Update 3

Jon Edwards and YoungHyo Kim

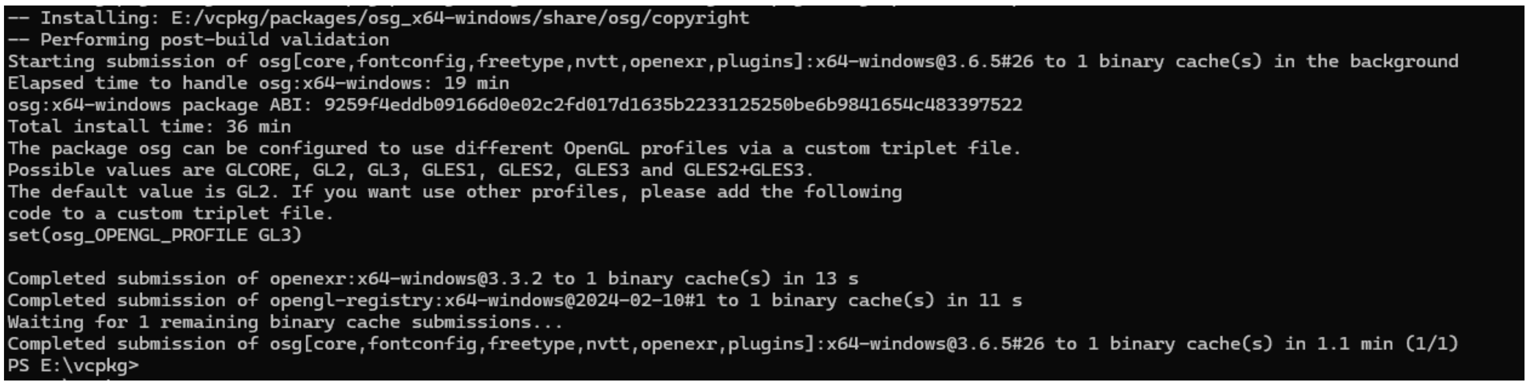
Intro

One of the key applications we explored is a long-distance communication relay between ground stations, specifically from Victoria to Vancouver. This scenario presents a fundamental challenge in balancing line-of-sight (LOS) communication constraints with the optimal positioning of each UAV within the relay network. Since UAVs serve as intermediate nodes, their placement and movement have been strategically planned to ensure continuous and efficient data transmission. To make improvements, multiple testing parameters were evaluated, including optimal UAV spacing, flight path optimization, and latency minimization strategies. These elements will guide the refinement of our relay model in future simulation phases. These enhancements aim to increase network resilience and ensure reliable UAV network under constrained conditions.

Successful Setup of OSG and osgEarth

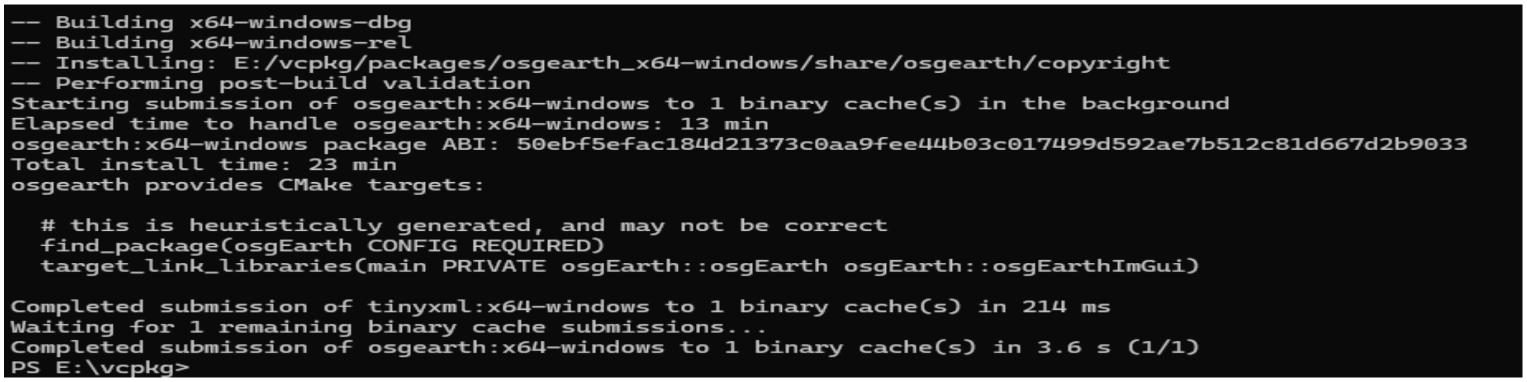
After numerous unsuccessful attempts, we finally installed OpenSceneGraph (OSG) and osgEarth using the vcpkg package manager. The process required careful configuration and repeated troubleshooting to resolve compatibility and dependency issues. As shown in the images below, the osg and osgEarth libraries were compiled and installed successfully on the Windows platform.

- **Figure 1** (below): Confirmation of successful installation of `osg:x64-windows` via vcpkg.



```
-- Installing: E:/vcpkg/packages/osg_x64-windows/share/osg/copyright
-- Performing post-build validation
Starting submission of osg[core,fontconfig,freetype,nvtt,openexr,plugins]:x64-windows@3.6.5#26 to 1 binary cache(s) in the background
Elapsed time to handle osg:x64-windows: 19 min
osg:x64-windows package ABI: 9259f4eddb09166d0e02c2fd017d1635b2233125250be6b9841654c483397522
Total install time: 36 min
The package osg can be configured to use different OpenGL profiles via a custom triplet file.
Possible values are GLCORE, GL2, GL3, GLES1, GLES2, GLES3 and GLES2+GLES3.
The default value is GL2. If you want use other profiles, please add the following
code to a custom triplet file.
set(osg_OPENGL_PROFILE GL3)

Completed submission of openexr:x64-windows@3.3.2 to 1 binary cache(s) in 13 s
Completed submission of opengl-registry:x64-windows@2024-02-10#1 to 1 binary cache(s) in 11 s
Waiting for 1 remaining binary cache submissions...
Completed submission of osg[core,fontconfig,freetype,nvtt,openexr,plugins]:x64-windows@3.6.5#26 to 1 binary cache(s) in 1.1 min (1/1)
PS E:\vcpkg>
```

- **Figure 2** (below): Confirmation of successful installation of osgearth:x64-windows via vcpkg.

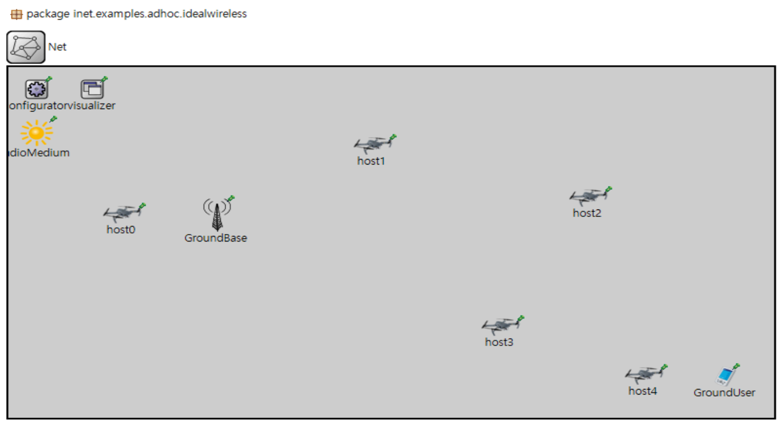


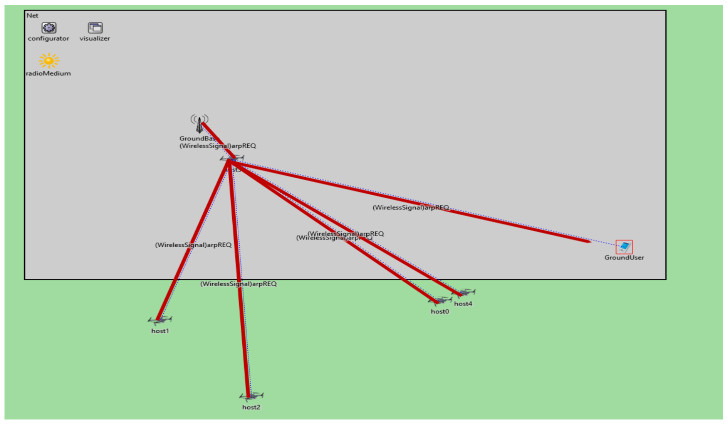
```
-- Building x64-windows-dbg
-- Building x64-windows-rel
-- Installing: E:/vcpkg/packages/osgearth_x64-windows/share/osgearth/copyright
-- Performing post-build validation
Starting submission of osgearth:x64-windows to 1 binary cache(s) in the background
Elapsed time to handle osgearth:x64-windows: 13 min
osgearth:x64-windows package ABI: 50ebf5efac184d21373c0aa9fee44b03c017499d592ae7b512c81d667d2b9033
Total install time: 23 min
osgearth provides CMake targets:

  # this is heuristically generated, and may not be correct
  find_package(osgEarth CONFIG REQUIRED)
  target_link_libraries(main PRIVATE osgEarth::osgEarth osgEarth::osgEarthImGui)

Completed submission of tinyxml:x64-windows to 1 binary cache(s) in 214 ms
Waiting for 1 remaining binary cache submissions...
Completed submission of osgearth:x64-windows to 1 binary cache(s) in 3.6 s (1/1)
PS E:\vcpkg>
```

Following this, we successfully configured these libraries within OMNeT++, allowing for 3D simulation environments and osgEarth-based visualization features to function as intended within our simulation models. This setup now provides a stable and visually rich simulation environment, enabling the integration of geospatial data and enhanced visualization for UAV and satellite network simulations.

Network Topology Before Enhancement

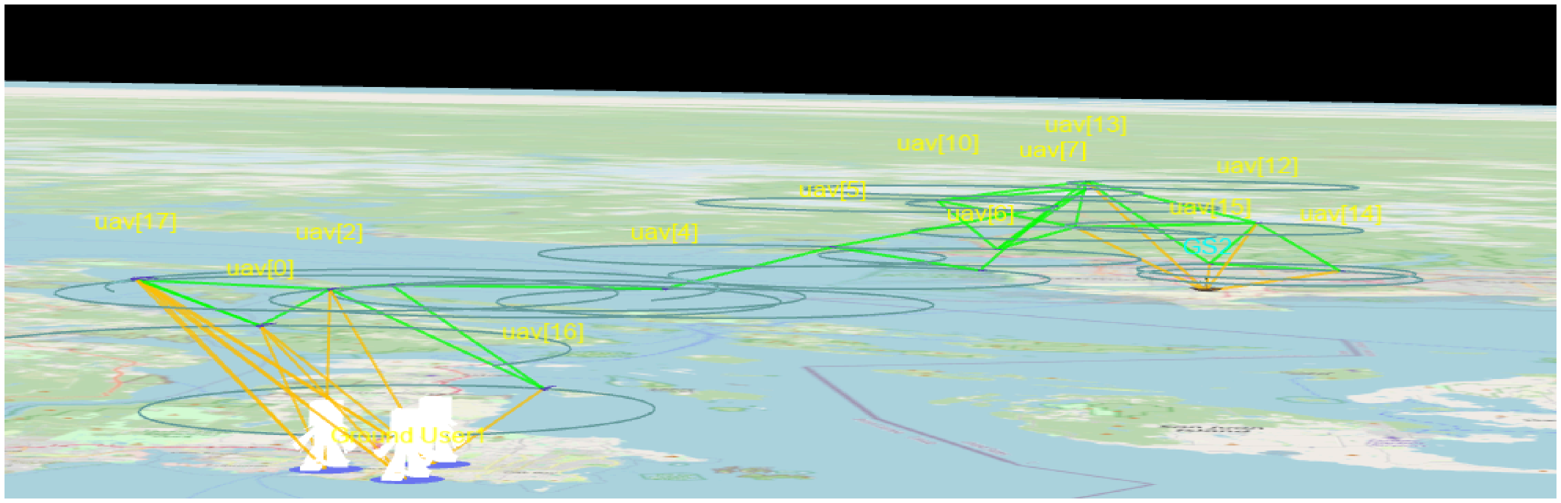
Initially, our drone network simulation in OMNeT++ was represented in a basic 2D schematic view with limited interaction and visualization capabilities. Communication was shown via standard lines with simple icons. The following images represent the initial state of the project.

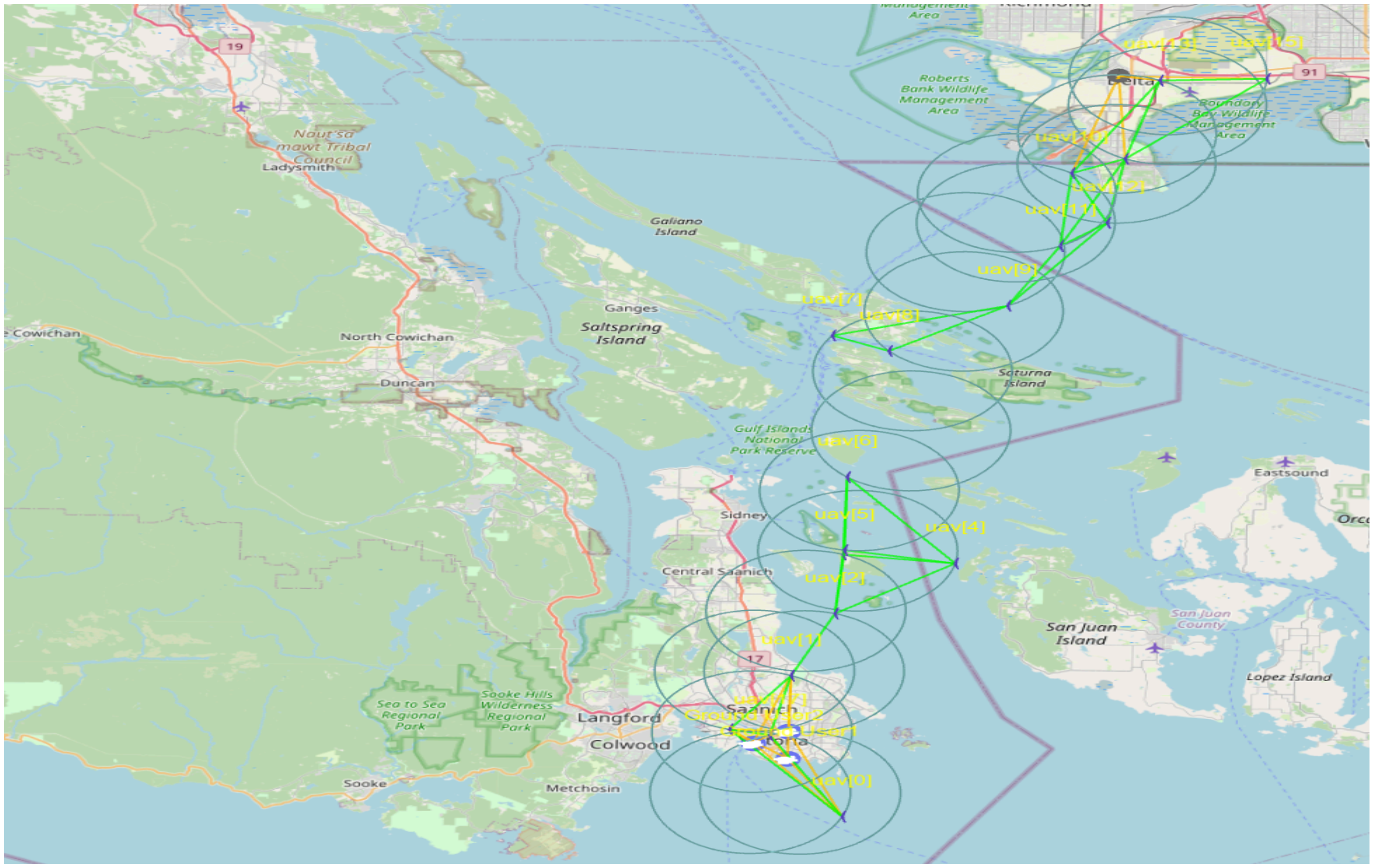


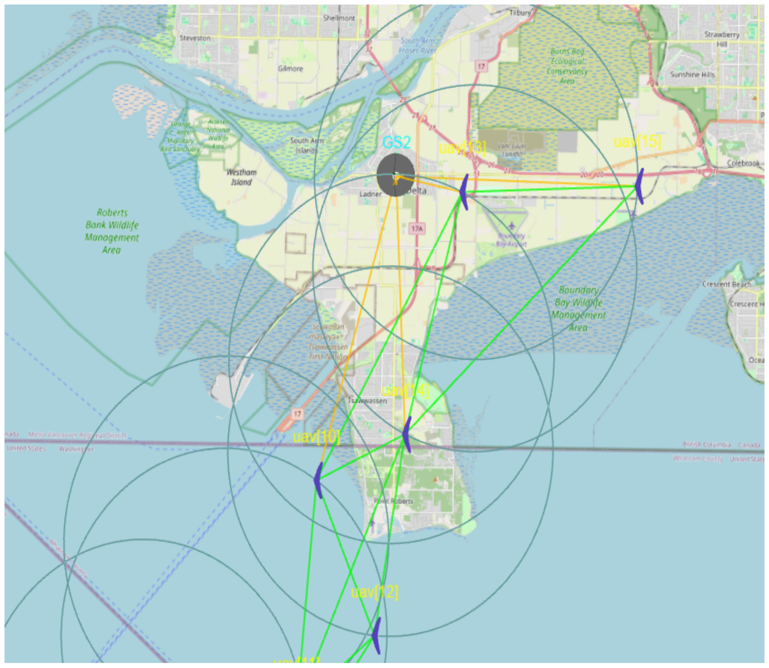


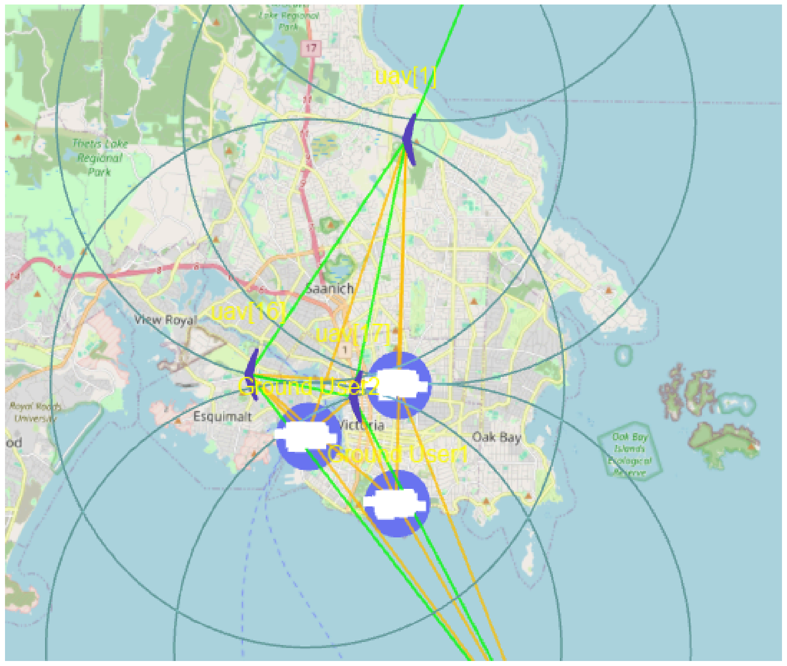
Transition to 3D Visualization with OSGEarth

After successfully configuring OSGEarth with OMNeT++, we transitioned our UAV simulation from a basic 2D schematic layout to a fully integrated 3D visualization environment using real geographic data and map tiles from OpenStreetMap. This shift allowed us to represent UAV flight paths and ground station placements accurately within real-world coordinates, significantly enhancing the realism of the simulation. UAVs now orbit along defined paths over an actual map, and their dynamic movements and connections are clearly visualized in context. Additionally, range circles and inter-UAV communication links are rendered using OSG geometry, providing a more informative and interactive view of the network. This transition has led to a substantial improvement in simulation fidelity and has greatly enhanced our ability to model realistic scenarios and visually verify network behaviors during runtime.


uav[13]
uav[10]
uav[7]
uav[12]
uav[5]
uav[15]
uav[14]
uav[6]
GS2
uav[17]
uav[2]
uav[4]
uav[0]
uav[16]
Ground User1


Richmond
Roberts Bank Wildlife Management Area
Delta
91
Boundary Bay Wildlife Management Area
19
Naut'sa mawt Tribal Council
Ladysmith
Galiano Island
Ganges
Saltspring Island
North Cowichan
Duncan
Saturna Island
Gulf Islands National Park Reserve
Sidney
Central Saanich
Eastsound
San Juan Island
San Juan County
Lopez Island
17
Saanich
Victoria
Langford
Colwood
Sea to Sea Regional Park
Sooke Hills Wilderness Regional Park
Sooke
Metchosin
uav[10]
uav[13]
uav[12]
uav[11]
uav[9]
uav[7]
uav[8]
uav[6]
uav[5]
uav[4]
uav[2]
uav[1]
uav[0]
Ground User1





Simulation setup

We simulated a scenario in which communication infrastructure on Vancouver Island was assumed to be completely connected. In this setup, three ground users were positioned in Victoria, and communication was established with a ground station located in Vancouver using 18 UAVs. All communication in the simulation was based on line-of-sight (LOS) connectivity, ensuring that data transmission only occurred when there was a direct visual link between nodes.

To model realistic constraints, the maximum communication range between UAVs and ground units was set to 10 kilometers, while the maximum communication range between UAVs was configured as 15 kilometers. Each UAV followed a circular orbit with a rotation radius of 10 kilometers centered around designated points, and their altitude was randomly assigned between 1 km and 10 km to reflect variations in flight levels.

This configuration allowed us to explore the dynamics of multi-hop UAV communication relays in a disrupted network environment, ensuring that ground users could maintain connectivity to distant ground infrastructure through airborne nodes. The use of realistic distance limits and randomized UAV altitude added complexity and practical relevance to the simulation.

Initially, the UAVs were configured with randomized altitude and starting positions using the following parameters:

```
*.uav[*].altitude = uniform(1km, 10km)
*.uav[*].startingPhase = uniform(0deg, 360deg)
```

However, this configuration led to unstable connectivity across the network, as UAVs occasionally moved beyond optimal displacement, resulting in frequent link disconnection. To address this issue, we constrained the starting phase angles of all UAVs to fall within 0 to 180 degrees, using the revised parameter:

```
*.uav[*].startingPhase = uniform(0deg, 180deg)
```

This adjustment helped stabilize the network by keeping UAVs within more favorable angular positions for maintaining consistent line-of-sight communication, thereby improving overall connectivity and reducing the occurrence of signal loss.

Network Connectivity Monitoring and Stability Analysis

To evaluate the stability and reliability of the UAV-assisted communication network, we continuously monitored the number of active connections between UAVs and between UAVs and ground units throughout the simulation. The simulation log revealed dynamic yet consistent connectivity patterns, with UAV-to-UAV links ranging from 28 to 33 active connections and UAV-to-ground links varying between 17 and 18.

For instance, at the start of the simulation, the network established **28 UAV-to-UAV and 17 UAV-to-ground connections**, which gradually increased and stabilized, occasionally peaking at **33 UAV-to-UAV and 18 UAV-to-ground connections**. This fluctuation indicates an adaptive and resilient multi-hop communication structure, effectively maintaining connectivity despite UAV mobility and variable altitude.

The following sample log excerpt illustrates this evolution:

```
Initializing module OsgEarthNet.channelController, stage 2
INFO: Active connections: 28 UAV-to-UAV and 17 UAV-to-ground
INFO: Active connections: 28 UAV-to-UAV and 17 UAV-to-ground
INFO: Active connections: 28 UAV-to-UAV and 17 UAV-to-ground
INFO: Active connections: 28 UAV-to-UAV and 17 UAV-to-ground
INFO: Active connections: 29 UAV-to-UAV and 17 UAV-to-ground I
INFO: Active connections: 29 UAV-to-UAV and 17 UAV-to-ground I
INFO: Active connections: 30 UAV-to-UAV and 17 UAV-to-ground
INFO: Active connections: 31 UAV-to-UAV and 18 UAV-to-ground
INFO: Active connections: 33 UAV-to-UAV and 17 UAV-to-ground
...
** Event #1 t=60 OsgEarthNet.clock (Clock, id=26) on selfmsg timer (omnetpp::cMessage, id=0)
INFO (ChannelController)OsgEarthNet.channelController: Active connections: 31 UAV-to-UAV and 18 UAV-to-ground
INFO (ChannelController)OsgEarthNet.channelController: Active connections: 30 UAV-to-UAV and 17 UAV-to-ground
INFO (ChannelController)OsgEarthNet.channelController: Active connections: 30 UAV-to-UAV and 18 UAV-to-ground
```

These results confirm that the UAV network not only maintained robust link availability but also dynamically adapted to node positioning to sustain communication. This stability was particularly crucial in the scenario of disrupted terrestrial infrastructure, as it ensured reliable data relay between isolated ground users and remote base stations.

Summary

So far, we successfully configured and simulated a UAV-based communication network using OMNeT++ integrated with OSGEarth. We addressed installation challenges by utilizing `vcpkg` to install and configure OpenSceneGraph (OSG) and OSGEarth, enabling real-time 3D visualization of UAV orbits and ground stations over real geographic maps.

We implemented line-of-sight (LOS) communication constraints, randomized UAV altitude and starting phase, and adjusted parameters to improve network stability. Through iterative testing, we optimized initial configurations and observed the dynamic behavior of UAV-to-UAV and UAV-to-ground communication links.

Future Work

In the next phase of our project, we plan to simulate optimal UAV displacement strategies and routing protocols as proposed in existing research papers. These simulations will help us evaluate the performance of various UAV coordination schemes under real-world constraints, with a focus on optimizing connectivity and reducing latency. The upcoming updates will involve implementing these protocols and analyzing their effectiveness within our simulation environment.

Reference Studies

1. The optimal placement for caching UAV-assisted mobile relay communication

This study investigates optimal UAV positioning to maximize data throughput in a half-duplex relay system. The authors propose a closed-form solution that optimizes UAV placement while considering caching strategies to improve overall communication efficiency.

2. Efficient deployment of multiple unmanned aerial vehicles for optimal wireless coverage

The paper addresses UAV relay placement in long-distance communication scenarios using modified Bellman-Ford and Dual Ascent algorithms. It emphasizes optimal relay chain formation and line-of-sight (LOS) maintenance under real-world constraints.

3. Routing Protocol Strategies for Flying Ad-Hoc Network

This comprehensive review categorizes FANET routing protocols (proactive, reactive, hybrid, position-based) and analyzes their strengths and limitations in high-mobility 3D environments. Protocols like ZRP, TORA, and OLSR are compared in depth.

[1] F. Liang, J. Zhang, B. Li, Z. Yang and Y. Wu, "The Optimal Placement for Caching UAV-assisted Mobile Relay Communication," 2019 IEEE 19th International Conference on Communication Technology (ICCT), Xi'an, China, 2019, pp. 540-544, doi: 10.1109/ICCT46805.2019.8947051.

[2] M. Mozaffari, W. Saad, M. Bennis and M. Debbah, "Efficient Deployment of Multiple Unmanned Aerial Vehicles for Optimal Wireless Coverage," in IEEE Communications Letters, vol. 20, no. 8, pp. 1647-1650, Aug. 2016, doi: 10.1109/LCOMM.2016.2578312.

[3] I. Bekmezci, O. K. Sahingoz and Ş. Temel, "Flying Ad-Hoc Networks (FANETs): A survey," Ad Hoc Networks, vol. 11, no. 3, pp. 1254–1270, May 2013, doi: 10.1016/j.adhoc.2012.12.004.

Source Codes

The implementation of the UAV orbit simulation and visualization was developed by modifying sample code originally provided with OMNeT++. Specifically, we adapted the existing satellite simulation modules (`Satellite.cc` and `Satellite.h`) to fit our custom UAV network scenario. This included implementing real-time position updates based on line-of-sight communication constraints, customizing orbit parameters such as radius and altitude, and enhancing visualization through OSGEarth integration. The modified code reflects both our adaptation of existing modules and the addition of new functionalities tailored to our simulation goals.

Satellite.cc

```
#include "Satellite.h"
#ifdef WITH_OSGEARTH
#include "OsgEarthScene.h"
#include "ChannelController.h"
#include <sstream>
#include <iomanip>
#include "omnetpp/osgutil.h"
#include <osg/Node>
#include <osg/Texture>
#include <osg/LineWidth>
#include <osg/PolygonMode>
#include <osg/Texture2D>
#include <osg/Image>
#include <osg/Depth>
#include <osg/PositionAttitudeTransform>
#include <osgEarth/Capabilities>
#include <osgEarthAnnotation/LabelNode>
#include <osgEarthSymbology/Geometry>
#include <osgEarthFeatures/Feature>
using namespace omnetpp;
Define_Module(Satellite)
using namespace osgEarth;
using namespace osgEarth::Annotation;
using namespace osgEarth::Features;
void Satellite::initialize(int stage) {
  switch (stage) {
  case 0: {
     modelURL = par("modelURL").stringValue();
     modelScale = par("modelScale");
     labelColor = par("labelColor").stringValue();
     altitude = par("altitude");
     double phaseDeg = par("startingPhase").doubleValueInUnit("deg");
     phase = startingPhase = phaseDeg * M_PI / 180.0;
     std::string centerStr = par("orbitCenter").stringValue();
     std::stringstream ss(centerStr);
     double cx, cy;
     ss >> cx >> cy;
     if (!ss)
        throw cRuntimeError("orbitCenter parsing fail: \"%s\"", centerStr.c_str());
     orbitCenter.set(cx, cy);
     getOsgCanvas()->setScene(osgDB::readNodeFile(modelURL));
     break;
  }
  case 1: {
     ChannelController::getInstance()->addSatellite(this);
     auto scene = OsgEarthScene::getInstance()->getScene();
     mapNode = osgEarth::MapNode::findMapNode(scene);
     geoTransform = new osgEarth::GeoTransform();
     auto modelNode = osgDB::readNodeFile(modelURL);
     modelNode->getOrCreateStateSet()->setMode(GL_LIGHTING,
osg::StateAttribute::OFF | osg::StateAttribute::OVERRIDE);
     auto pat = new osg::PositionAttitudeTransform();
     pat->setScale(osg::Vec3d(modelScale, modelScale, modelScale));
     auto objectNode = new cObjectOsgNode(this);
     pat->addChild(objectNode);
     objectNode->addChild(modelNode);
     geoTransform->addChild(pat);
     if (!labelColor.empty()) {
        Style labelStyle;
        labelStyle.getOrCreate<TextSymbol>()->alignment() =
TextSymbol::ALIGN_CENTER_TOP;
        labelStyle.getOrCreate<TextSymbol>()->declutter() = true;
        labelStyle.getOrCreate<TextSymbol>()->pixelOffset() = osg::Vec2s(0, 50);
        labelStyle.getOrCreate<TextSymbol>()->fill()->color() = osgEarth::Color(labelColor);
        geoTransform->addChild(new osgEarth::Annotation::LabelNode(getFullName(),
labelStyle));
```

Satellite.h

```
#ifndef __MOBILENODE_H__
#define __MOBILENODE_H__
#include <omnetpp.h>
#ifdef WITH_OSGEARTH
#include "OsgEarthScene.h"
#include <osgEarth/MapNode>
#include <osgEarth/GeoTransform>
#include <osgEarthAnnotation/CircleNode>
#include <osgEarthAnnotation/FeatureNode>
using namespace omnetpp;
/**
* Satellite model. See NED file for details.
*/
class Satellite : public cSimpleModule
{
 protected:
  // configuration
  double startingPhase;
  std::string labelColor;
  std::string modelURL;
  double modelScale;
  // the node containing the osgEarth data
  osg::observer_ptr<osgEarth::MapNode> mapNode = nullptr;
  // osgEarth node for 3D visualization
  osgEarth::GeoTransform *geoTransform = nullptr;
  const double mu = 398600.4418; // "geocentric gravitational
constant" - source: wikipedia, units: km^3 / s^2
  const double earthRadius = 6371; // in km
  double altitude = 10000; // in km, above the surface
  double phase = 0; // on the orbit, in radians, unbounded
  osg::Vec3d normal = osg::Vec3d(0, 0, 1); // doesn't have to be unit
length, just can't be 0
  osg::Vec3d orbitX, orbitY; // the base of the orbit plane, orthogonal,
and both are unit length, computed from the normal
  osg::Vec3d pos;
  simtime_t lastPositionUpdateTime = -1;
 public:
  osg::Node *getLocatorNode() { return geoTransform; };
  void updatePosition();
 protected:
  // angular velocity in rad/sec
  double getOmega() { return std::sqrt(mu / std::pow(altitude +
earthRadius, 3)); }
  // in world coordinates, units is meters, phase is the angle on the orbit
in radians
  osg::Vec3 getPositionAtPhase(double alpha) const;
  virtual void initialize(int stage) override;
  virtual int numInitStages() const override { return 2; }
  virtual void handleMessage(cMessage *msg) override;
  virtual void refreshDisplay() const override;
};
#endif
#endif
```

```
    }
    scene->asGroup()->addChild(geoTransform);
    std::string orbitColor = par("orbitColor").stringValue();
    if (!orbitColor.empty()) {
      osg::ref_ptr<osg::Geometry> orbitGeom = new osg::Geometry;
      osg::ref_ptr<osg::DrawArrays> drawArrayLines = new
osg::DrawArrays(osg::PrimitiveSet::LINE_STRIP);
      osg::ref_ptr<osg::Vec3Array> vertexData = new osg::Vec3Array;
      double radius = 10000;
      for (int i = 0; i <= 100; ++i) {
        double angle = i / 100.0 * 2 * M_PI;
        double x = orbitCenter.x() + std::cos(angle) * radius;
        double y = orbitCenter.y() + std::sin(angle) * radius;
        double z = altitude * 1000;  // m
        vertexData->push_back(osg::Vec3(x, y, z));
      }
      orbitGeom->addPrimitiveSet(drawArrayLines);
      orbitGeom->setVertexArray(vertexData);
      drawArrayLines->setFirst(0);
      drawArrayLines->setCount(vertexData->size());
      osg::ref_ptr<osg::Vec4Array> colorData = new osg::Vec4Array;
      colorData->push_back(osgEarth::Color(orbitColor));
      orbitGeom->setColorArray(colorData, osg::Array::BIND_OVERALL);
      auto stateSet = orbitGeom->getOrCreateStateSet();
      stateSet->setMode(GL_BLEND, osg::StateAttribute::ON);
      stateSet->setMode(GL_LINE_SMOOTH, osg::StateAttribute::ON);
      stateSet->setMode(GL_LIGHTING, osg::StateAttribute::OFF);
      stateSet->setAttributeAndModes(new osg::LineWidth(1.5), osg::StateAttribute::ON);
      osg::ref_ptr<osg::Geode> orbitGeode = new osg::Geode;
      orbitGeode->addDrawable(orbitGeom);
      scene->asGroup()->addChild(orbitGeode);
    }
    break;
  }
  }
}
void Satellite::handleMessage(cMessage *msg)
{
  throw cRuntimeError("This module does not process messages");
}
void Satellite::updatePosition() {
  simtime_t t = simTime();
  if (t != lastPositionUpdateTime) {

    double rotationPeriod = 120.0;

    double angle = startingPhase + (t.dbl() / rotationPeriod) * 2 * M_PI;
    double radius = 10000;  /
    double centerX = orbitCenter.x();
    double centerY = orbitCenter.y();
    double x = centerX + std::cos(angle) * radius;
    double y = centerY + std::sin(angle) * radius;
    double z = altitude * 1000;  //km->m
    pos.set(x, y, z);
    osg::Vec3d v;
    mapNode->getMapSRS()->transformFromWorld(pos, v);
    geoTransform->setPosition(osgEarth::GeoPoint(mapNode->getMapSRS(), v));
    lastPositionUpdateTime = t;
  }
}
osg::Vec3 Satellite::getPositionAtPhase(double alpha) const
{
  return (orbitX * std::cos(alpha) + orbitY * std::sin(alpha)) * (earthRadius + altitude) *
1000;
}
void Satellite::refreshDisplay() const
{
  const_cast<Satellite *>(this)->updatePosition();
  // update the position on the 2D canvas
  getDisplayString().setTagArg("p", 0, 300 + pos.x() / 100000);
  getDisplayString().setTagArg("p", 1, 300 - pos.y() / 100000);
}
#endif // WITH_OSG
```

ChannelController.cc

```
//
// This file is part of an OMNeT++/OMNEST simulation example.
//
// Copyright (C) 2015 OpenSim Ltd.
//
// This file is distributed WITHOUT ANY WARRANTY. See the file
// `license' for details on this and other legal matters.
//
#include "ChannelController.h"
#ifdef WITH_OSGEARTH
#include <osgEarthUtil/LinearLineOfSight>
#include <osgUtil/UpdateVisitor>
#include <osg/ValueObject>
#include <osg/LineWidth>
#include <osg/Depth>
using namespace osgEarth;
using namespace osgEarth::Annotation;
using namespace osgEarth::Features;
Define_Module(ChannelController);
ChannelController *ChannelController::instance = nullptr;
template<typename T>
bool contains(const std::vector<T>& vector, T item) { return std::find(vector.begin(),
vector.end(), item) != vector.end(); }
ChannelController::ChannelController()
{
  if (instance)
     throw cRuntimeError("There can be only one ChannelController instance in the
network");
  instance = this;
}
ChannelController::~ChannelController()
{
  instance = nullptr;
}
ChannelController *ChannelController::getInstance()
{
  if (!instance)
     throw cRuntimeError("ChannelController::getInstance(): there is no ChannelController
module in the network");
  return instance;
}
void ChannelController::initialize(int stage)
{
  switch (stage) {
  case 0:
     satToSatColor = par("satToSatColor").stringValue();
     satToSatWidth = par("satToSatWidth").doubleValue();
     satToGroundColor = par("satToGroundColor").stringValue();
     satToGroundWidth = par("satToGroundWidth").doubleValue();
     maxSatToGroundDistance = par("maxSatToGroundDistance").doubleValue();
     maxSatToSatDistance = par("maxSatToSatDistance").doubleValue();
     break;
  case 1:
     scene = OsgEarthScene::getInstance()->getScene()->asGroup();
```

ChannelController.h

```
//
// This file is part of an OMNeT++/OMNEST simulation example.
//
// Copyright (C) 2015 OpenSim Ltd.
//
// This file is distributed WITHOUT ANY WARRANTY. See the file
// `license' for details on this and other legal matters.
//
#ifndef __CHANNELCONTROLLER_H_
#define __CHANNELCONTROLLER_H_
#include <omnetpp.h>
#ifdef WITH_OSGEARTH
#include "OsgEarthScene.h"
#include "GroundStation.h"
#include "Satellite.h"
#include <osg/Node>
#include <osgEarth/MapNode>
#include <osgEarthAnnotation/FeatureNode>
#include <osgEarthAnnotation/LocalGeometryNode>
#include <osgEarthUtil/LineOfSight>
#include <osgEarthUtil/LinearLineOfSight>
#include <osgEarthSymbology/Style>
#include <osgEarthSymbology/Geometry>
#include <osgEarthFeatures/Feature>
using namespace omnetpp;
/**
* This module is responsible for tracking the distance of mobile
nodes
* and visualizing the connectivity graph using OSG nodes.
*/
class ChannelController : public cSimpleModule
{
 protected:
  static osg::ref_ptr<osg::Drawable>
createLineBetweenPoints(osg::Vec3 start, osg::Vec3 end, float
width, osg::Vec4 color);
  static ChannelController *instance;
  std::vector<Satellite *> satellites;
  std::vector<GroundStation *> stations;
  std::vector<osgEarth::Util::LinearLineOfSightNode *> losNodes;
  std::map<Satellite *, osg::Geometry *> orbitsMap;
  osg::ref_ptr<osg::Geode> connections = nullptr;
  // visual parameters of the connections
  std::string satToSatColor;
  double satToSatWidth = 0;
  std::string satToGroundColor;
  double satToGroundWidth = 0;
  double maxSatToGroundDistance = 0;
  double maxSatToSatDistance = 0;
  // the node containing the osgEarth data
  osg::Group *scene = nullptr;
 protected:
  virtual void initialize(int stage) override;
  virtual int numInitStages() const override { return 3; }
```

```
        connections = new osg::Geode();
        scene->addChild(connections);
        break;
    case 2:
        for (int i = 0; i < (int)satellites.size(); ++i) {
            for (int j = 0; j < (int)stations.size(); ++j)
                addLineOfSight(satellites[i]->getLocatorNode(), stations[j]->getLocatorNode(), 0);
            for (int j = i+1; j < (int)satellites.size(); ++j)
                addLineOfSight(satellites[i]->getLocatorNode(), satellites[j]->getLocatorNode(), 1);
        }
        break;
    }
}
void ChannelController::addSatellite(Satellite *p)
{
    ASSERT(!contains(satellites, p));
    ASSERT(losNodes.empty()); // we are before init stage 2
    satellites.push_back(p);
}
void ChannelController::addGroundStation(GroundStation *p)
{
    ASSERT(!contains(stations, p));
    ASSERT(losNodes.empty()); // we are before init stage 2
    stations.push_back(p);
}
void ChannelController::addLineOfSight(osg::Node *a, osg::Node *b, int type)
{
    auto mapNode = osgEarth::MapNode::findMapNode(scene);
    osgEarth::Util::LinearLineOfSightNode *los = new
osgEarth::Util::LinearLineOfSightNode(mapNode);
    losNodes.push_back(los);
    // not drawing the line of sight nodes' lines
    los->setGoodColor(osg::Vec4f(0, 0, 0, 0));
    los->setBadColor(osg::Vec4f(0, 0, 0, 0));
    auto stateSet = los->getOrCreateStateSet();
    stateSet->setRenderingHint(osg::StateSet::TRANSPARENT_BIN);
    auto depth = new osg::Depth;
    depth->setWriteMask(false);
    stateSet->setAttributeAndModes(depth, osg::StateAttribute::ON);
    los->setUserValue("type", type);
    los->setUpdateCallback(new osgEarth::Util::LineOfSightTether(a, b));
    los->setTerrainOnly(true); // so the dish model itself won't occlude
    scene->addChild(los);
}
void ChannelController::refreshDisplay() const
{

    for (auto satellite : satellites)
        satellite->updatePosition();

    osgUtil::UpdateVisitor updateVisitor;
    scene->traverse(updateVisitor);

    connections->removeDrawables(0, connections->getNumDrawables());
    int numSatToSat = 0;
    int numSatToGround = 0;
    for (auto losNode : losNodes) {
        if (losNode->getHasLOS()) {
            auto start = losNode->getStartWorld();
            auto end = losNode->getEndWorld();

            double distance = (start - end).length();  //: meter
            int type;
            losNode->getUserValue("type", type);
            bool withinRange = true;
            switch (type) {
            case 0:  // uav→ ground
                withinRange = distance <= maxSatToGroundDistance;
                if (withinRange) {
```

```
    virtual void handleMessage(cMessage *msg) override;
    virtual void refreshDisplay() const override;
    void addLineOfSight(osg::Node *a, osg::Node *b, int type);
 public:
    ChannelController();
    virtual ~ChannelController();
    static ChannelController *getInstance();
    void addSatellite(Satellite *p); // to be called exactly in initialize
stage 1
    void addGroundStation(GroundStation *p); // to be called
exactly in initialize stage 1
};
#endif // WITH_OSGEARTH
#endif // __CHANNELCONTROLLER_H_
```

```
                ++numSatToGround;
                if (!satToGroundColor.empty())
                    connections->addDrawable(createLineBetweenPoints(start, end,
satToGroundWidth, osgEarth::Color(satToGroundColor)));
            }
            break;
        case 1:  // uav↔ uav
            withinRange = distance <= maxSatToSatDistance;
            if (withinRange) {
                ++numSatToSat;
                if (!satToSatColor.empty())
                    connections->addDrawable(createLineBetweenPoints(start, end,
satToSatWidth, osgEarth::Color(satToSatColor)));
            }
            break;
        }
      }
   }
   EV << "Active connections: " << numSatToSat << " UAV-to-UAV and " << numSatToGround
<< " UAV-to-ground \n";
}
osg::ref_ptr<osg::Drawable> ChannelController::createLineBetweenPoints(osg::Vec3
start, osg::Vec3 end, float width, osg::Vec4 color)
{
   osg::ref_ptr<osg::Geometry> orbitGeom = new osg::Geometry;
   osg::ref_ptr<osg::DrawArrays> drawArrayLines = new
osg::DrawArrays(osg::PrimitiveSet::LINE_STRIP);
   osg::ref_ptr<osg::Vec3Array> vertexData = new osg::Vec3Array;
   orbitGeom->addPrimitiveSet(drawArrayLines);
   auto stateSet = orbitGeom->getOrCreateStateSet();
   stateSet->setAttributeAndModes(new osg::LineWidth(width), osg::StateAttribute::ON |
osg::StateAttribute::OVERRIDE);
   stateSet->setRenderingHint(osg::StateSet::TRANSPARENT_BIN);
   auto depth = new osg::Depth;
   depth->setWriteMask(false);
   stateSet->setAttributeAndModes(depth, osg::StateAttribute::ON);
   vertexData->push_back(start);
   vertexData->push_back(end);
   orbitGeom->setVertexArray(vertexData);
   drawArrayLines->setFirst(0);
   drawArrayLines->setCount(vertexData->size());
   osg::ref_ptr<osg::Vec4Array> colorData = new osg::Vec4Array;
   colorData->push_back(osgEarth::Color(color));
   orbitGeom->setColorArray(colorData, osg::Array::BIND_OVERALL);
   return orbitGeom;
}
void ChannelController::handleMessage(cMessage *msg)
{
   throw cRuntimeError("This module does not process messages");
}
#endif // WITH_OSG
```