BURSA TEKNİK ÜNİVERSİTESİ

Mühendislik ve Doğa Bilimleri Fakültesi

Bilgisayar Mühendisliği Bölümü

Bilgisayar Ağları Dersi Projesi

Kablosuz İletişim Teknolojilerinin Performansı Hareketlilikten Nasıl Etkilenir?

Github Linki: https://github.com/Aytacus/Ns-3_Sumo_Application.git

Yücel Aytaç AKGÜN-20360859059

Osman ÇAMKERTEN-20360859024

Sanem COŞKUN- 21360859022

2023-2024 BAHAR DÖNEMİ

ÖNSÖZ

Merhabalar,

LoRaWAN kablosuz iletişim teknolojisinin hareketli ortamlarda nasıl performans gösterdiğini inceleyen bu çalışmamızı sizlerle paylaşmaktan mutluluk duyuyoruz. Günümüzde IoT uygulamalarının artmasıyla birlikte, LoRaWAN gibi teknolojilerin önemi giderek artmaktadır.

Bu projede, LoRaWAN'ın hareketli ortamlarda nasıl çalıştığını anlamak için simülasyonlar kullanıyoruz. Simülasyonların gerçekçi olması için OpenStreetMap'ten alınan verileri Sumo kullanarak daha gerçekçi hale getirdik. Amacımız, bu teknolojinin avantajlarını ve sınırlamalarını daha iyi anlamak ve gelecekteki uygulamalara katkı sağlamaktır.

Projemizin gerçekleştirilmesinde bize destek olan herkese teşekkür ederiz. Elde edeceğimiz sonuçlarla LoRaWAN teknolojisinin performansını daha iyi anlamayı ve bu alanda yapılan çalışmalara katkı sağlamayı hedefliyoruz.

İyi okumalar!

Saygılarımızla,

Yücel Aytaç AKGÜN

Osman ÇAMKERTEN

Sanem COŞKUN

Bursa 2024

İÇİNDEKİLER

1. Giriş

1.1. Ns-3 Nedir?

Ns-3 (Network Simulator version 3), bilgisayar ağlarının ve iletişim sistemlerinin simülasyonunu gerçekleştirmek için kullanılan açık kaynaklı bir simülasyon aracıdır. Bu simülasyon aracı, çeşitli ağ türlerini, iletişim protokollerini ve ağdaki farklı cihazların davranışlarını modelleyerek, gerçek dünya senaryolarını taklit etme yeteneğine sahiptir.

Ağ simülasyonu, yeni iletişim protokolleri veya ağ algoritmalarının tasarımı ve değerlendirilmesi için oldukça önemlidir. Ns-3, araştırmacıların ve mühendislerin yeni fikirleri test etmelerine, ağ performansını analiz etmelerine ve mevcut protokollerin veya algoritmaların davranışlarını incelemelerine olanak tanır.

1.2. LoRaWAN Nedir?

LoRaWAN (Long Range Wide Area Network), düşük güç tüketimi, uzun menzil ve geniş kapsama alanı gibi özellikleriyle öne çıkan kablosuz bir iletişim protokolüdür. Nesnelerin İnterneti (IoT) uygulamaları için tasarlanmış bir protokoldür ve genellikle uzun pil ömrü gerektiren cihazlarla kullanılır.

LoRaWAN, düşük bit hızlarında veri iletimi için optimize edilmiştir ve genellikle sensör ağları gibi uygulamalarda kullanılır. Bu protokol, özellikle uzak mesafelerde iletişim gerektiren uygulamalar için idealdir ve bir LoRaWAN ağı, birkaç kilometre mesafedeki cihazlar arasında veri iletişimi sağlayabilir.

1.3. SUMO Nedir?

SUMO (Simulation of Urban MObility), kentsel hareketlilik simülasyonları için kullanılan açık kaynaklı bir araçtır. SUMO, trafik akışı, araçların hareketi, trafik ışıkları ve diğer ulaşım unsurlarının modellenmesini sağlar.

SUMO, ulaşım altyapısının planlanması, trafik yönetimi stratejilerinin test edilmesi, trafik sinyallerinin optimizasyonu ve acil durum senaryolarının simülasyonu gibi birçok farklı amaç için kullanılmaktadır. Ayrıca, ulaşım mühendisliği alanında eğitim ve araştırma için de yaygın olarak kullanılmaktadır.

Bunlardan bahsetmemizin sebebi, neleri kullanacağımızı bilmenizi istememizdir.

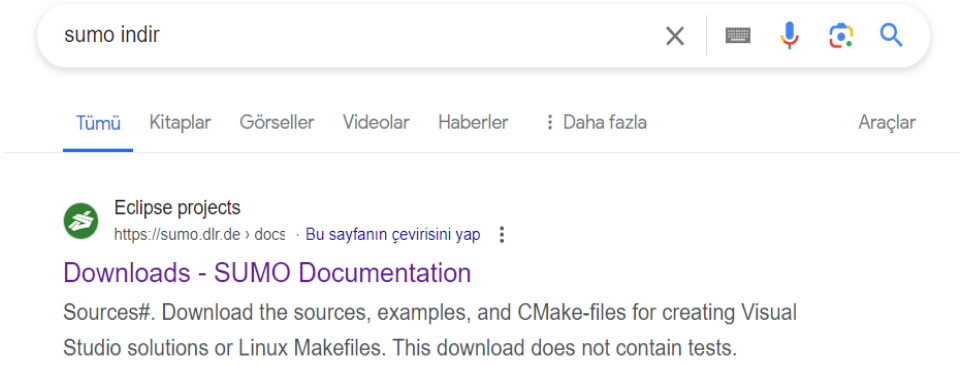
Kafa karışıklığını önlemek için yapılacak işlemler şunlardır:

1. OpenStreetMap uygulamasından bir alan seçilecek ve bu alan indirilecek.
2. İndirilen dosya, SUMO/tools içine atılıp komut satırından SUMO'ya uygun hale getirilecektir.
3. Oluşan dosya, Ns-2 dosyasına çevrilecektir (.tcl).
4. Bu işlemlerden sonra Ns-3 işlemlerine geçilecektir.

Not: İşlemler Linux üzerinden yapılacaktır (Ubuntu kullanılacaktır).

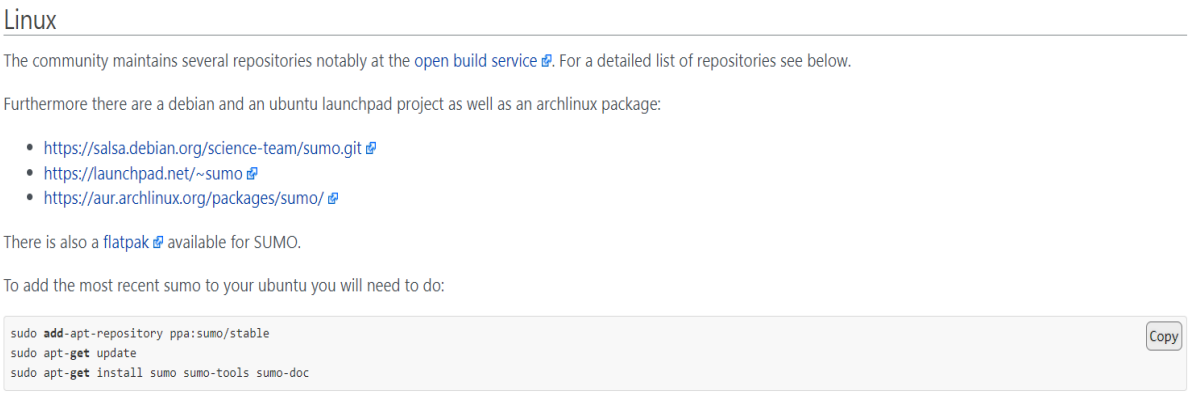
2. SUMO Kurulumu, Simülasyon Oluşturma ve Ns-2 Dosyası Oluşturma

2.1. Sumo Kurulumu



Şekil 1: *SUMO indirme.*

Google'da "sumo indir" yazdıktan sonra "Downloads – SUMO Documentation" bağlantısına tıklayınız.



Şekil 2: *SUMO Linux kurulumu.*

Linux kısmında "to add the most recent sumo to your Ubuntu you will need to do:" satırının altındaki komutları Ubuntu komut satırında çalıştırınız. (Ubuntu'ya kurulacaktır, ancak daha sonradan ihtiyacınız olan SUMO uygulamasının dosya konumuna erişemeyeceksiniz. Sanal bilgisayarlarda böyle bir sorun yaşanmıştır. Bu yüzden devam işlemlerini de uygulayın.)

Where to get it

You can download SUMO via our downloads site.

As the program is still under development (and is being extended continuously), we advice you to use the latest sources from our GitHub repository. Using a command line client, execute the following command:

```
git clone --recursive https://github.com/eclipse-sumo/sumo
```

Şekil 3: *SUMO indirme ve kurulum.*

Bu kısmın önemi şudur: Bu işlem sayesinde SUMO dosyasının içeriğini oluşturabiliyorsunuz.

Build and Installation

For Windows we provide pre-compiled binaries and CMake files to generate Visual Studio projects. If you want to develop under Windows, please also clone the dependent libraries using:

```
git clone --recursive https://github.com/DLR-TS/SUMOLibraries
```

If you're using Linux, you should have a look whether your distribution already contains sumo. There is also a ppa for ubuntu users and an open build service instance. If you want to build SUMO yourself, the steps for ubuntu are:

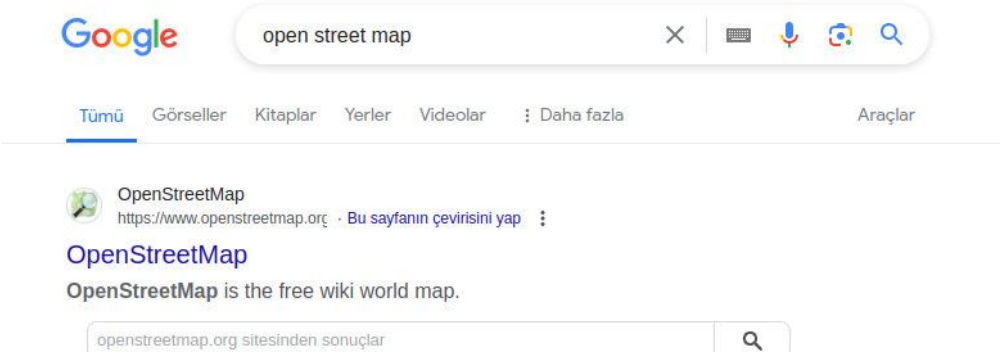
```
sudo apt-get install cmake python g++ libxerces-c-dev libfox-1.6-dev libgdal-dev libproj-dev libgl2ps-d
cd <SUMO_DIR> # please insert the correct directory name here
export SUMO_HOME="$PWD"
cmake -B build .
cmake --build build -j$(nproc)
```

Şekil 4: *SUMO derleme ve kurulum.*

Bu kısımda SUMO'yu kuruyor. Şimdi yaşadığımız sıkıntı şuydu: Şekil 4 başarılı olamadığı için SUMO.exe oluşmadı. Bunun çözümü ise önce Şekil 2 ile SUMO'yu kurmak oldu. Ardından, Şekil 3 ve Şekil 4 sayesinde SUMO uygulamasının dosya konumunda neler bulunduğunu öğrenmiş olduk. Yani, bir tanesi çalıştırma için, diğeri ise SUMO klasörünün içeriğini ve neler bulunduğunu göstermek için kullanıldı. SUMO kurulumu bu kadardı. Karışık gelmiş olabilir ve eğer bilgisayarınızda sorun çıkmayacağını düşünüyorsanız, Şekil 3 ve Şekil 4'ü uygulayın.

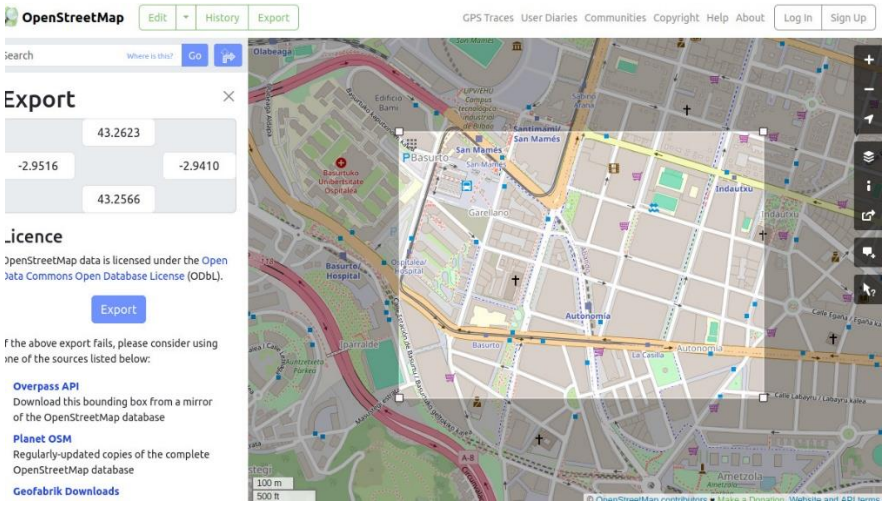
2.2. Simülasyon Oluşturma

Öncelikle, buradan sonraki kısımlarda Linux işletim sisteminizde Ns-3 ve LoRaWAN kurulu olduğu varsayılmıştır. Konuya dönecek olursak, adım adım nasıl simülasyon ve Ns-2 dosyası oluşturulacağı anlatılacaktır.



Şekil 5: *OpenStreetMap site girişi.*

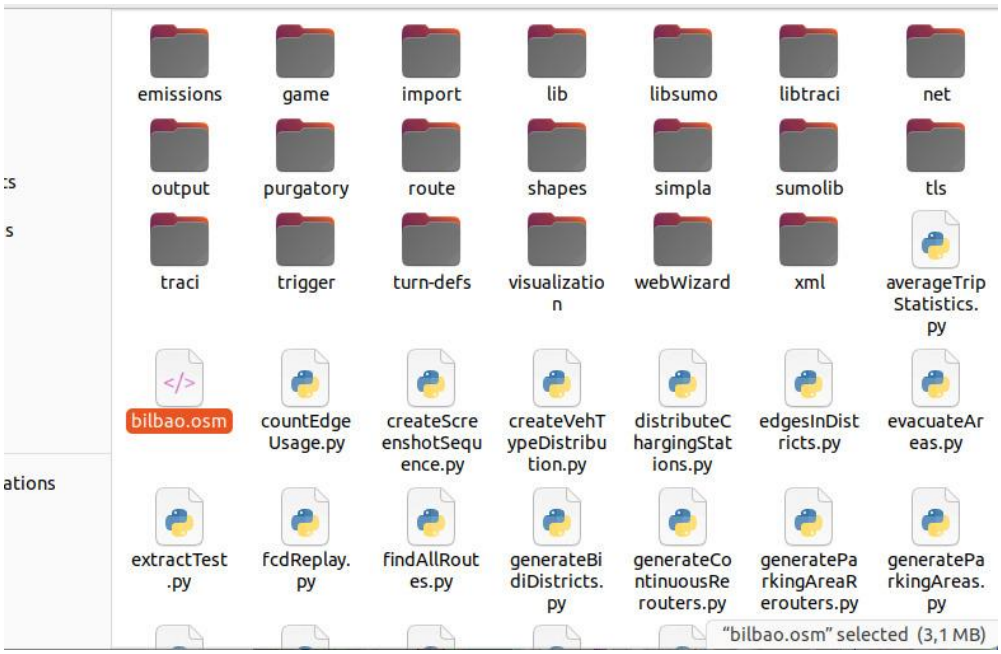
Öncelikle Google'da "OpenStreetMap" yazıp, Şekil 5'te gösterilen siteye giriniz.



Şekil 6: *OpenStreetMap alan seçimi.*

Burada istediğiniz bir şehrin bir kısmını alabilirsiniz. Karşınıza normal bir harita gelecek. Alanı seçip dışa aktarmak için yeşil renkli "Export" düğmesine tıklayın. Tıkladıktan sonra, Şekil 6'da gösterildiği gibi ekranınızın solunda "Export" bölümü açılacak. Oradaki "manuel ayarla" tuşuna tıkladığınızda istediğiniz alanı seçebilirsiniz.

Biz bu projede Bilbao şehrinden bir bölge seçtik. İndirdiğimiz dosyanın adını bilbao.osm yaptık (Normalde map.osm oluyordu).



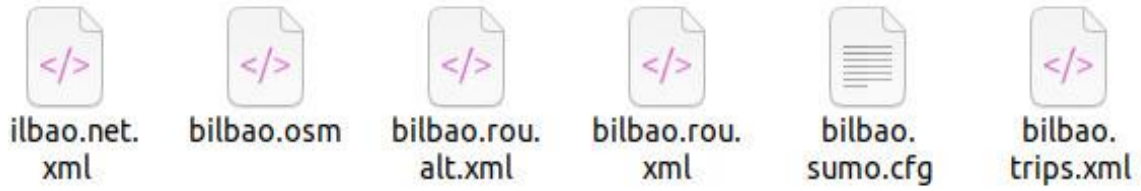
Şekil 7: *SUMO tools klasörü.*

İnen dosyayı sumo/tools klasörünün içine attık. Buraya kadar olan kısımda sadece bir alan oluşturduk.

```
netconvert --osm-files bilbao.osm --output-file bilbao.net.xml --geometry.remove --roundabouts.guess --ramps.guess --junctions.join
 --tls.guess-signals --tls.discard-simple --tls.join

python randomTrips.py -n bilbao.net.xml -e 300 -o bilbao.trips.xml

duarouter -n bilbao.net.xml --route-files bilbao.trips.xml -o bilbao.rou.xml --ignore-errors
```

Şekil 8: *Terminal komutları.*

Terminali açıp önce cd sumo/ yazarak SUMO'nun klasörüne gitmelisiniz. Ondan sonra cd tools/ yazarak sumo/tools klasörüne geçmelisiniz. Bu işlemlerden sonra Şekil 8'deki komutları tek tek Shell'de(komut terminali/terminal) çalıştırınız. Buradaki -e 300 ise 300 tane araba üretmeye çalışıyor (Her zaman üretemeyebilir, bu rakam bizim durumumuzda 238 çıktı).



Şekil 9: *SUMO dosyaları.*

Şekil 8'deki komutları yazdıktan sonra bilbao.net.xml, bilbao.rou.alt.xml, bilbao.rou.xml ve bilbao.trips.xml dosyaları oluştu. bilbao.sumo.cfg dosyasını ise biz kendimiz oluşturduk ve içerisine şu bilgileri girdik**:**

```
<?xml version="1.0" encoding="iso-8859-1"?>

<configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="http://sumo.sf.net/xsd/
sumoConfiguration.xsd">

<input>
    <net-file value="bilbao.net.xml"/>
    <route-files value="bilbao.rou.xml"/>
</input>

<time>
    <begin value="0"/>
    <end value="2500"/>
</time>

</configuration>
```

Şekil 10: *SUMO konfigürasyon dosyası.*

bilbao.sumo.cfg dosyasına bu değerleri girdikten sonra simülasyon ve network(ağ) dosyası hazır hale gelmiştir.

Not: Time ve inputlar ya da diğer özellikler başka haritalarda ve farklı araç sayılarında değişebilir. Bu kısmı göstermemizin sebebi, bu dosyayı yani simülasyon dosyasını Ns-2 dosyasına dönüştürecek olmamızdır.

2.3. Ns-2 Dosyasına Dönüştürme

Ns-2 dosyasına dönüştürmek için yapılması gereken iki basit işlem vardır**:**

1. **sumo -c bilbao.sumo.cfg --fcd-output trace.xml** komutunu yazmaktır.

   Bu komut bilbao.sumo.cfg dosyasını kullanarak trace.xml dosyasını oluşturur. Bu dosya ismi projede kullanılan mobility için böyle yazılmıştır (Sonuçta bilbao.sumo.cfg adında bir dosya var). Bu işlemden sonra trace.xml dosyası oluşur.

2. **python traceExporter.py -i trace.xml --ns2mobility-output=ns2mobility.tcl** komutunu yazmaktır.

   Bu komut ile ns2mobility.tcl dosyası oluşur ve bu dosya bizim için Ns-2 dosyamız olur.

Bu işlemlerden sonra artık Ns-3 ile Ns-2 dosyamızı çalıştırıp performans ölçebileceğiz.

3. Ns-3 Kodunu İnceleme

~/ns-allinone-3.41/ns-3.41/

```
#include "ns3/command-line.h"
#include "ns3/constant-position-mobility-model.h"
#include "ns3/end-device-lora-phy.h"
#include "ns3/end-device-lorawan-mac.h"
#include "ns3/gateway-lora-phy.h"
#include "ns3/gateway-lorawan-mac.h"
#include "ns3/log.h"
#include "ns3/lora-helper.h"
#include "ns3/mobility-helper.h"
#include "ns3/node-container.h"
#include "ns3/periodic-sender-helper.h"
#include "ns3/position-allocator.h"
#include "ns3/simulator.h"
#include "ns3/core-module.h"
#include "ns3/mobility-module.h"
#include "ns3/ns2-mobility-helper.h"
#include <algorithm>
#include <ctime>
#include <map>
#include <vector>

using namespace ns3;
using namespace lorawan;
```

Şekil 11: *Ns-3 kütüphanelerinin import edilmesi.*

Şekil 11'de gözüken kütüphaneleri import etmeniz gerekiyor çünkü bu kütüphaneler sayesinde Ns-3'ün LoRaWAN'ı çalışır.

```
NS_LOG_COMPONENT_DEFINE("SimpleLorawanNetworkExample");
```

Şekil 12: *Ns-3 loglarının kontrolü.*

Ns-3'teki logları kontrol etmek için kullanılır (Şekil 12).

```
std::vector<int> packetsSent(1, 0);
std::vector<int> packetsReceived(1, 0);
std::map<uint64_t, Time> transmissionTimes;
std::vector<Time> delays;
```

Şekil 13: *Global değişkenlerin tanımlanması.*

Bunları global değişken olarak tanımlamanız gerekiyor**. packetsSent** değişkeni gönderilen paket sayısını tutacak, yani toplam paket sayısını tutacak. **packetsReceived** ise ulaşan paket sayısını tutacak. **transmissionTimes** ise paketlerin gönderim zamanını tutar**. delays** ise gecikmeyi tutar. **transmissionTimes** değişkeninin kullanılmasının sebebi paket gecikmesini bulabilmek içindir.

```
void OnTransmissionCallback(Ptr<const Packet> packet, uint32_t senderNodeId)
{
    NS_LOG_FUNCTION(packet << senderNodeId);
    LoraTag tag;
    packet->PeekPacketTag(tag);

    packetsSent[0]++;
    uint64_t packetId = packet->GetUid();
    transmissionTimes[packetId] = Simulator::Now();
}
```

Şekil 14: *Paket gönderim fonksiyonu.*

Paket gönderirken bu fonksiyonu çağırarak hem paket gönderim sayısını güncelleyebilir hem de **transmissionTimes** ile süreyi başlatabilirsiniz. **packetID** benzersiz bir ID'ye sahip olduğu için paket karmaşıklığı olmayacak ve "Bu kimin paketi?" diye bir soru olmayacak.

```
void OnPacketReceptionCallback(Ptr<const Packet> packet, uint32_t receiverNodeId)
{
    NS_LOG_FUNCTION(packet << receiverNodeId);
    LoraTag tag;
    packet->PeekPacketTag(tag);

    packetsReceived[0]++;
    uint64_t packetId = packet->GetUid();
    if (transmissionTimes.find(packetId) != transmissionTimes.end())
    {
        Time transmissionTime = transmissionTimes[packetId];
        Time receptionTime = Simulator::Now();
        Time delay = receptionTime - transmissionTime;
        delays.push_back(delay);
        NS_LOG_INFO("Packet " << packetId << " delay: " << delay.GetSeconds() << " seconds
    }
}
```

Şekil 15: *Paket alım fonksiyonu.*

Burada, gönderilen paket ulaştığında ulaşılan paketi bir arttırır ve önceki fonksiyondaki gönderim zamanını içerir. **delay** ise ulaştığı zaman ile gönderim zamanı arasındaki farktır, bu yüzden ulaştığı zamanı simülasyondaki şimdiki zaman ile gönderim zamanının farkı olarak hesaplar.

```
CommandLine cmd;
int nDevices = 50;
std::string OutputFolder = "";
int nGateways = 1;
int appPeriodSeconds = 50;
```

Şekil 16: *CommandLine değişkenleri.*

Şekil 16'daki amaç, scriptten gelecek değerlere bakmak ve eğer değer yoksa bu değişkenleri bu şekilde çalıştırmaktır. **CommandLine cmd** ise terminaldeki değişkenleri aktarabilmek için kullanılmıştır.

```
// Create the channel
NS_LOG_INFO("Creating the channel...");

Ptr<LogDistancePropagationLossModel> loss = CreateObject<LogDistancePropagationLossModel>();
loss->SetPathLossExponent(3.76);
loss->SetReference(1, 7.7);

Ptr<PropagationDelayModel> delay = CreateObject<ConstantSpeedPropagationDelayModel>();

Ptr<LoraChannel> channel = CreateObject<LoraChannel>(loss, delay);
```

Şekil 17: *Kanal oluşturma*

Bu kısımda kanal oluşturuyoruz. Bu kodları LoRaWAN simple-network-example'dan alabilirsiniz. Kanal oluşturma kodu hazırdır.

```
// Create the helpers
NS_LOG_INFO("Setting up helpers...");

MobilityHelper mobility;
Ptr<ListPositionAllocator> allocator = CreateObject<ListPositionAllocator>();
allocator->Add(Vector(1000,0,0));
mobility.SetPositionAllocator(allocator);
mobility.SetMobilityModel("ns3::ConstantPositionMobilityModel");

LoraPhyHelper phyHelper = LoraPhyHelper();
phyHelper.SetChannel(channel);

LorawanMacHelper macHelper = LorawanMacHelper();

LoraHelper helper = LoraHelper();
```

Şekil 18: *Yardımcılar (Helpers) oluşturma.*

MobilityHelper, düğümlerin pozisyonlarını ve hareketlerini yönetir. Burada sabit bir pozisyon atanır. **phyHelper**, fiziksel katman yardımcısı, kanal ile ilişkilendirilir. **macHelper** ise MAC katman yardımcısıdır. **LoraHelper** ise LoRaWAN yardımcısıdır.

Ayrıca allocator add fonksiyonu Gateway konumunu değiştirmeye yarıyor. Sabit konumdan kasıt ise mobility dosyasının içeriğine dokunulmamasından bahsediliyor.

```
// Create end devices
NS_LOG_INFO("Creating the end devices...");

cmd.AddValue("nDevices", "Number of devices to simulate", nDevices);
cmd.AddValue("nGateways","Number of Gateways",nGateways);
cmd.AddValue("appPeriodSeconds","AppPeriodSeconds ",appPeriodSeconds);
// Diğer komut satırı argümanlarını ekleyin
cmd.AddValue("OutputFolder", "OutputFolder create", OutputFolder);
cmd.Parse(argc, argv);

std::string traceFile = "/home/yucel/ns-allinone-3.41/ns-3.41/scratch/ns2mobility.tcl";
```

Şekil 19: *Komut satırı değerlerinin kullanımı.*

Burada, komut satırındaki (terminaldeki) değerleri alıp kullanmamız için kod yazıldı. **traceFile** kısmına **ns2mobility.tcl** dosyasının bulunduğu konum yazıldı. (Bu konumu kendi çalışma ortamınıza göre değiştirin).

```
double simulationTimeSeconds = 2171.0;
Ns2MobilityHelper ns2 = Ns2MobilityHelper(traceFile);
NodeContainer endDevices;
endDevices.Create(nDevices);
ns2.Install();
phyHelper.SetDeviceType(LoraPhyHelper::ED);
macHelper.SetDeviceType(LorawanMacHelper::ED_A);
helper.Install(phyHelper, macHelper, endDevices);

// Create gateways
NS_LOG_INFO("Creating the gateway...");

NodeContainer gateways;
gateways.Create(nGateways);

mobility.Install(gateways);

phyHelper.SetDeviceType(LoraPhyHelper::GW);
macHelper.SetDeviceType(LorawanMacHelper::GW);
helper.Install(phyHelper, macHelper, gateways);
```

Şekil 20: *Simülasyon süresi ayarlama.*

Öncelikle simülasyon süresi ayarlandı. Bu süre **ns2mobility.tcl** dosyasına bağlıdır ve bizim yaptığımız simülasyon **2171** saniye sürüyor, bu yüzden bu değeri belirledik.

Şekil 20'den devam edecek olursak, **Ns2MobilityHelper** ise izleme dosyasını kullanarak düğümlerin hareketlilik modelini yükler. **NodeContainer** belirtilen sayıda uç cihaz oluşturmak için kullanılır **(nDevices)**. **endDevices.Create(nDevices)** ise **nDevices** sayısı kadar uç cihaz üretir. **ns2.Install()** ise hareketlilik modelini uç cihazlara yükler.

phyHelper.SetDeviceType(LoraPhyHelper::ED) fiziksel katmanını uç cihazlar için yapılandırır. **macHelper.SetDeviceType(LorawanMacHelper::ED_A)** MAC katmanını uç cihazlar için yapılandırır. **helper.Install(phyHelper, macHelper, endDevices)** ise fiziksel ve MAC katmanlarını uç cihazlara yükler.

NodeContainer gateways, ağ geçitlerini tutmak için bir konteyner oluşturur. **gateways.Create(nGateways)**, **nGateways** adedinde ağ geçidi oluşturur (1 tane tavsiye edilir çünkü birkaç tane olunca mobility saçmalamaya başlıyor). **mobility.Install(gateways)** ise ağ geçitlerine hareketlilik modeli uygular.

phyHelper.SetDeviceType(LoraPhyHelper::GW) fiziksel katmanı ağ geçitleri için yapılandırır. **macHelper.SetDeviceType(LorawanMacHelper::GW)** ise MAC katmanını ağ geçitleri için yapılandırır. **helper.Install(phyHelper, macHelper, gateways)** fiziksel ve MAC katmanlarını ağ geçitlerine yükler.

```
// Install applications in end devices

PeriodicSenderHelper appHelper = PeriodicSenderHelper();
appHelper.SetPeriod(Seconds(appPeriodSeconds));
ApplicationContainer appContainer = appHelper.Install(endDevices);

// Set Data Rates
std::vector<int> sfQuantity(1);
sfQuantity = LorawanMacHelper::SetSpreadingFactorsUp(endDevices, gateways, channel);
```

Şekil 21: *Uç cihazlara uygulama yükleme.*

PeriodicSenderHelper sınıfından bir nesne oluşturur. Bu sınıf, belirli aralıklarla veri gönderen bir uygulama yüklemeye yardımcı olur. **SetPeriod** fonksiyonu, uygulamanın ne sıklıkla veri göndereceğini ayarlar. Bu periyot, **appPeriodSeconds** değişkeni kullanılarak belirlenir. **Seconds(appPeriodSeconds)** ifadesi, periyodu saniye cinsinden ayarlar.

ApplicationContainer appContainer = appHelper.Install(endDevices) kodunda ise **Install** fonksiyonu, daha önce oluşturulan uç cihazlar (**endDevices)** üzerinde uygulamayı kurar. **ApplicationContainer**, yüklenen uygulamaların bir konteynerini tutar.

std::vector<int> sfQuantity(1) kodunda ise **sfQuantity**, yayılım faktörlerinin sayısını tutan bir vektördür ve burada başlangıç boyutu 1 olarak ayarlanmıştır.

sfQuantity = LorawanMacHelper::SetSpreadingFactorsUp(endDevices, gateways, channel) kodunda ise **SetSpreadingFactorsUp** fonksiyonu, uç cihazların ve ağ geçitlerinin yayılım faktörlerini belirlemek ve ayarlamak için kullanılır. Bu fonksiyon, yayılım faktörlerini uç cihazlar, ağ geçitleri ve kanal bilgileri kullanarak otomatik olarak belirler ve ayarlar. Bu fonksiyon, yayılım faktörlerinin ne sıklıkla kullanıldığını belirten bir vektör olarak döner.

```
// Install trace sources for packet transmission and reception
for (auto node = endDevices.Begin(); node != endDevices.End(); ++node)
{
    (*node)->GetDevice(0)->GetObject<LoraNetDevice>()->GetPhy()->TraceConnectWithoutContext(
        "StartSending",
        MakeCallback(OnTransmissionCallback));
}

for (auto node = gateways.Begin(); node != gateways.End(); ++node)
{
    (*node)->GetDevice(0)->GetObject<LoraNetDevice>()->GetPhy()->TraceConnectWithoutContext(
        "ReceivedPacket",
        MakeCallback(OnPacketReceptionCallback));
}
```

Şekil 22: *Paket gönderim ve alım fonksiyonlarını kullanma.*

Bu kod parçacığında, Şekil 14 ve 15'te tanımladığımız **OnTransmissionCallback** ve **OnPacketReceptionCallback** fonksiyonlarını kullanmak için gerekli olan kodlar yer almaktadır.

```
// Simulation
Simulator::Stop(Seconds(simulationTimeSeconds));
Simulator::Run();
Simulator::Destroy();

// Calculate packet delivery ratio
NS_LOG_INFO("Calculating performance metrics...");

float totalPacketsSent = packetsSent[0];
float totalPacketsReceived = packetsReceived[0];
float totalPacketsLost = totalPacketsSent - totalPacketsReceived;
float percantage = totalPacketsReceived / totalPacketsSent;
std::cout << "Veri Orani: "
          << "Gonderilen: " << totalPacketsSent << ", "
          << "Ulaşan: " << totalPacketsReceived << ", "
          << "Kaybolan: " << totalPacketsLost << ", "
          << "Basari orani: " << percantage << std::endl;
```

Şekil 23: *Simülasyon çalıştırma ve performans metriklerinin hesaplanması.*

Bu kod parçacığı, simülasyon çalıştırma komutlarını ve performans metriklerinden birinin (Paket Ulaşım Başarı Oranı) hesaplanmasını içerir.

```
// Simulation
Simulator::Stop(Seconds(simulationTimeSeconds));
Simulator::Run();
Simulator::Destroy();

// Calculate packet delivery ratio
NS_LOG_INFO("Calculating performance metrics...");

float totalPacketsSent = packetsSent[0];
float totalPacketsReceived = packetsReceived[0];
float totalPacketsLost = totalPacketsSent - totalPacketsReceived;
float percantage = totalPacketsReceived / totalPacketsSent;
std::cout << "Veri Orani: "
          << "Gonderilen: " << totalPacketsSent << ", "
          << "Ulaşan: " << totalPacketsReceived << ", "
          << "Kaybolan: " << totalPacketsLost << ", "
          << "Basari orani: " << percantage << std::endl;
```

Şekil 24: *Paketlerin ortalama gecikme süresi.*

Bu kod parçacığı, paketlerin ortalama gecikme süresini ve ulaşan her paketin gecikme süresini yazdırmak için kullanılır.

4. Script İnceleme

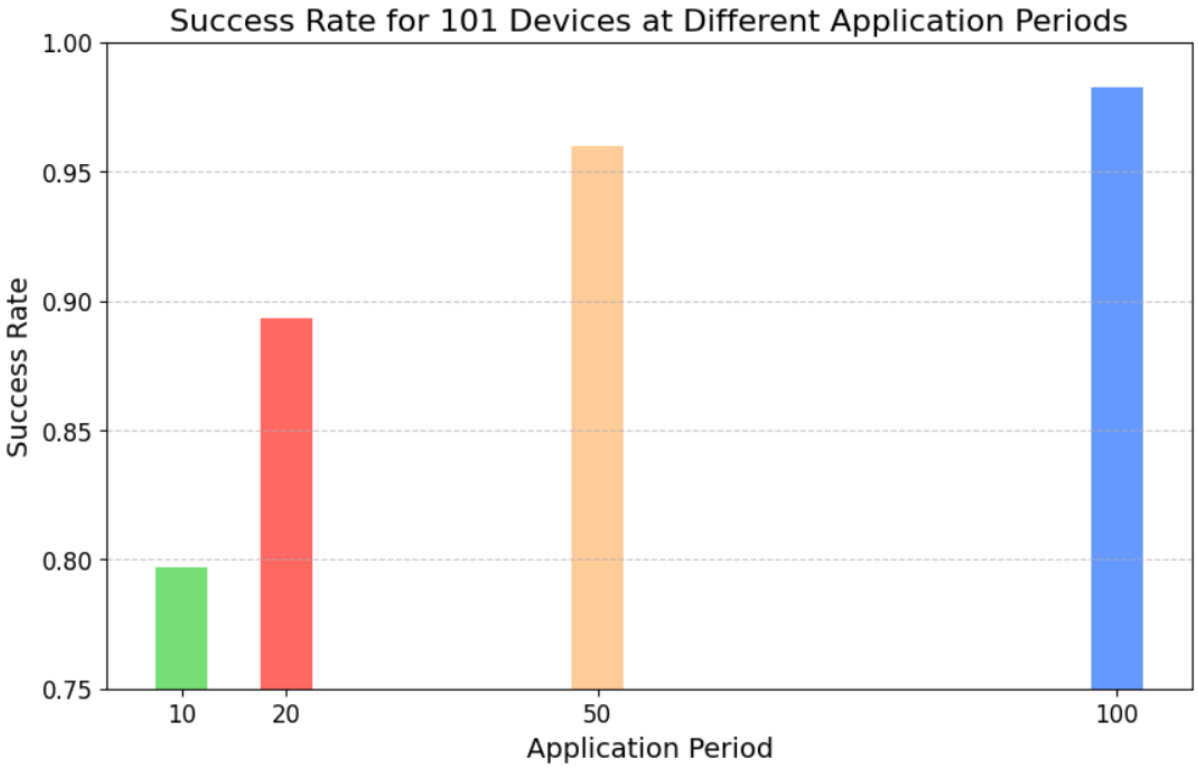
```
script="scratch/lastchance"

nDevices=50    # Change from 50 to 200
nGateways=1
appPeriodSeconds=20


folder="scratch/experiments/Test_${nDevices}_${nGateways}_${appPeriodSeconds}/run"


echo -n "Running experiments: "

for r in `seq 1 30`;

do

  echo -n " $r"

  mkdir -p $folder${r}

        ./ns3 run "$script --RngRun=$r --nDevices=$nDevices --nGateways=$nGateways --appPeriodSeconds=$appPeriodSeconds --OutputFolder=${folder}${r}" > "$folder${r}/log.txt" 2>&1

done

echo " END"
```

Şekil 25: *Script kodu.*

Bu kod parçacığında, çalıştırmak için gerekli script kodumuzu gösteriyor. Bu kodu çalıştırmak için **ns-3.41** klasörünün içinde olmanız gerekiyor. **Ns-3.41** klasörünün içinde **scratch** klasörü var ve onun içine **lastchance.cc** dosyasını atmanız gerekiyor. Önceden de bahsettiğimiz gibi, **tracefile** yani **ns2mobility.tcl** dosyanızın konumunu **lastchance.cc** dosyasından değiştirmeniz gerekiyor.

Bu kodda 50 tane uç cihaz, bir tane gateway (ağ geçidi) ve her 20 saniyede bir paket gönderme yapılmıştır (her bir uç cihaz için). Bu kodun amacı aynı dosyayı 30 kere çalıştırmak ve böylelikle 30 kez çalıştıktan sonra her birinin başarı oranını alıp ortalamasını hesaplayarak istatistiksel sonuç elde etmektir. Aynı şeyi ortalama **delay** (gecikme) süresi için de yapılması gerekiyor.

5. Çıktılar (Sonuçlar)



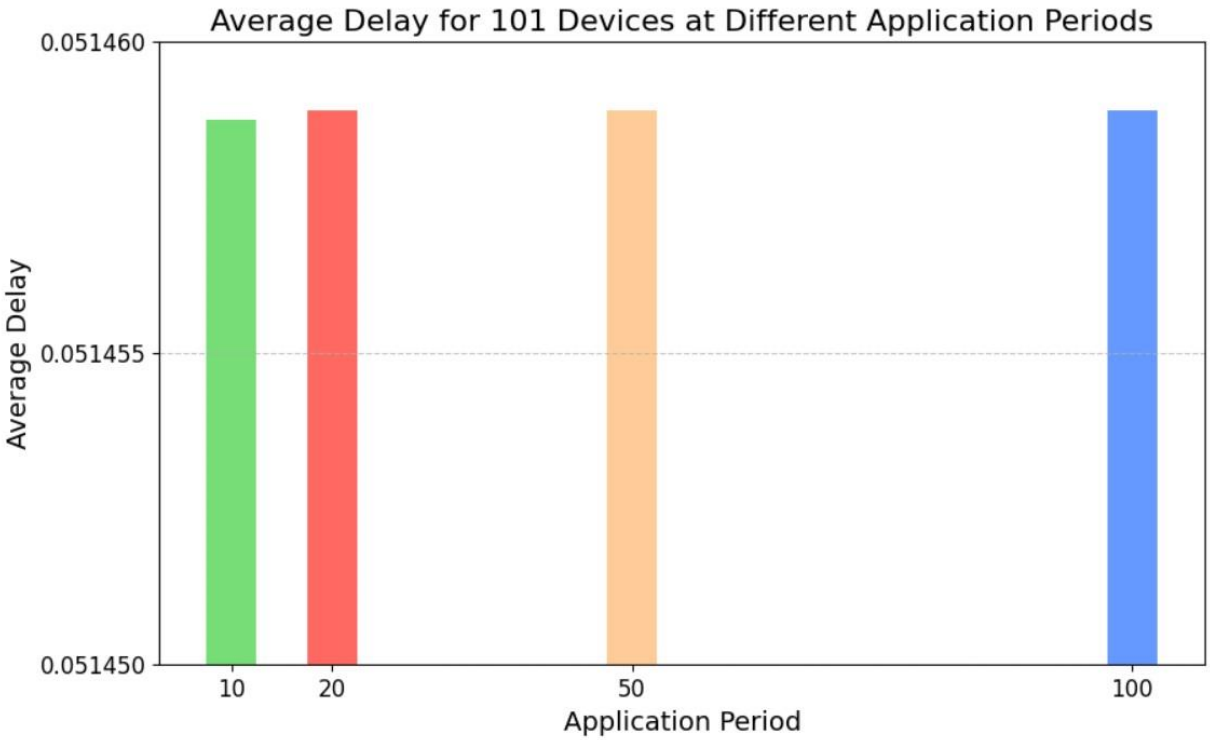
Şekil 26: *Farklı uygulama periyotlarında 101 cihaz için başarı oranı.*

Bu şekilde, **nDevice** değeri 101 ve **nGateway =1** olarak sabit tutuldu ve **allocator->Add(Vector(1000,0,0))** konumu şeklinde ayarlandı.

Konum yakın olduğu için paket başarı oranı yüksek çıkmıştır. **Application Period** (uygulama periyodu) arttıkça daha az paket gönderileceği için başarı oranı artar. Toplam gönderilen paket sayısını şu şekilde hesaplayabilirsiniz: $\frac{simülasyon\ süresi * uç\ cihaz\ sayısı}{Application\ Period}$

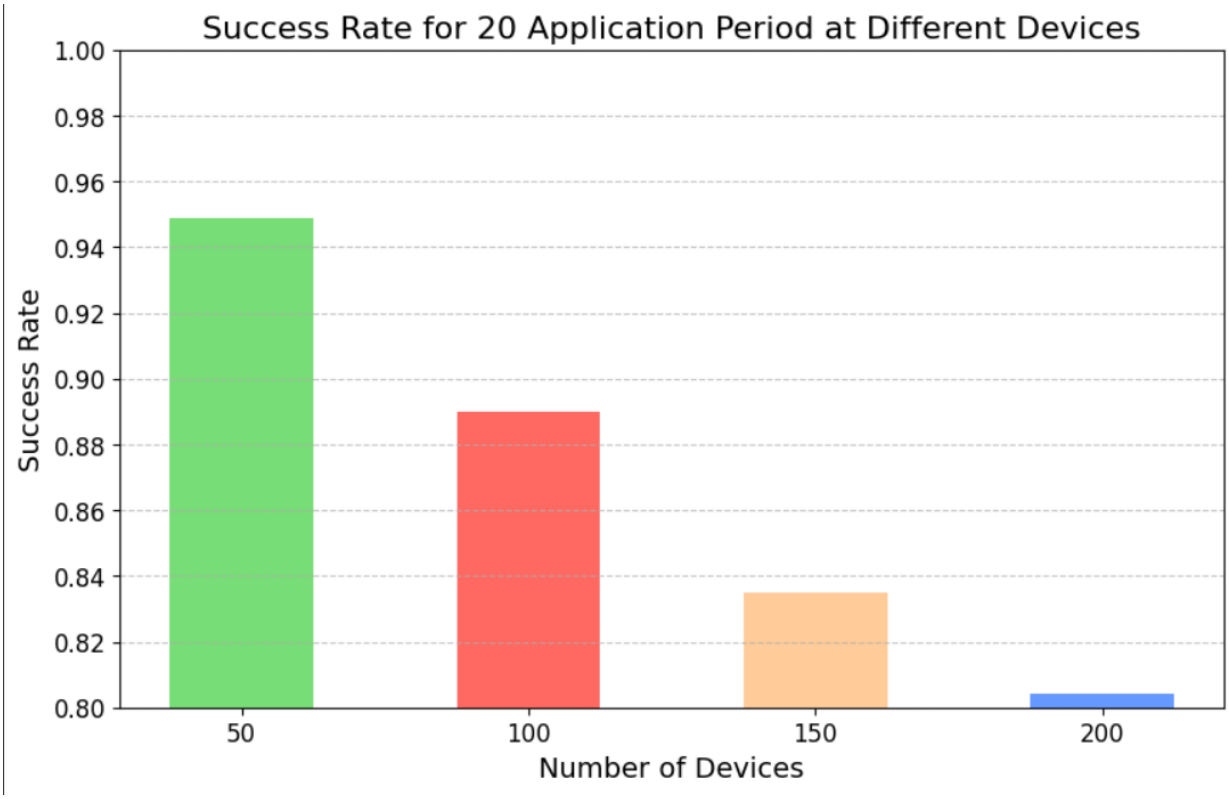
Yani**, Application Period** arttıkça ve diğer değişkenler sabit kalırsa gönderilen paket sayısı düşer ve başarı oranı daha da artar. (Sonuçta 20000 paketi teslim edebilmek mi daha kolay, yoksa 2000 paketi teslim edebilmek mi?)

Not: Burada ulaşılan paket sayısını tartışmıyoruz. Başarı oranı konusunda tartışıyoruz.



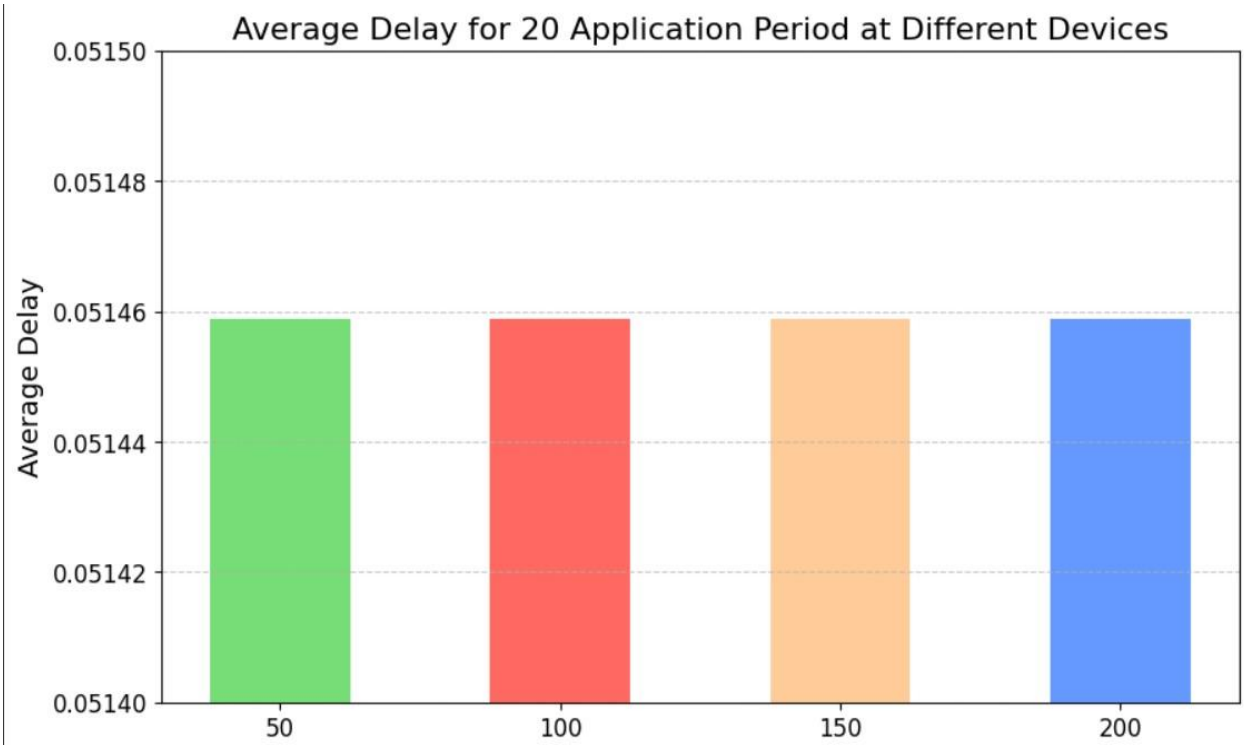
Şekil 27: *Farklı uygulama periyotlarında 101 cihaz için ortalama gecikme süresi.*

Ortalama paket gecikme süreleri ise yaklaşık olarak aynı çıkmıştır. Bu değerler konumla alakalıdır. İlerleyen sayfalarda konum değiştiğinde **average delay**'in (ortalama gecikme süresinin) değiştiğini göreceksiniz.



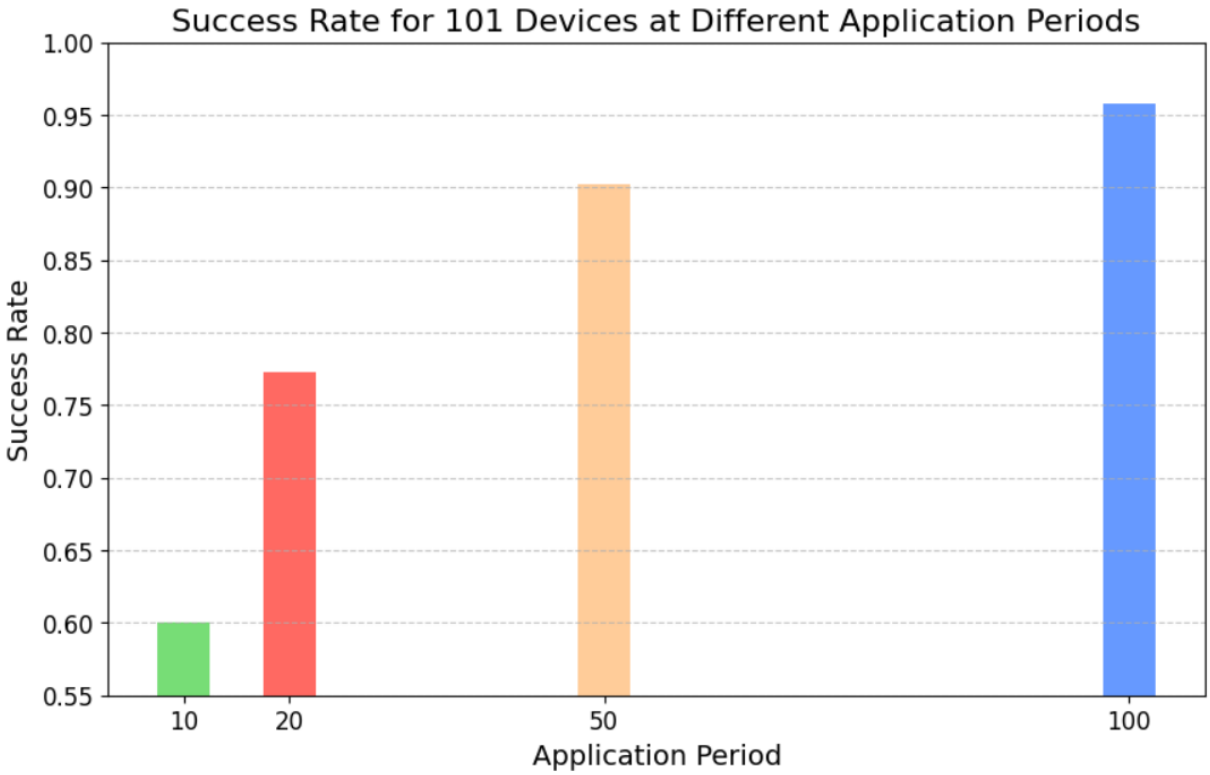
Şekil 28: *20 Uygulama periyodunda farklı cihaz sayılarında başarı oranı.*

Buradaki olay, Application Period sabit bırakılıp **cihaz sayısının(number of Devices)** değiştirilmesi durumundaki başarı oranını göstermektedir. Uç cihaz sayısı arttıkça başarı oranı düşüyor. Sebebi ise uç cihaz arttıkça toplam gönderilen paket sayısının da artmasıdır. (Konum olarak şu alınmıştır: **allocator->Add(Vector(1000,0,0))**)



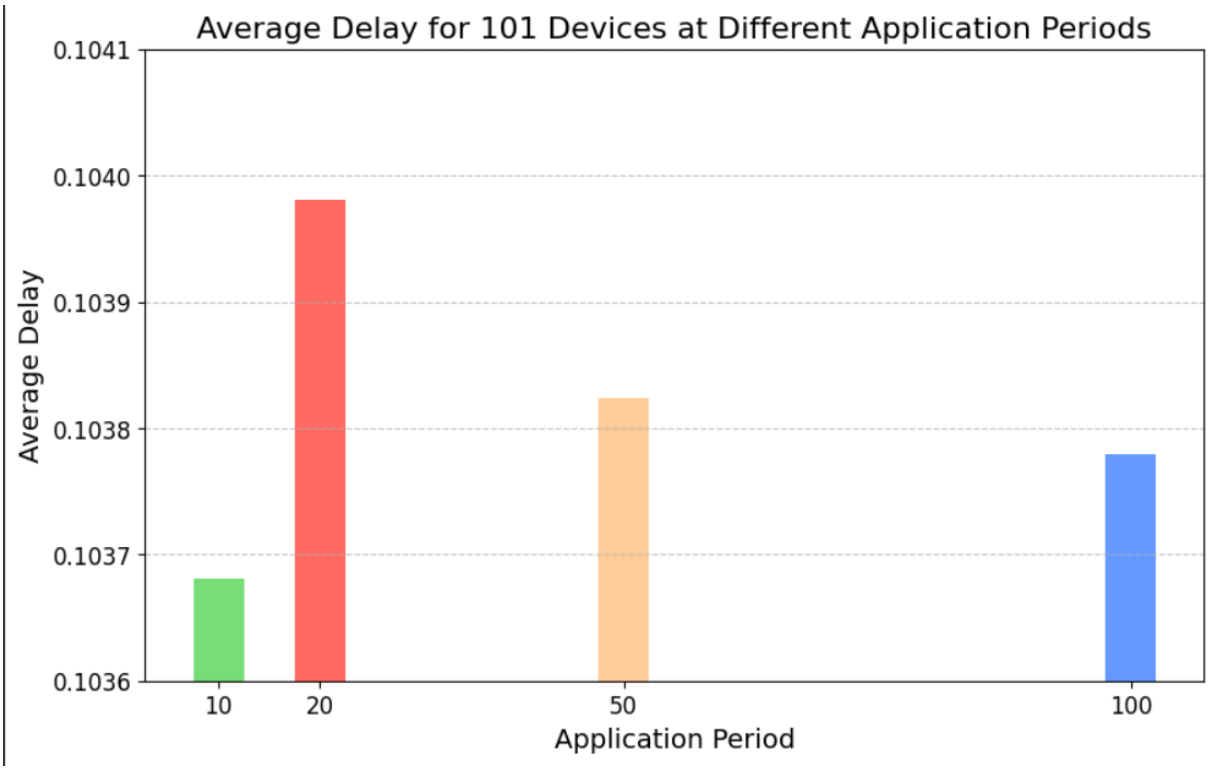
Şekil 29: *20 Uygulama periyodunda farklı cihaz sayılarında ortalama gecikme süresi.*

Burada da delay'ler hep aynı çıkıyor çünkü önceden bahsedildiği gibi gecikme süresini (**delay**) en çok etkileyen parametre konum değişikliğidir.



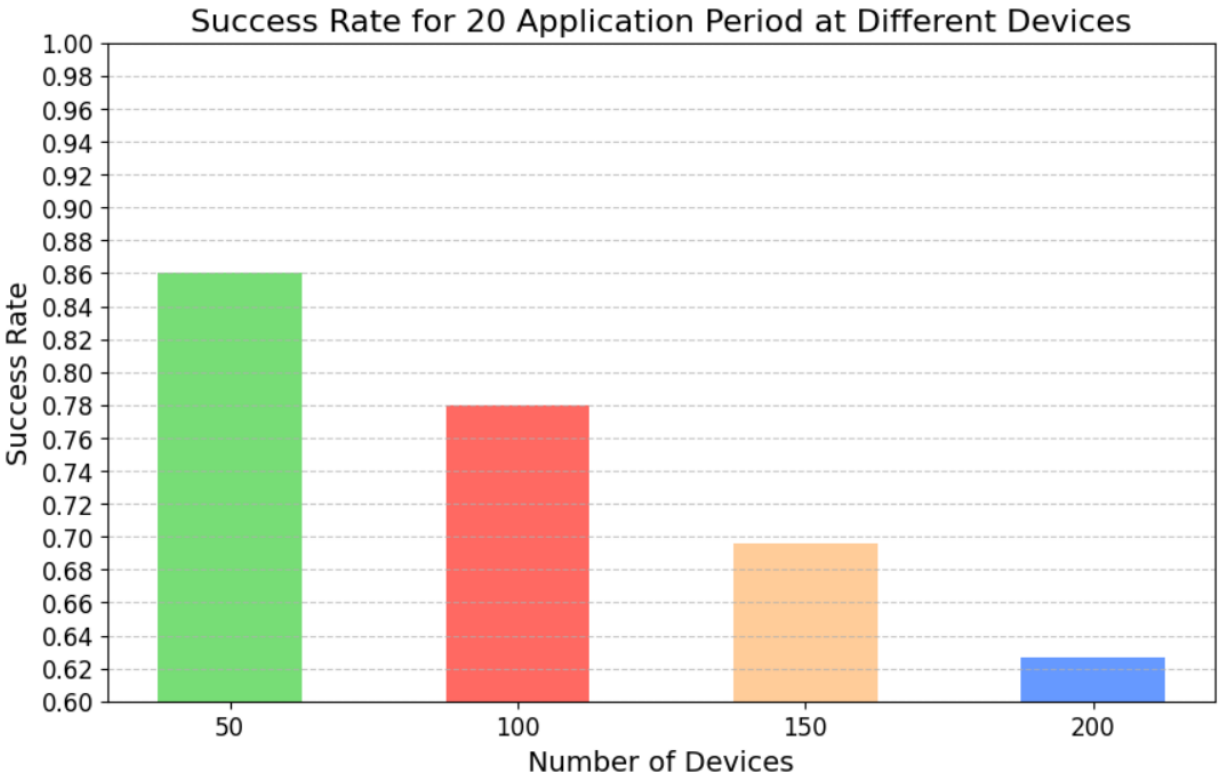
Şekil 30: *101 cihaz için farklı uygulama periyotlarında başarı oranı.*

Burada ise **nDevice = 101** ve **gateway = 1** iken bu değişikliğin olmasının sebebi konum değişikliğinden dolayıdır **(allocator->Add(Vector(2500,2500,2500)))**. Konum (Gateway konumu) uzaklaştığı için toplam gönderilen paket sayısı sabit kalsa bile ulaşılan paket sayısında düşüş yaşanmıştır. Bunun sebebi ise konumun uzaklaşmasından dolayıdır. Ne kadar uzaklaşırsa o kadar düşer, hatta başarı oranı 0'a bile düşebilir.



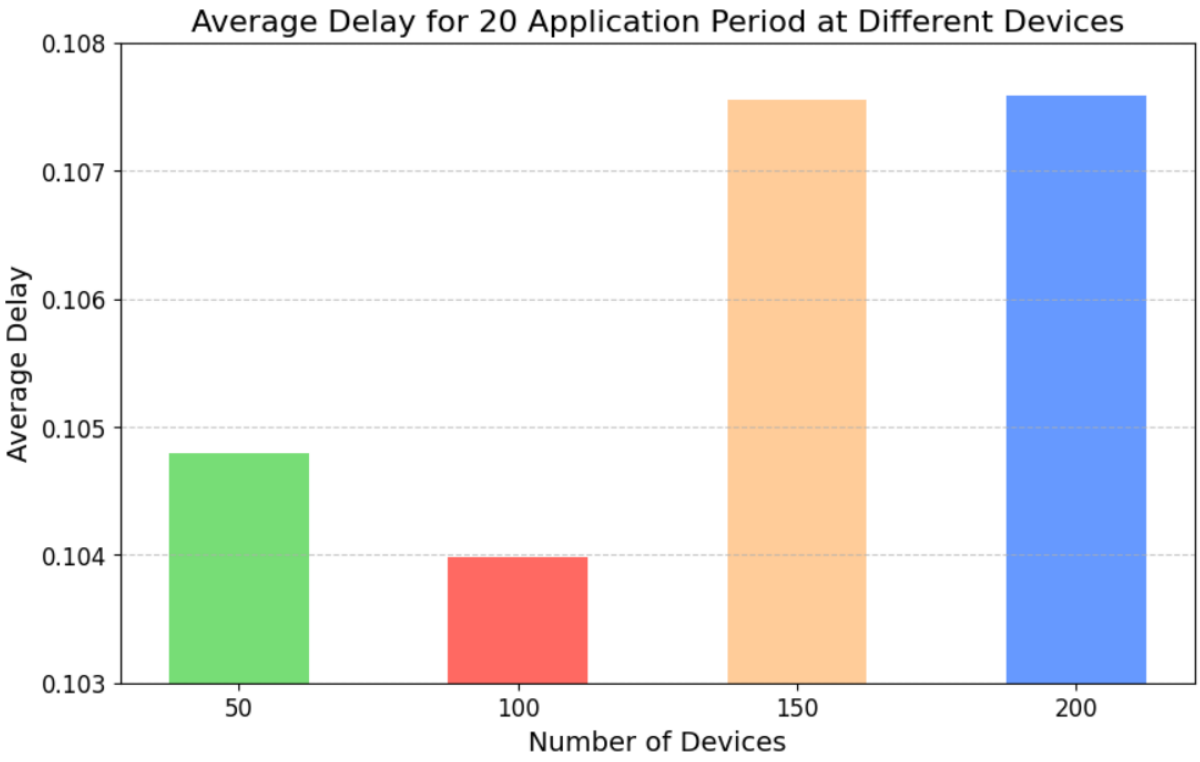
Şekil 31: *101 cihaz için farklı uygulama periyotlarında ortalama gecikme süresi.*

Delay süreleri konum uzaklaştığı için önceki sonuçtan (Şekil 27) daha farklı çıkmıştır. Delay süresi mobility konumu değiştikçe ve uzaklaştıkça artar ve sonunda 0 saniye olur. (0 saniye olmasının sebebi, paket ulaşamadığı için.)



***Şekil 32**: 20 Uygulama periyodunda farklı cihaz sayılarında başarı oranı(2).*

Buradaki konum **(allocator->Add(Vector(2500,2500,2500))** olduğundan başarı oranı düşmüştür. Ayrıca, uç cihaz sayısı arttıkça paket ulaşma başarı oranı düşmüştür. Çünkü toplam gönderilen paket sayısı artmıştır.



Şekil 33: *20 Uygulama periyodunda farklı cihaz sayılarında ortalama gecikme süresi(2).*

Sonuçlar arasında en dikkat çekici olay bu grafikte(sonuçta) yatıyor. Normalde uç cihaz sayısı arttıkça delay oranının düşmesi beklenirken (sonuçta daha fazla cihaz var), kırılma noktası yaşanıyor. 100 cihaz iken 0.104 saniyeye düşerken, 150 cihaz olduğunda ise 0.108'e yaklaşmış. Ayrıca 150'den 200'e çıktığında ise ufak bir artma meydana gelmiştir.

Çıkarım olarak şunu yaptık: "Maddelerin doygunluk hali gibi bu mobility'nin de doygunluk hali var ve o hali aşınca delay süresi çok yükseliyor, sonra da ufak ufak bir değişime uğruyor."

Not: Bu grafiklerdeki her bir değeri elde etmek için kod 30 kez çalıştırılıp ortalaması alınmıştır.

6. Özet

Bu çalışmada, LoRaWAN kablosuz iletişim teknolojisinin hareketli ortamlarda performansının nasıl etkilendiği incelenmiştir. Günümüzde IoT uygulamalarının yaygınlaşmasıyla birlikte, bu tür teknolojilerin performans değerlendirmesi büyük önem taşımaktadır. Çalışma kapsamında, Ns-3 simülatörü kullanılarak ve gerçek dünya verilerini yansıtan OpenStreetMap tabanlı SUMO simülasyonlarıyla çeşitli senaryolar oluşturulmuştur.

Temel Bulgular:

- **Paket Başarı Oranı:** Uç cihaz sayısı arttıkça toplam gönderilen paket sayısı da artmakta, bu da paket başarı oranını düşürmektedir. AppPeriod düştükçe de aynı senaryo gerçekleşmektedir. Ayrıca, konum mesafesi arttıkça başarı oranı düşmektedir çünkü cihazlar ve ağ geçidi arasında mesafe artmaktadır.

- **Gecikme Süreleri:** Cihazların konumu uzaklaştıkça gecikme süreleri artmakta ve bazı durumlarda iletişim tamamen kesilmektedir. Kesilmesinin sebebi, hiçbir paketin alıcıya ulaşmaması durumunda delay süresinin 0 olmasıdır.

- **Konumun Etkisi:** Cihazlar ve ağ geçidi arasındaki mesafe arttıkça, iletilen paket sayısında belirgin bir düşüş gözlenmiştir.

- **Hareketliliğin Etkisi:** Yüksek hızlarda hareket eden cihazların paket başarı oranı ve gecikme süreleri önemli ölçüde değişiklik göstermiştir.

Çalışma, LoRaWAN ağlarının stratejik cihaz yerleşimi ve hareketlilik optimizasyonu ile performansının artırılabileceğin göstermektedir.

7. Kaynaklar

1. URL: https://github.com/signetlabdei/lorawan/tree/develop/examples
2. URL: https://youtu.be/wZycufsTEGU?si=IrDSkf_UrTQALuxP
3. URL: https://youtu.be/9Tt5GNMuOpc?si=ztPzcfmn4v6kmKhI
4. URL: https://chat.openai.com/

Github Linki(Kaynak Kodu):
https://github.com/Aytacus/Ns-3_Sumo_Application.git