodes;
  macroNodes.Create (macroNodes_cnt);
  NS_LOG_LOGIC (macroNodes_cnt << " macro eNB nodes created across " << macroSites_cnt << " sites");
  NodeContainer smallNodes;
  smallNodes.Create (smallNodes_cnt);
  NS_LOG_LOGIC (smallNodes_cnt << " small eNB nodes created");
  NodeContainer ueNodes;
  ueNodes.Create (ueNodes_cnt);
  NS_LOG_LOGIC (ueNodes_cnt << " UE nodes created");

  // place city blocks.
  if (blkDensity > 0)
    {
      uint32_t blkX_cnt = floor ((topoBounds.xMax - topoBounds.xMin) / (blkX_dim + stWidth));
      uint32_t blkY_cnt = floor ((topoBounds.yMax - topoBounds.yMin) / (blkY_dim + stWidth));
      uint32_t blksTot_cnt = blkDensity * blkX_cnt * blkY_cnt;
      BlockAllocator cityBlockAllocator (topoBounds, blkX_dim, blkY_dim, blkZmin_dim, blkZmax_dim, stWidth, blkX_cnt,
blkY_cnt);
      cityBlockAllocator.Create (blksTot_cnt);
      NS_LOG_LOGIC (blksTot_cnt << " buildings created");

      lteHelper->SetAttribute ("PathlossModel", StringValue ("ns3::HybridBuildingsPropagationLossModel"));

      lteHelper->SetPathlossModelAttribute ("ShadowSigmaExtWalls", DoubleValue (0));
      lteHelper->SetPathlossModelAttribute ("ShadowSigmaOutdoor", DoubleValue (1));
      lteHelper->SetPathlossModelAttribute ("ShadowSigmaIndoor", DoubleValue (1.5));
      lteHelper->SetPathlossModelAttribute ("Los2NlosThr", DoubleValue (1e6));
      lteHelper->SetSpectrumChannelType ("ns3::MultiModelSpectrumChannel");
    }

  // install macro eNB mobility model and configure nodes.
  MobilityHelper macroCellMobility;
  macroCellMobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
  macroCellMobility.Install (macroNodes);
  BuildingsHelper::Install (macroNodes);
  Ptr<LteHexGridEnbTopologyHelper> lteHexGridEnbTopologyHelper = CreateObject <LteHexGridEnbTopologyHelper> ();
  lteHexGridEnbTopologyHelper->SetLteHelper (lteHelper);
  lteHexGridEnbTopologyHelper->SetAttribute ("InterSiteDistance", DoubleValue (macroCells_spc));
  lteHexGridEnbTopologyHelper->SetAttribute ("MinX", DoubleValue ((0.5 + gridMargin_fact) * macroCells_spc));
  lteHexGridEnbTopologyHelper->SetAttribute ("MinY", DoubleValue (gridMargin_fact * macroCells_spc));
  lteHexGridEnbTopologyHelper->SetAttribute ("SiteHeight", DoubleValue (macroCellsZ));
  lteHexGridEnbTopologyHelper->SetAttribute ("GridWidth", UintegerValue (macroCellsX_cnt));
  Config::SetDefault ("ns3::LteEnbPhy::TxPower", DoubleValue (macroCellTxPowerDbm));
  lteHelper->SetEnbAntennaModelType ("ns3::ParabolicAntennaModel");
  lteHelper->SetEnbAntennaModelAttribute ("Beamwidth", DoubleValue (70));
  lteHelper->SetEnbAntennaModelAttribute ("MaxAttenuation", DoubleValue (20.0));
  lteHelper->SetEnbDeviceAttribute ("DlEarfcn", UintegerValue (macroCellDlEarfcn));
  lteHelper->SetEnbDeviceAttribute ("UlEarfcn", UintegerValue (macroCellDlEarfcn + 18000));
  lteHelper->SetEnbDeviceAttribute ("DlBandwidth", UintegerValue (macroCellBandwidth));
  lteHelper->SetEnbDeviceAttribute ("UlBandwidth", UintegerValue (macroCellBandwidth));
  NetDeviceContainer macroCellDevs = lteHexGridEnbTopologyHelper->SetPositionAndInstallEnbDevice (macroNodes);
```

```
  NS_LOG_LOGIC ("macro eNB nodes configured");

  // install small eNB mobility model and configure nodes.
  Ptr<ListPositionAllocator> smallCellPositionAlloc = CreateObject<ListPositionAllocator> ();
  Ptr<ListPositionAllocator> smallCellSitePosition = CreateObject<ListPositionAllocator> ();
  Ptr<UniformRandomVariable> cellX_coord;
  Ptr<UniformRandomVariable> cellY_coord;
  Ptr<UniformRandomVariable> cellOffset_ang;
  cellX_coord = CreateObject<UniformRandomVariable> ();
  cellX_coord->SetAttribute ("Min", DoubleValue (topoBounds.xMin + (gridMargin_fact * macroCells_spc) +
smallCells_spc));
  cellX_coord->SetAttribute ("Max", DoubleValue (topoBounds.xMax - (gridMargin_fact * macroCells_spc) -
smallCells_spc));
  cellY_coord = CreateObject<UniformRandomVariable> ();
  cellY_coord->SetAttribute ("Min", DoubleValue (topoBounds.yMin + (gridMargin_fact * macroCells_spc) +
smallCells_spc));
  cellY_coord->SetAttribute ("Max", DoubleValue (topoBounds.yMax - (gridMargin_fact * macroCells_spc) -
smallCells_spc));
  cellOffset_ang = CreateObject<UniformRandomVariable> ();
  cellOffset_ang->SetAttribute ("Min", DoubleValue (0));
  cellOffset_ang->SetAttribute ("Max", DoubleValue (2 * pi));
  for (uint32_t i = 0; i < smallSites_cnt; ++i)
    {
      double xVal = cellX_coord->GetValue ();
      double yVal = cellY_coord->GetValue ();
      double oVal = cellOffset_ang->GetValue ();
      double rVal = smallCells_spc / sqrt (3);
      Vector sitePosition (xVal, yVal, smallCellsZ);
      smallCellSitePosition->Add (sitePosition);
      for (uint16_t j = 0; j < 3; j++)
      {
        Vector nodePosition (xVal + (rVal * cos (oVal + (j * 2 * pi / 3))), yVal + (rVal * sin (oVal + (j * 2 * pi /
3))), smallCellsZ);
        smallCellPositionAlloc->Add (nodePosition);
      }
    }
  MobilityHelper smallCellMobility;
  smallCellMobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
  smallCellMobility.SetPositionAllocator (smallCellPositionAlloc);
  smallCellMobility.Install (smallNodes);
  BuildingsHelper::Install (smallNodes);
  Config::SetDefault ("ns3::LteEnbPhy::TxPower", DoubleValue (smallCellTxPowerDbm));
  lteHelper->SetEnbAntennaModelType ("ns3::IsotropicAntennaModel");
  lteHelper->SetEnbDeviceAttribute ("DlEarfcn", UintegerValue (smallCellDlEarfcn));
  lteHelper->SetEnbDeviceAttribute ("UlEarfcn", UintegerValue (smallCellDlEarfcn + 18000));
  lteHelper->SetEnbDeviceAttribute ("DlBandwidth", UintegerValue (smallCellBandwidth));
  lteHelper->SetEnbDeviceAttribute ("UlBandwidth", UintegerValue (smallCellBandwidth));
  NetDeviceContainer smallCellDevs = lteHelper->InstallEnbDevice (smallNodes);
  NS_LOG_LOGIC ("small eNB nodes configured");

  NodeContainer enbNodes;
  enbNodes.Add(macroNodes);
  enbNodes.Add(smallNodes);

  NetDeviceContainer enbDevs;
  enbDevs.Add(macroCellDevs);
  enbDevs.Add(smallCellDevs);

  // install UE mobility model and configure nodes.
  MobilityHelper ueMobility;
  if (ueMaxSpeed > 1e-6 && ueMinSpeed > 1e-6)
    {
      if (ueRangeConstrain)
        {
          for (NodeContainer::Iterator it = ueNodes.Begin (); it != ueNodes.End (); ++it)
            {
              Vector sitePosition = smallCellSitePosition->GetNext ();
              double xValMin = std::max (sitePosition.x - ueRangeMargin, topoBounds.xMin);
              double xValMax = std::min (sitePosition.x + ueRangeMargin, topoBounds.xMax);
              double yValMin = std::max (sitePosition.y - ueRangeMargin, topoBounds.yMin);
              double yValMax = std::min (sitePosition.y + ueRangeMargin, topoBounds.yMax);
              ueMobility.SetMobilityModel ("ns3::SteadyStateRandomWaypointMobilityModel");
              Config::SetDefault ("ns3::SteadyStateRandomWaypointMobilityModel::MinX", DoubleValue (xValMin));
              Config::SetDefault ("ns3::SteadyStateRandomWaypointMobilityModel::MinY", DoubleValue (yValMin));
              Config::SetDefault ("ns3::SteadyStateRandomWaypointMobilityModel::MaxX", DoubleValue (xValMax));
              Config::SetDefault ("ns3::SteadyStateRandomWaypointMobilityModel::MaxY", DoubleValue (yValMax));
              Config::SetDefault ("ns3::SteadyStateRandomWaypointMobilityModel::Z", DoubleValue (ueZ));
              Config::SetDefault ("ns3::SteadyStateRandomWaypointMobilityModel::MaxSpeed", DoubleValue (ueMaxSpeed));
              Config::SetDefault ("ns3::SteadyStateRandomWaypointMobilityModel::MinSpeed", DoubleValue (ueMinSpeed));
              ueMobility.Install (*it);
              (*it)->Initialize ();
              NS_LOG_LOGIC ("UE nodes configured with constrained mobility range");
            }
        }
      else
        {
          ueMobility.SetMobilityModel ("ns3::SteadyStateRandomWaypointMobilityModel");
          Config::SetDefault ("ns3::SteadyStateRandomWaypointMobilityModel::MinX", DoubleValue (topoBounds.xMin));
          Config::SetDefault ("ns3::SteadyStateRandomWaypointMobilityModel::MinY", DoubleValue (topoBounds.yMin));
          Config::SetDefault ("ns3::SteadyStateRandomWaypointMobilityModel::MaxX", DoubleValue (topoBounds.xMax));
          Config::SetDefault ("ns3::SteadyStateRandomWaypointMobilityModel::MaxY", DoubleValue (topoBounds.yMax));
          Config::SetDefault ("ns3::SteadyStateRandomWaypointMobilityModel::Z", DoubleValue (ueZ));
```

```
          Config::SetDefault ("ns3::SteadyStateRandomWaypointMobilityModel::MaxSpeed", DoubleValue (ueMaxSpeed));
          Config::SetDefault ("ns3::SteadyStateRandomWaypointMobilityModel::MinSpeed", DoubleValue (ueMinSpeed));
          ueMobility.Install (ueNodes);
          for (NodeContainer::Iterator it = ueNodes.Begin (); it != ueNodes.End (); ++it)
            {
              (*it)->Initialize ();
            }
          NS_LOG_LOGIC ("UE nodes configured with unconstrained mobility range");
        }
    }
    else
    {
      Ptr<PositionAllocator> uePositionAlloc = CreateObject<RandomBoxPositionAllocator> ();
      Ptr<UniformRandomVariable> xVal = CreateObject<UniformRandomVariable> ();
      xVal->SetAttribute ("Min", DoubleValue (topoBounds.xMin));
      xVal->SetAttribute ("Max", DoubleValue (topoBounds.xMax));
      uePositionAlloc->SetAttribute ("X", PointerValue (xVal));
      Ptr<UniformRandomVariable> yVal = CreateObject<UniformRandomVariable> ();
      yVal->SetAttribute ("Min", DoubleValue (topoBounds.yMin));
      yVal->SetAttribute ("Max", DoubleValue (topoBounds.yMax));
      uePositionAlloc->SetAttribute ("Y", PointerValue (yVal));
      Ptr<ConstantRandomVariable> zVal = CreateObject<ConstantRandomVariable> ();
      zVal->SetAttribute ("Constant", DoubleValue (ueZ));
      uePositionAlloc->SetAttribute ("Z", PointerValue (zVal));
      ueMobility.SetPositionAllocator (uePositionAlloc);
      ueMobility.Install (ueNodes);
      NS_LOG_LOGIC ("UE nodes configured with constant position");
    }
  BuildingsHelper::Install (ueNodes);
  NetDeviceContainer ueDevs = lteHelper->InstallUeDevice (ueNodes);

  Ipv4Address remoteHostAddr;
  Ipv4StaticRoutingHelper ipv4RoutingHelper;
  Ipv4InterfaceContainer ueIPInterfaces;
  Ptr<Node> remoteHost;

  if (epc)
    {
      // create a single RemoteHost.
      NodeContainer remoteHostContainer;
      remoteHostContainer.Create (1);
      remoteHost = remoteHostContainer.Get (0);
      InternetStackHelper internet;
      internet.Install (remoteHostContainer);

      // create the Internet.
      PointToPointHelper p2ph;
      p2ph.SetDeviceAttribute ("DataRate", DataRateValue (DataRate ("100Gb/s")));
      p2ph.SetDeviceAttribute ("Mtu", UintegerValue (1500));
      p2ph.SetChannelAttribute ("Delay", TimeValue (Seconds (0.010)));
      Ptr<Node> pgw = epcHelper->GetPgwNode ();
      NetDeviceContainer internetDevices = p2ph.Install (pgw, remoteHost);
      Ipv4AddressHelper ipv4h;
      ipv4h.SetBase ("1.0.0.0", "255.0.0.0");
      Ipv4InterfaceContainer internetIpIfaces = ipv4h.Assign (internetDevices);
      remoteHostAddr = internetIpIfaces.GetAddress (1);

      // routing of the Internet host towards the LTE network.
      // interface 0 is the pgw, 1 is the remoteHost.
      Ipv4StaticRoutingHelper ipv4RoutingHelper;
      Ptr<Ipv4StaticRouting> remoteHostStaticRouting = ipv4RoutingHelper.GetStaticRouting (remoteHost->GetObject<Ipv4>
());
      remoteHostStaticRouting->AddNetworkRouteTo (Ipv4Address ("7.0.0.0"), Ipv4Mask ("255.0.0.0"), 1);

      // install IP stack on the UEs.
      internet.Install (ueNodes);
      ueIPInterfaces = epcHelper->AssignUeIpv4Address (NetDeviceContainer (ueDevs));

      // attach UEs (needs to be done after IP stack configuration)
      // using initial cell selection
      lteHelper->Attach (ueDevs);
    }
  else
    {
      // UEs are attached to the closest macro eNB.
      lteHelper->AttachToClosestEnb (ueDevs, macroCellDevs);
    }

  if (epc)
    {
      // install and start applications on UEs and remote host.
      uint16_t dlPort = 10000;
      uint16_t ulPort = 20000;

      // randomize bit start times to avoid simulation artifacts
      Ptr<UniformRandomVariable> startTimeSeconds = CreateObject<UniformRandomVariable> ();
      if (useUdp)
        {
          startTimeSeconds->SetAttribute ("Min", DoubleValue (0));
          startTimeSeconds->SetAttribute ("Max", DoubleValue (0.010));
        }
      else
```

```
  {
    // TCP needs to be started late enough so that all UEs are connected
    // otherwise TCP SYN packets will get lost
    startTimeSeconds->SetAttribute ("Min", DoubleValue (0.100));
    startTimeSeconds->SetAttribute ("Max", DoubleValue (0.110));
  }

for (uint32_t u = 0; u < ueNodes.GetN (); ++u)
  {
    Ptr<Node> ue = ueNodes.Get (u);
    // set the default gateway for the UE
    Ptr<Ipv4StaticRouting> ueStaticRouting = ipv4RoutingHelper.GetStaticRouting (ue->GetObject<Ipv4> ());
    ueStaticRouting->SetDefaultRoute (epcHelper->GetUeDefaultGatewayAddress (), 1);

    for (uint32_t b = 0; b < numBearersPerUe; ++b)
      {
        ++dlPort;
        ++ulPort;
        ApplicationContainer clientApps;
        ApplicationContainer serverApps;

        if (useUdp)
          {
            if (epcDl)
              {
                NS_LOG_LOGIC ("installing UDP DL app for UE " << u);
                UdpClientHelper dlClientHelper (ueIPInterfaces.GetAddress (u), dlPort);
                clientApps.Add (dlClientHelper.Install (remoteHost));
                PacketSinkHelper dlPacketSinkHelper ("ns3::UdpSocketFactory",
                                                     InetSocketAddress (Ipv4Address::GetAny (), dlPort));
                serverApps.Add (dlPacketSinkHelper.Install (ue));
              }
            if (epcUl)
              {
                NS_LOG_LOGIC ("installing UDP UL app for UE " << u);
                UdpClientHelper ulClientHelper (remoteHostAddr, ulPort);
                clientApps.Add (ulClientHelper.Install (ue));
                PacketSinkHelper ulPacketSinkHelper ("ns3::UdpSocketFactory",
                                                     InetSocketAddress (Ipv4Address::GetAny (), ulPort));
                serverApps.Add (ulPacketSinkHelper.Install (remoteHost));
              }
          }
        else // use TCP
          {
            if (epcDl)
              {
                NS_LOG_LOGIC ("installing TCP DL app for UE " << u);
                BulkSendHelper dlClientHelper ("ns3::TcpSocketFactory",
                                               InetSocketAddress (ueIPInterfaces.GetAddress (u), dlPort));
                dlClientHelper.SetAttribute ("MaxBytes", UintegerValue (0));
                clientApps.Add (dlClientHelper.Install (remoteHost));
                PacketSinkHelper dlPacketSinkHelper ("ns3::TcpSocketFactory",
                                                     InetSocketAddress (Ipv4Address::GetAny (), dlPort));
                serverApps.Add (dlPacketSinkHelper.Install (ue));
              }
            if (epcUl)
              {
                NS_LOG_LOGIC ("installing TCP UL app for UE " << u);
                BulkSendHelper ulClientHelper ("ns3::TcpSocketFactory",
                                               InetSocketAddress (remoteHostAddr, ulPort));
                ulClientHelper.SetAttribute ("MaxBytes", UintegerValue (0));
                clientApps.Add (ulClientHelper.Install (ue));
                PacketSinkHelper ulPacketSinkHelper ("ns3::TcpSocketFactory",
                                                     InetSocketAddress (Ipv4Address::GetAny (), ulPort));
                serverApps.Add (ulPacketSinkHelper.Install (remoteHost));
              }
          }

        Ptr<EpcTft> tft = Create<EpcTft> ();
        if (epcDl)
          {
            EpcTft::PacketFilter dlpf;
            dlpf.localPortStart = dlPort;
            dlpf.localPortEnd = dlPort;
            tft->Add (dlpf);
          }
        if (epcUl)
          {
            EpcTft::PacketFilter ulpf;
            ulpf.remotePortStart = ulPort;
            ulpf.remotePortEnd = ulPort;
            tft->Add (ulpf);
          }

        if (epcDl || epcUl)
          {
            EpsBearer bearer (EpsBearer::NGBR_VIDEO_TCP_DEFAULT);
            lteHelper->ActivateDedicatedEpsBearer (ueDevs.Get (u), bearer, tft);
          }
        Time startTime = Seconds (startTimeSeconds->GetValue ());
        serverApps.Start (startTime);
        clientApps.Start (startTime);
```

```
            }
        }

    }
  else
    {
      for (uint32_t u = 0; u < ueDevs.GetN (); ++u)
        {
          Ptr<NetDevice> ueDev = ueDevs.Get (u);
          for (uint32_t b = 0; b < numBearersPerUe; ++b)
            {
              enum EpsBearer::Qci q = EpsBearer::NGBR_VIDEO_TCP_DEFAULT;
              EpsBearer bearer (q);
              lteHelper->ActivateDataRadioBearer (ueDev, bearer);
            }
        }
    }

  BuildingsHelper::MakeMobilityModelConsistent ();

  // generate REM without running simulation.
  Ptr<RadioEnvironmentMapHelper> remHelper;
  if (generateRem)
    {
      buildingsPrintListToFile ("BuildingsList.txt");
      eNBsPrintListToFile ("eNBsList.txt");
      uesPrintListToFile ("UEsList.txt");

      remHelper = CreateObject<RadioEnvironmentMapHelper> ();
      remHelper->SetAttribute ("ChannelPath", StringValue ("/ChannelList/0"));
      remHelper->SetAttribute ("OutputFile", StringValue ("RadioEnvironmentMap.rem"));
      remHelper->SetAttribute ("XMin", DoubleValue (topoBounds.xMin));
      remHelper->SetAttribute ("XMax", DoubleValue (topoBounds.xMax));
      remHelper->SetAttribute ("XRes", UintegerValue (200));
      remHelper->SetAttribute ("YMin", DoubleValue (topoBounds.yMin));
      remHelper->SetAttribute ("YMax", DoubleValue (topoBounds.yMax));
      remHelper->SetAttribute ("YRes", UintegerValue (200));
      remHelper->SetAttribute ("Z", DoubleValue (1.5));

      if (remRbId >= 0)
        {
          remHelper->SetAttribute ("UseDataChannel", BooleanValue (true));
          remHelper->SetAttribute ("RbId", IntegerValue (remRbId));
        }
      remHelper->Install ();
    }
  else
    {
      Simulator::Stop (Seconds (simTime));
    }

  //lteHelper->EnablePhyTraces ();
  //lteHelper->EnableMacTraces ();
  //lteHelper->EnableRlcTraces ();

  if (epc)
    {
      lteHelper->AddX2Interface (enbNodes);
      lteHelper->EnablePdcpTraces ();
    }

  CustomTracesHelper customTraces;
  customTraces.EnableHandoverTraces ();
  customTraces.EnableMobilityTraces ();

  Simulator::Run ();
  lteHelper = 0;
  Simulator::Destroy ();

  if (!generateRem)
    {
      time(&toc);
      std::cout << "Simulation time: " << simTime << "s" << std::endl;
      std::cout << "Execution time: " << difftime (toc, tic) << "s" << std::endl;
    }

  return 0;
}
```

APPENDIX B

\- - -

Simulation Parameters

Parameter	A2 / A4 High	A2 / A4 Low	A3 Low / Short	A3 High / Long
macroCellsX_cnt	1	1	1	1
macroCellsY_cnt	2	2	2	2
macroCellsZ	30	30	30	30
macroCells_spc	500	500	500	500
gridMargin_fact	0.25	0.25	0.25	0.25
macroCellTxPowerDbm	46.0	46.0	46.0	46.0
macroCellDlEarfcn	100	100	100	100
macroCellBandwidth	25	25	25	25
smallSites_cnt	1	1	1	1
smallCellsZ	6	6	6	6
smallCells_spc	20	20	20	20
smallCellTxPowerDbm	20.0	20.0	20.0	20.0
smallCellDlEarfcn	100	100	100	100
smallCellBandwidth	25	25	25	25
ueNodes_cnt	1	1	1	1
ueZ	1.75	1.75	1.75	1.75
numBearersPerUe	1	1	1	1
ueMinSpeed	0.5	0.5	0.5	0.5
ueMaxSpeed	3.5	3.5	3.5	3.5
ueRangeMargin	30.0	30.0	30.0	30.0
ueRangeConstrain	true	true	true	true
blkDensity	0.2	0.2	0.2	0.2
blkX_dim	80	80	80	80
blkY_dim	80	80	80	80
stWidth	20	20	20	20
blkZmin_dim	6.0	6.0	6.0	6.0
blkZmax_dim	15.0	15.0	15.0	15.0
simTime	900.0	900.0	900.0	900.0
epc	true	true	true	true
epcDl	true	true	true	true
epcUl	true	true	true	true
useUdp	true	true	true	true
srsPeriodicity	80	80	80	80

Parameter	A2 / A4 High	A2 / A4 Low	A3 Low / Short	A3 High / Long
A3Hysteresis	-	-	1	3
A3TimeToTrigger	-	-	100	250
A2A4SrvCellThresh	30	20	-	-
A2A4NbrCellOffset	1	1	-	-
handoverAlgorithm	A2A4	A2A4	A3	A3
generateRem	false	false	false	false
remRbId	-1	-1	-1	-1