İSTANBUL TEKNİK ÜNİVERSİTESİ
ELEKTRİK-ELEKTRONİK FAKÜLTESİ

GÜVENLİ BİR NESNELERİN İNTERNETİ AĞININ MİKRODENETLEYİCİLER KULLANILARAK GERÇEKLENMESİ

LİSANS BİTİRME TASARIM PROJESİ

Büşra ÖZEN
(040130020)

Furkan KURT
(040120385)

Proje Danışmanı: Doç. Dr. Sıddıka Berna Örs Yalçın

ELEKTRONİK VE HABERLEŞME MÜHENDİSLİĞİ BÖLÜMÜ

MAYIS, 2018

İTÜ, Elektronik ve Haberleşme Mühendisliği Bölümü'nün ilgili Bitirme Tasarım Projesi yönergesine uygun olarak tamamen kendi çalışmamız sonucu hazırladığımız "Güvenli Bir Nesnelerin İnterneti Ağının Mikrodenetleyiciler Kullanılarak Gerçeklenmesi" başlıklı Bitirme Tasarım Projesi'ni sunmaktayız. Bu çalışmayı intihal olmaksızın hazırladığımızı taahhüt eder; intihal olması durumunda bitirme tasarım projesinin başarısız sayılacağını kabul ederiz.

Büşra ÖZEN .....
(040130020)

Furkan KURT .....
(040120385)

ÖNSÖZ

Engin bilgi birikimi ile bu projenin ortaya çıkmasını sağlayan ve bu süreçte her zaman bizlere yol gösteren değerli hocamız Doç. Dr. Sıddıka Berna Örs Yalçın'a sonsuz saygı ve teşekkürlerimizi sunuyoruz.

Projenin gidişatında karşılaştığımız sorunlarda bizden yardımlarını esirgemeyen değerli hocamız Doç. Dr. Güneş Karabulut Kurt'a ayrıca teşekkürlerimizi sunuyoruz.

Ve son olarak yaşamımızın her anında yanımızda olan ve mezuniyetimizde en büyük destekçilerimiz olan ailelerimize teşekkürlerimizi sunuyoruz.

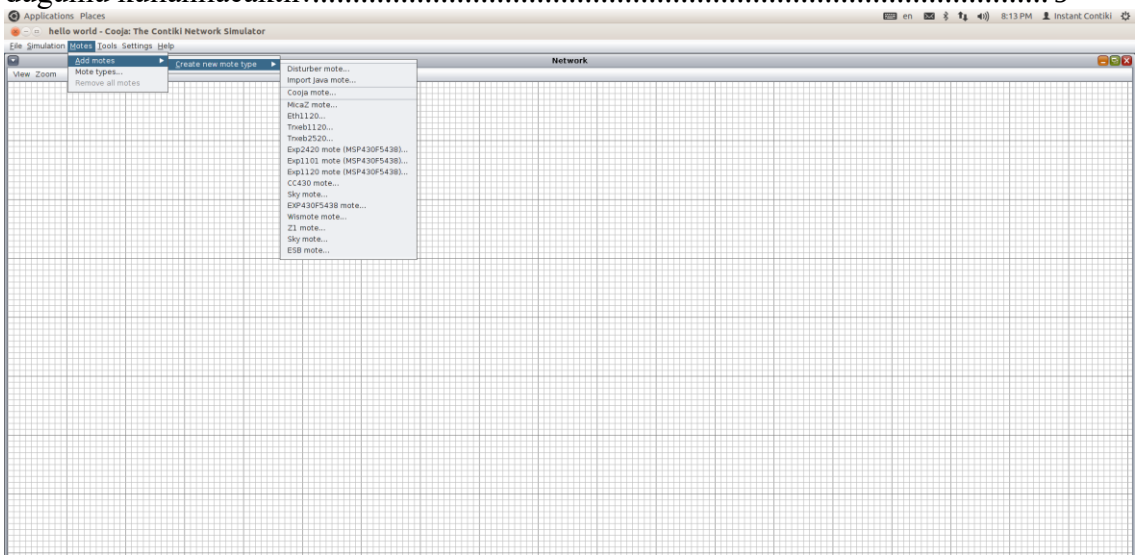
Mayıs 2018

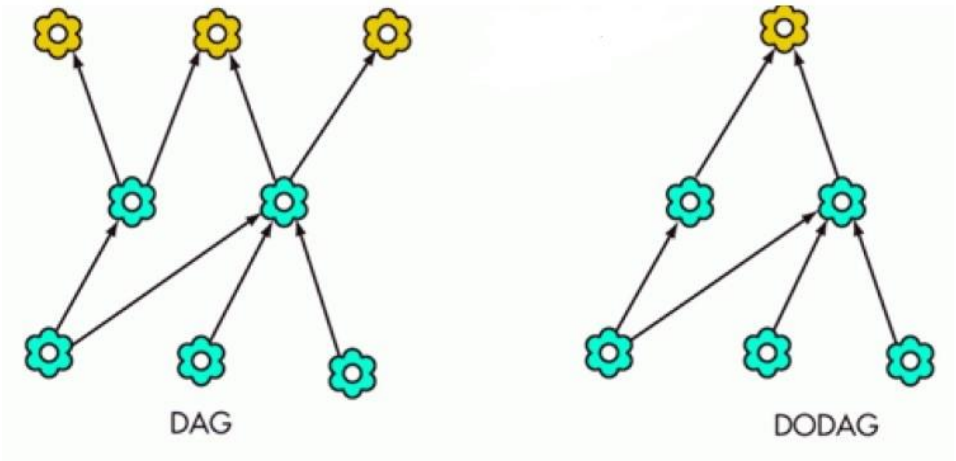
Büşra ÖZEN
Furkan KURT

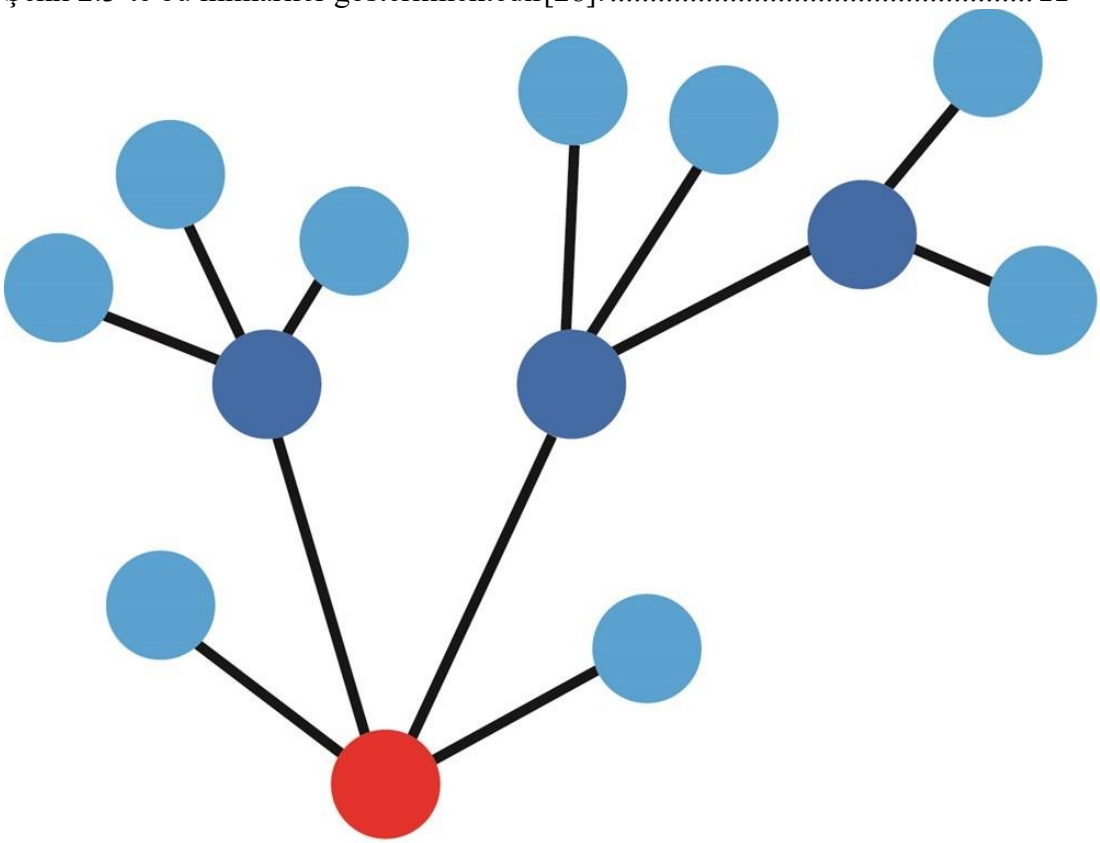
Proje Danışmanı : Doç. Dr. Sıddıka Berna Örs Yalçın .....

İÇİNDEKİLER

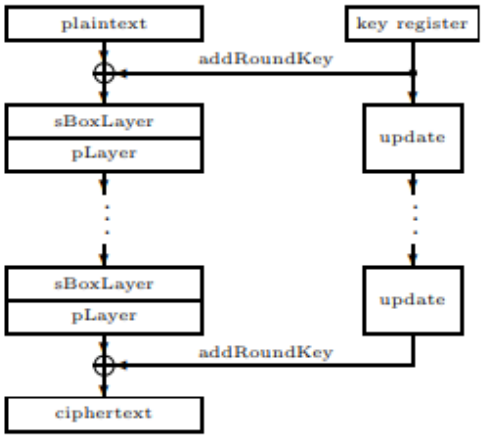
Sayfa



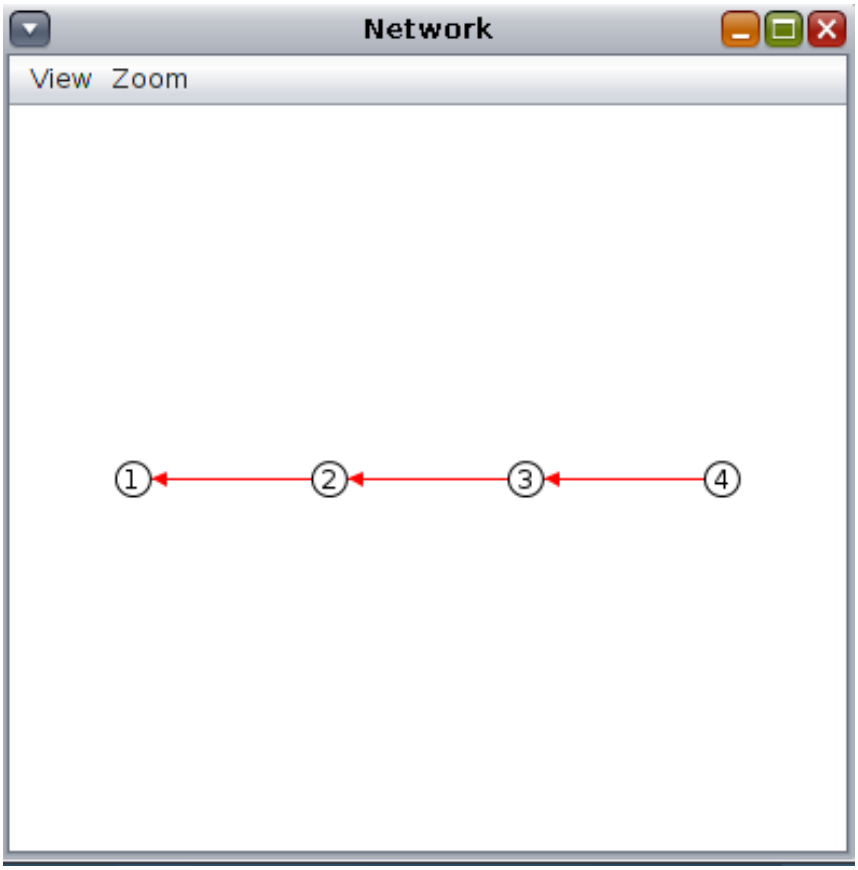




generateRoundKeys()
for $i = 1$ to 31 **do**
  addRoundKey(STATE,K_i)
  sBoxLayer(STATE)
  pLayer(STATE)
end for
addRoundKey(STATE,K_{32})



$$Z = H^{\mathrm{Nyb}}(K_Z, Y) = K_Z \odot \prod_{i=1}^{m} \alpha_r(y_i)$$
$$= K_Z \prod_{i=1}^{m} \alpha_r(h(x_i))$$

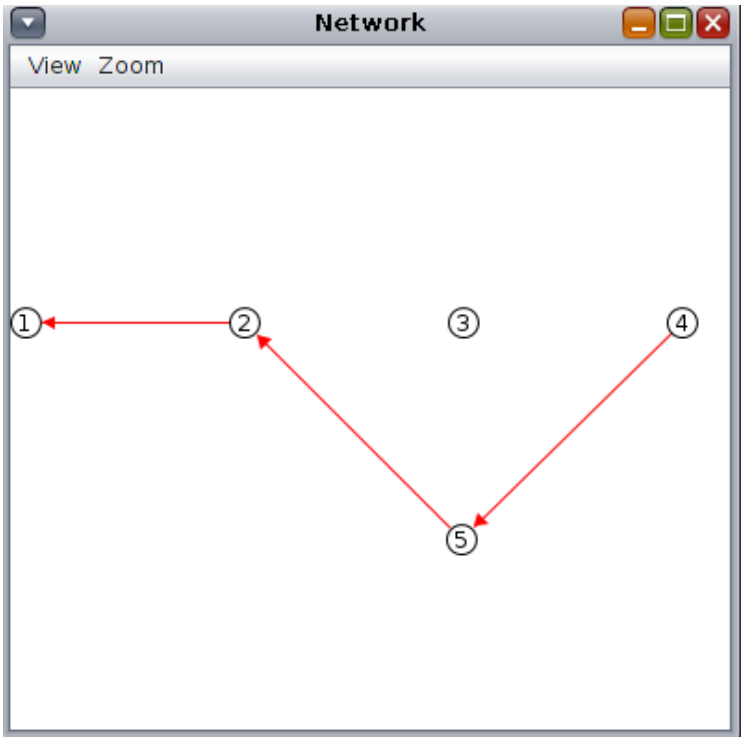



```
00:06.450   ID:2   Data sending...
00:06.450   ID:2   #L 1 1; red
00:06.454   ID:1   Data received!
00:06.454   ID:1   Data received from: 2
00:06.454   ID:1   Received data size: 8
00:06.454   ID:1    OK
```


Şekil 4.2'de 2 numaralı düğümün mesaj gönderme anındaki Cooja çıktısı verilmiştir. Haberleşmede önce 8 baytlık bir veri şifreleme ve kimlik doğrulama olmadan gönderilmiştir. 00:06.450 anında gönderme işlemi başlamış ve 00:06.454 anında 1 numaralı yönlendiriciye ulaşmıştır. Haberleşmenin toplamda 4 ms sürdüğü gözlemlenmiştir.

Yönlendirci diğer düğümlerin iletişim menzilinde olmadığından, haberleşmek için düğümler mesajları birbirleri üzerinden atlatarak göndermelidir. Her düğümün haberleşmesi için gereken süreler ölçülmüş ve ölçülen değerler Şekil 4.3'te verilmiştir.

Şekil 4.4'te 2 numaralı düğümün mesaj gönderme anındaki Cooja çıktısı verilmiştir. Haberleşmede önce 8 baytlık veri, 8 bayt kimlik doğrulama koduyla birlikte şifrelenmiş şekilde gönderilmiştir. 01:11.540 anında gönderme işlemi başlamış ve 01:11.544 anında 1 numaralı yönlendiriciye ulaşmış ve mesaj doğrulanmıştır. Haberleşmenin toplamda 4 ms sürdüğü gözlemlenmiştir.



KISALTMALAR

AES **:** Advenced Encrypion Standard
ALU : Arithmetic Logic Unit
DAG **:** Directed Acyclic Graphs
DODAG **:** Destination Oriented Directed Acyclic Graphs
IoT **:** Internet of Things
IPv4 **:** Internet Protocol version 4
IPv6 **:** Internet Protocol version 6
LED **:** Light Emitting Diode
LLN **:** Low-power and Lossy Network
MAC **:** Media Access Control
MP2P : Multi-Point-To-Point
P2MP : Point-To-Multi-Point
P2P : Point-To-Point
RAM **:** Random Access Memory
RFID **:** Radio Frequency Identification
ROM **:** Read Only Memory
RPL **:** IPv6 Routing Protocol for Low-Power and Lossy Networks
SP : Substitution-Permutation
UDP **:** User Datagram Protocol
WSN **:** Wireless Sensor Network
XOR **:** Exclusive OR

ŞEKİL LİSTESİ

Sayfa

GÜVENLİ BİR NESNELERİN İNTERNETİ AĞININ MİKRODENETLEYİCİLER KULLANILARAK GERÇEKLENMESİ

ÖZET

İzlenebilen, algılanabilen ve kontrol edilebilen duyarga, araç ve cihaz gibi nesnelerin, internete bağlanarak bir ağ oluşturması olarak tanımlanabilen Nesnelerin İnterneti son yıllarda oldukça yaygın bir kullanım alanına sahiptir. Akıllı ev uygulamalarından araç takip sistemlerine, yaygın sağlık uygulamalarından sürücüsüz araçlara kadar birçok alanda yaşam kalitesini arttırma, enerjiyi verimli kullanma, olası problemleri önceden tespit edip önleme gibi pek çok fayda sağlayan Nesnelerin İnterneti henüz gelişmekte olduğu için gizlilik ve güvenilirlik gibi çözülmeyi bekleyen çok ciddi sorunlar vardır.

Nesnelerin İnterneti ağların denetimi ve mesajların gizliliği konularında henüz standartlaşmış bir önleme sahip değildir, dolayısıyla mevcut uygulamalarda veri ve kimlik bilgilerinin çalınması gibi mahremiyeti kötü etkileyen sonuçlar ortaya çıkabilmektedir. Oluşturulan ağa istenmeyen bir cihazın katılabilmesi, ağı meşgul ederek sistemin çalışmasını önleyebilmesi ya da paylaşılan bilgilere ulaşıp yorumlayarak sistem hakkında bilgi sahibi olması bu uygulamalarda karşılaşılabilecek sorunlardandır. Bu bağlamda süregelen araştırmalar olsa da henüz standartlaştırılmış bir çözüm bulunmamaktadır ve mutlak güvenlik gereksinimi olan alanlarda Nesnelerin İnterneti kullanımı yaygınlaşamamaktadır.

Bu projenin amacı, veri gizliliği ve bütünlüğü ile ağ güvenliği konularında eksiklikleri giderilmiş güvenli bir nesnelerin interneti ağı tasarlamaktır. Bu amaçla yapılan literatür araştırmaları sonucu veri gizliliği ve bütünlüğü için PRESENT şifreleme algoritması; ağ güvenliği için Nyberg Çoklu Gönderim Kimlik Doğrulama algoritması kullanılarak Cooja benzetim ortamında Düşük Güç ve Kayıplı Ağlar için IPv6 Yönlendirme Protokolü üzerine şekillenen güvenli bir Nesnelerin İnterneti ağı kurulmuştur.

Ağ kurulumu sırasında öncelikle düğümlerin lineer bir şekilde sıralanıp paketlerin bu yolla yönlendiriciye iletildiği bir ağ modeli gerçeklenmiş ve herhangi bir şifreleme/kimlik doğrulama algoritması kullanılmadan düğümlerin birbirleri üzerinden yönlendiriciye veri göndermeleri sağlanmıştır. Bu esnada düğüm sayıları gittikçe arttırılarak her düğüm sayısı için bir gönderim süresi hesaplanmış, sonuçlar listelenmiştir. Daha sonra ise sisteme şifreleme ve kimlik doğrulama fonksiyonları uygulanarak aynı ağ tekrar kurulmuş ve yine düğüm sayısı arttırılarak her düğüm için gönderim süresi ölçülmüş ve listelenmiştir.

Bu modelden sonra ise güvenli veri gönderiminin sisteme getirdiği enerji yükünü tahmin edebilmek amacıyla bir pil tahmin algoritması yazılmışır. Buna göre düğümlerin işlevlerine göre bir birim güç harcama oranı belirlenmiş ve bu orana göre her düğümün çalıştırdığı fonksiyona göre harcadığı gücü tutabilme özelliği kazandırılmıştır. Daha sonra bir sınır değer belirlenmiş ve gücü bu sınır değere gerileyen düğümlerin kendilerini diğer düğümlerin veri iletimine kapatacağı bir mekanizma oluşturulmuştur. Bu doğrultuda ata düğümü ağdan düşen düğümlerin

alternatif bir rotaya yönelebileceği basit bir ağ oluşturulmuş ve pil tahmin mekanizmasının doğruluğu test edilmiştir.

Kablosuz duyarga cihazlarının kısıtlı bir hafıza alanına ve düşük bir enerji kapasitesine sahip olmaları; kablosuz duyarga ağlarının ise gizlilik ve güvenlik endişeleri barındırmaları Nesnelerin İnterneti temelli uygulamaların en önemli problemlemleridir. Bu projede veri gizliliği ve ağ güvenliği olabildiğince hafif şifreleme ve kimlik doğrulama algoritmalarıyla sağlandığı için Nesnelerin İnterneti'nin yaygınlaşmasının önündeki en büyük engeller aşılmıştır. Dolayısıyla bu proje, internete bağlanabilen cihazların var olduğu herhangi bir haberleşme senaryosunda kendine yer bulabilecektir.

IMPLEMENTATION OF A SECURE INTERNET OF THINGS NETWORK BY USING MICROCONTROLLERS

SUMMARY

Internet of Things can be defined as a network of objects such as sensors and devices that can be monitored, detected and controlled and has a very common usage area in recent years. IoT networks increase the quality of life in many areas, use energy efficiently and identify and prevent potential problems by used in a wide range of application including smart homes, vehicle tracking systems, driverless otomobiles and many health apllications. Nevertheless, there are some problems that are very serious and still waiting to be solved, such as privacy and reliability.

IoT does not have any standardized measure of message privacy and network control yet, therefore in the current applications, the results that can be effect the privacy in a bad way like stolen of data and identification informations could be happen. For instance a device can be connect the network without any permission, damage the system by keeping busy the network or have a knowledge about the system by reaching the shared datas. In this context, there are several research about this topic but there is no standard resolution; hence IoT usage cannot have a widespread applications in the areas that need a certain security.

In this project, designing a secure Internet of Things network without any deficiencies about network security and data privacy and integrity is the main purpose. For this purpose, first of all, a literature survey was conducted. That survey's first aim is the finding the most lightweight and effective solutions for privacy and security of wireless sensor networks. In this direction, PRESENT Block Cipher algorithm for data privacy and integrity and Nyberg Multicast Authentication algorithm for network security are chosen.

After the algorithms to be used are decided, the implementations of them were done in C programming language. Then these implementations were runned in Contiki operating system and Cooja simulation environment.

Contiki is an operating system that used commonly in constrained systems like wireless communication networks that include embedded devices. It enables a sufficient and lightweight operating system to the users. And Cooja is a simulation environment in Contiki system and provides to set up a network and simulate it in a way that is so much close to the actual conditions for constrained devices.

During the network setup, first of all a network model was implemented in which the nodes were sorted in a linear model and the packets were forwarded to the router in this way and nodes were sent data to each other through the routers without using any encryption/authentication algorithm. At this time, the number of nodes is gradually increased and a transmission period is calculated for each node count, then the results are listed. After these tests, the same network is reestablished by applying encryption and authentication functions to the system. Then the number of nodes is increased and the transmission time for each node is measured and listed again.

After this model, a battery estimation algorithm was written in order to estimate the power consumption that the secure data transmission brings to the system. According to this, an unit power consumption ratio was determined for each function of nodes. In this context, each node is given the ability to hold the power information that it spends according to the function it operates. Then a limit value was determined and a mechanism was implemented in which the nodes whose power limit was reduced to this limit themselves to the transmission of data by the other nodes. In this direction, a simple network was created in which the nodes that whose parent nodes fall from the network can turn to an alternate route. In this way, the correctness of the battery estimation mechanism was tested.

Wireless sensor devices having a limited memory space and a low energy capacity and wireless sensor networks containing privacy and security concerns are the most important problems of Internet of Things based applications. In this project, the biggest obstacle to the spreading of the Internet of Things has been overcome since data privacy and network security are provided with the most lightweight encryption and authentication algorithms as possible. Therefore, this project will find its place in any communication scenario where there are devices that can connect the Internet.

1. GİRİŞ

İzlenebilen, algılanabilen ve kontrol edilebilen duyarga, araç ve cihaz gibi nesnelerin internete bağlanarak bir ağ oluşturması olarak tanımlanabilen Nesnelerin İnterneti (Internet of Things – IoT)[1] son yıllarda oldukça yaygın bir kullanım alanına sahiptir.

Nesnelerin İnterneti; akıllı ev uygulamalarından araç takip sistemlerine, yaygın sağlık uygulamalarından sürücüsüz araçlara kadar birçok alanda yaşam kalitesini arttırma, enerjiyi verimli kullanma, olası problemleri önceden tespit edip önleme gibi pek çok fayda sağlamaktadır[2]. Fakat henüz gelişmekte olan bu alanda gizlilik ve güvenilirlik gibi çözülmeyi bekleyen çok ciddi sorunlar vardır.

1.1 Problem Tanımı

IoT standartları ağların denetimi ve mesajların gizliliği konularında henüz bir önlem alınmasını şart koşmamaktadır, dolayısıyla mevcut IoT uygulamalarında veri ve kimlik bilgilerinin çalınması gibi mahremiyeti kötü etkileyen sonuçlar ortaya çıkabilmektedir[3]. Örneğin oluşturulan ağa istenmeyen bir cihaz katılabilir, ağı meşgul ederek sistemin çalışmasını önleyebilir ya da paylaşılan bilgilere ulaşıp yorumlayarak sistem hakkında bilgi sahibi olabilir. Bu bağlamda süregelen araştırmalar olsa da henüz standartlaştırılmış bir çözüm bulunmamaktadır ve mutlak güvenlik gereksinimi olan alanlarda IoT kullanımı yaygınlaşamamaktadır.

1.2 Projenin Amacı

Projenin amacı, veri gizliliği ve bütünlüğü ile ağ güvenliği konularında eksiklikleri giderilmiş güvenli bir nesnelerin interneti ağı tasarlamaktır. Bu amaçla öncelikle bir literatür araştırması yapılarak veri gizliliği ve ağ denetimi üzerine yapılan araştırmalarda önerilmiş çözümlerden en uygun olanları seçilerek gerçeklenecektir. Algoritma seçimi yapılırken bu sistemlerin düşük hafıza ve düşük güç tüketimi sınırlandırmaları göz önünde bulundurularak mümkün olan en hızlı ve doğruluk payı

en yüksek algoritmalara karar verilecektir. Seçilen algoritmalar basit bir benzetim ortamında gerçeklendikten sonra mikrodenetleyiciler ile kurulan örnek bir ağda hafıza ve güç tüketimi kıstaslarına göre değerlendirilip test edilerek algoritmaların verimi ve seçimlerin doğruluğu irdelenecektir.

1.3 Projede Öngörülen Çalışma Planı

IoT için güvenli bir haberleşme sistemi tasarlanırken izlenecek süreç; öncelikle literatür araştırması yaparak en verimli uygulamaların seçilmesi, daha sonra Contiki[4] işletim sistemi ve Cooja[5] benzetim ortamı kullanılarak uygulamaların sisteme gerekli uyarlamalarının yapılması ve son olarak da projenin fiziksel gerçeklemesinin yapılıp güvenli bir haberleşme ağı kurularak gerekli testlerin yapılması şeklinde 3 aşamadan oluşmaktadır.

İlk aşamada yeterli araştırma yapılarak IoT cihazları için en büyük kısıtlama olan güç ve hafıza sorunları için en uygun şifreleme ve kimlik doğrulama algoritmalarına karar verilecektir. Karar aşamasında algoritmaların hafifliği ve sistemin ihtiyaçlarına uygunluğu göz önünde bulundurulacaktır.

İkinci aşamada Cooja benzetim aracı kullanımı öğrenilerek basit bir ağ kurulacaktır. Daha sonra ise seçilen şifreleme ve kimlik doğrulama algoritmalarının gerçeklemeleri yapılarak bu gerçeklemelerin Contiki işletim sistemine uyarlamaları gerçekleştirilecektir. Bu sürecin sonunda ise sistemin hızı, doğruluğu, güç tüketimi gibi faktörlerin şifreleme ve kimlik doğrulama öncesinde ve sonrasında ölçümleri yapılarak seçilen algoritmaların verimi ölçülecektir.

Üçüncü aşamada ise Cooja'da benzetimi yapılmış algoritmalar mikrodenetleyicilere aktarılarak fiziksel bir ağ oluşumu gerçekleştirilip teorik olarak hesaplanan hız, doğruluk, güç tüketimi gibi kıstaslar pratik olarak yeniden hesaplanacak ve seçilen algoritmaların doğruluğu ile ilgili bir yoruma varılacaktır.

1.4 Proje Çalışma Planı Gerçeklenme Düzeyi

Proje önerisinde önerilen çalışma planının, literatür araştırması, algoritma seçimi ve seçilen algoritmaların gerçeklenmesine ve Cooja ortamında çalıştırılmasına ve benzetim ortamında gerekli testlerin yapılarak şifreleme ve kimlik doğrulama algoritmalarının verimlerinin hesaplanmasına dayanan aşamaları tamamlanmıştır,

fiziksel bir ağ ortamında çalıştırılması ise ekipman yetersizliği nedeniye gerçekleştirilememiştir.

Yapılan araştırmalar sonucunda veri gizliliği için PRESENT[6] ve kimlik doğrulama için Nyberg Çoklu Gönderim Kimlik Doğrulama[7] algoritmalarına; paket bütünlüğü için ise seçilen şifreleme algoritmasından üretilen bir özet fonksiyonu[8] algoritmasına karar verilmiş, daha sonrasında ise bu algoritmaların C[9] programlama dilinde gerçeklemeleri yapılmıştır.

Bu aşamada gerçeklenen algoritmaların Cooja benzetim ortamında Düşük Güç ve Kayıplı Ağlar için IPv6[10] Yönlendirme Protokolü (Ipv6 Routing Ptorocol for Low Power and Lossy Networks – RPL)[11] kullanılarak kurulan bir ağ içerisinde çalıştırılması sağlanmıştır.

Daha sonrasında Cooja ortamında çeşitli ağ modelleri kullanılarak hız ve güç tüketim testleri yapılmıştır. Güç tüketimi testleri benzetim ortamının güç tüketimi ölçümüne imkan vermemesi ve fiziksel bir ortamın yoksunluğu dolayısıyla tahmini bir şekilde yapılmıştır.

Projenin son aşaması olan fiziksel gerçekleme bölümü ise gerekli ekipmana sahip olunamadığından gerçeklenememiştir.

1.5 Literatür Araştırması

Mevcut şifreleme ve kimlik doğulama algoritmaları yüksek işlem gücü gerektirmeleri nedeniyle, mikrodenetleyiciler ile kullanılırken yüksek hafıza alanı ve yüksek güç tüketimine neden olurlar. Güç tüketimini azaltmak amacıyla yüksek işlem gücü gerektirmeyen, hafif siklet algoritmalar araştırılmalı, bulunan algoritmaları mikrodenetleyici sistemlere uyumlu hale getirecek yöntemler incelenmelidir.

1.5.1 Veri Gizliliği

Veri şifreleme algoritması olarak seçilen PRESENT'e, yaygın olarak kullanılan Gelişmiş Şifreleme Standardı (Advenced Encryption Standard – AES)[12] ve benzeri algoritmaların Radyo Frekansı ile Tanımlama (Radio Frequency Identification –

RFID)[13] etiketleri ve duyarga ağları gibi kısıtlı ortamlarda istenilen sonucu vermemesi dolayısıyla ihtiyaç duyulmuştur[6].

PRESENT kendisinden önce geliştirilen algoritmaların aksine permütasyon katmanlı bir şifreleme değildir; bit tabanlıdır ve oldukça basittir[6]. PRESENT, 80 bitlik bir anahtar kullanarak 64 bitlik dizilerin şifrelemesini gerçekleştiren hafif sıklet bir IoT şifreleme algoritmasıdır. Üstelik basit bir kablo kullanımı ile donanıma kolayca entegre edilebilmektedir.

1.5.2 Kimlik Doğrulama

IoT ağlarında gönderilen ve alınan verilerin güvenliği kadar ağın güvenliği de önemlidir. Tanınmayan/güvenilmeyen cihazların ağa bağlanması ağın işleyişi tehlikeye atabileceği gibi ağdaki işlem yükünü arttıracağından ağın işleyişini de yavaşlatacaktır. Ağın dışardan gelecek bu gibi saldırılara karşı açık olmaması için haberleşme esnasında kimlik doğrulama yapılmalıdır.

Bu projede kimlik doğrulama için Nyberg Çoklu Gönderim Kimlik Doğrulama yöntemi kullanılmıştır[7]. Nyberg yöntemi, İmza ve Ortam Erişim Yönetimi (Media Access Control – MAC)[14] tabanlı algoritmalara göre daha az anahtar alanına ihtiyaç duyar ve daha az işlem yükü gerektirir.

Nyberg yönteminin düşük hafıza kullanımı, az işlem yükü ve mikrodenetleyicilerin sahip olduğu kısıtlı kaynaklar göz önüne alındığında bu yöntemin kullanılması uygun bulunmuştur.

Ayrıca Nyberg yöntemini gerçeklemek için gerekli olan işlemler, mikrodenetleyicilerin Aritmetik ve Mantık Birimi (Arithmetic Logic Unit – ALU)[15] tarafından desteklendiğinden, işlemler hızlıca ve düşük güç tüketecek şekilde yapılır[7]. Bu nedenle kablosuz ağlarda kullanımı uygundur.

1.5.3 Paket Bütünlüğü

İletim kayıplarının yaşanabileceği ağlarda, veri bütünlüğünün ve haberleşmenin sağlıklı bir şekilde yapılabilmesi için bir denetim mekanizmasının kullanılması oldukça önemlidir. Bunun için bir özet fonksiyonu kullanılmasında karar kılınmıştır.

Özet fonksiyonları farklı girdilerle farklı, aynı girdilerle aynı sonuçları vermek prensibini temel alarak sonuçların karşılaştırmalarını yaparak girişlerde bir değişiklik

olup olmadığının kontrolünü yapan fonksiyonlardır[8]. Bu sayede alınan mesajın özet değeriyle gönderilen mesajın özet değerinin karşılaştırması yapılarak mesajın alıcıya ulaşana kadar bir kayba ya da değişime uğrayıp uğramadığı anlaşılabilir.

Paket bütünlüğünü sağlamak amacıyla gerçeklenecek özet fonksiyonunun, şifreleme algoritması olan PRESENT'in çıktılarını kullanarak paketler arasında basit bir Dışlayıcı Ya Da (Exclusive OR – EXOR)[16] işlemi yapma temeline dayandırılmasına karar verilmiştir.

Kimlik doğrulama için kullanılan Nyberg yöntemi özet fonksiyonu tabanlı olduğundan, kimlik doğrulama ve paket bütünlüğünün sağlanması tek bir işlemle yapılabilmektedir. Bu sayede işlem yükü hafiflemiş ve haberleşmenin daha hızlı yapılabilmesi sağlanmıştır.

2. MATEMATİKSEL ALTYAPI

2.1 Contiki İşletim Sistemi

Kablosuz duyarga ağları (Wireless Sensor Network - WSN)[17], kablosuz haberleşme özelliklerine sahip çok sayıda küçük duyarga cihazdan oluşur[18]. Bu duyarga cihazların kaynakları özellikle güç ve hafıza bakımından genellikle çok kısıtlıdır, ayrıca çok küçük boyutlarda ve düşük maliyetlerle üretilme gereksinimlerine sahip olmaları sistemin karmaşıklığını sınırlar. Üstelik bu duyargalar her geçen gün ucuzlayıp küçülerek daha yaygın bir kullanıma sahip olmakta, fakat hafıza ve güç gibi gereksinimleri azalmamaktadır.

Kablosuz haberleşmenin sınırlı kaynak problemini göz önünde bulundururken aynı zamanda yeterince zengin bir derleme ortamı sunan Contiki işletim sistemi C dilinde gerçeklenmiştir[4]. Çalışma mantığı ise eşzamanlı yürütülen basit iş parçacıklarına ve önceden belirlenmiş bir durumun gerçekleşmesi durumunda dönüş yaparak iş parçacıklarının gerekli düzenlemeleri yapmasını sağlayan olay geri bildirimlerine dayanmaktadır. Bu sayede temelde olay güdümlü bir çekirdek ve üzerinde koşan çoklu kullanıma dayanan uygulama kütüphaneleriyle Contiki, sistem tabanını olabildiğince hafif ve derli toplu tutarken programları dinamik olarak yükleyip kolayca değiştirebilmenin kaynakları kısıtlı bir sistemde de mümkün olabileceğini göstermektedir.

Gömülü sistemlerde kullanılacak bir işletim sisteminin gerçek zamanlı çalışabilmesi, düşük hafızalı cihazlar için uygun bir çalışma ortamı oluşturabilmesi, mümkün olduğunca geniş bir donanım desteği verebilmesi, enerji verimliliği sağlayabilmesi ve internete ve ağdaki diğer cihazlara kolayca bağlanabilmesi, bu alanda bazı protokolleri gerçekleyebilmesi beklenmektedir[19].

Uygulama alanında Contiki, Texas Instruments MSP430[20] ve Atmel AVR[21] başta olmak üzere bir dizin mikrodenetleyici mimarisine aktarılmıştır[4]. Contiki, MSP430 mikrodenetleyicisini 2 kilobayt Rastgele Erişimli Bellek (Random Access Memory –

RAM[22]) ve 60 kilobayt Salt Okunur Bellek (Read Only Memory – ROM[22]) ile kullanmaktadır.

Contiki ağ merkezli bir işletim sistemi olduğu için İnternet Protokol sürüm 4 (Internet Protocol version 4 - IPv4)[10] ve İnternet Protokol sürüm 6 (Internet Protocol version 6 – Ipv6) [10] için tam bir yığın sağlamaktadır ve web sunucu ve istemci uygulamaları için pek çok örnek sunmaktadır[23].

Contiki'de pek çok işletim sisteminden farklı olarak uyku modu gibi bir özellik mevcuttur[19]. Burada mikrodenetleyici, gerekmediğinde çok düşük bir seviyede enerji harcadığı uyku modunda bekletilir, gerektiğinde olay geri bildirimleri sayesinde uyandırılarak işlevini devam ettirir. Böylelikle güç tüketimi azaltılmaktadır.

2.2 Cooja Benzetim Ortamı

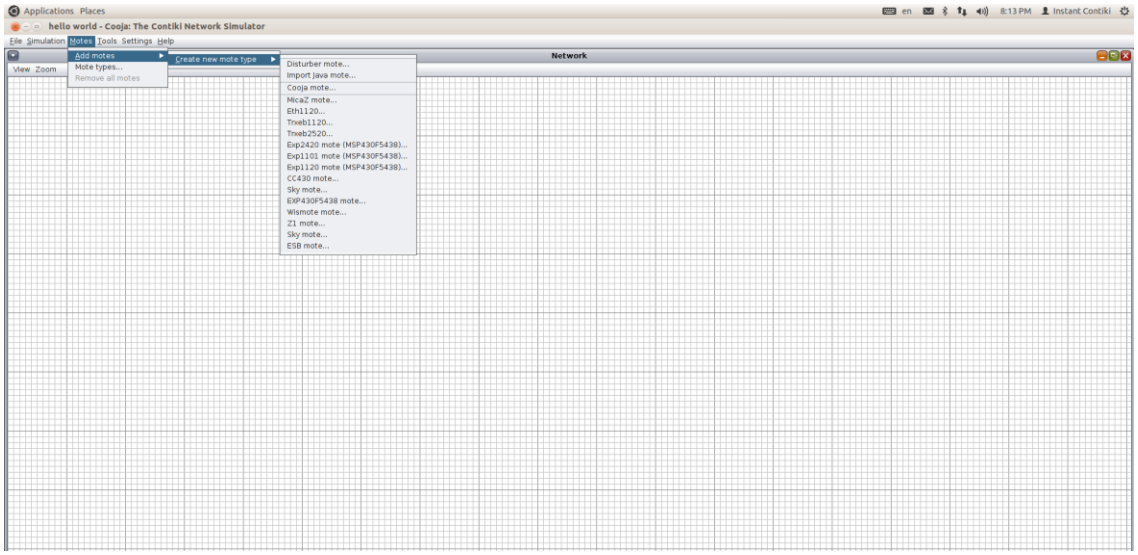
Cooja, Contiki işletim sisteminde çalıştırılan kablosuz duyarga ağları için özel olarak tasarlanmış Java[24] tabanlı bir benzetim ortamıdır[5]. Cooja ortamında her duyarganın bir düğümle temsil edildiği bir ağ kurularak istenilen benzetimler yapılmaktadır. Benzetimi yapılan bir düğümün 3 temel özelliği vardır; veri belleği, düğüm tipi ve donanım çevre birimleri. Bir benzetim ortamında birden fazla düğüm aynı düğüm tipini ve aynı donanım çevre birimlerini kullanarak aynı programı çalıştırılabilmektedir, veri bellekleri ise başlangıçta aynıdır fakat program çalıştırıldıktan sonra değişmektedirler. Düğüm tipleri hem yazılımsal hem donanımsal olarak birbirinden farklıdır ve bir benzetim ortamında farklı düğümler farklı düğüm tiplerini kullanarak aynı programı çalıştırabilmektedirler.

Cooja benzeticinin pek çok özelliğinin değiştirilebildiği ya da eklentilerle zenginleştirilebildiği esnek bir ağ yapısı sunmaktadır[5]. Örneğin Contiki'nin normalde desteklemediği bir düğüm tipi, Java'da gerçeklenmesi yapılarak Cooja ortamında da çalıştırılabilir.

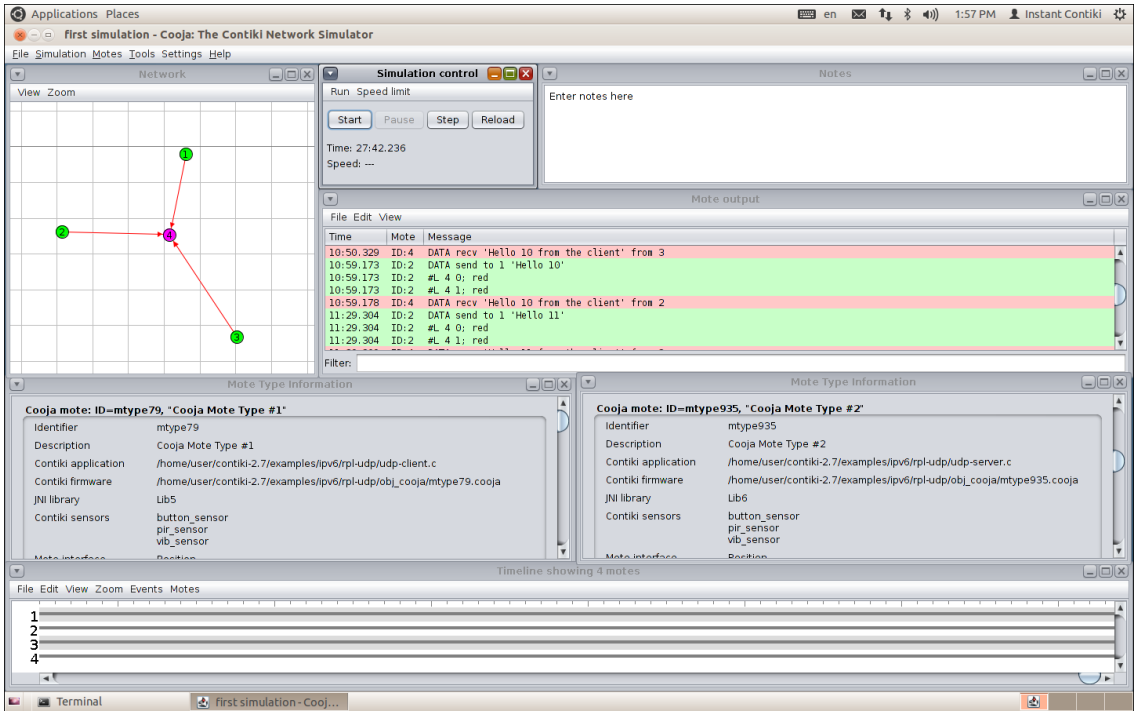
Düğümlerin donanım çevre birimleri arayüz olarak adlandırılmaktadır ve Java benzeticisinin mesaj trafiği ya da Işık Yayan Diyot (Light Emitting Diode – LED)[25]lerin durumu gibi olayları algılanması ve tetiklenmesini sağlamaktadır[5]. Arayüzler ayrıca düğümlerin farkında olamayacağı konum bilgilerini de gösterebilmektedir. Arayüzlere ek olarak benzetim ortamındaki bütün etkileşimleri kontrol etmeye yarayan eklentiler mevcuttur. Eklentiler de arayüzler de benzeticiye

kolayca eklenebilmekte ve kullanıcıya istediği uygulama ortamını hızlı bir şekilde sunmaktadır.

Şekil 2.1'de yeni bir benzetim ortamı oluştururken seçilebilecek düğüm tipleri gösterilmektedir. Farklı düğüm tipleri farklı donanım tabanlarına sahip oldukları için farklı bellek kapasitelerine sahiptir. Bu projede düğüm tipi olarak Cooja düğümü kullanılacaktır.



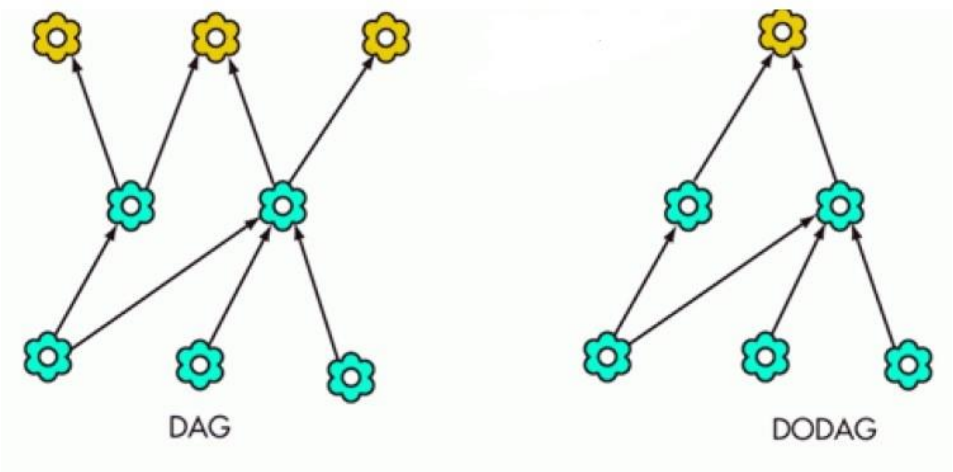
Şekil 2.1: Contiki düğüm tipleri.



Şekil 2.2: Cooja benzetim ortamında gerçeklemesi yapılan bir ağ örneği.

Şekil 2.2'de örnek bir Cooja benzetimi gösterilmektedir. Burada, 3 istemci ve 1 sunucudan oluşan temel bir ağ modellemesi yapılmıştır ve istemciler Kullanıcı Veribloğu İletişim Kuralları (User Datagram Protocol – UDP)[26] protokolünü kullanarak sunucuya basit bir "hello" mesajı göndermektedir. Şekilde yeşil renkle gösterilen 1,2 ve 3 düğümleri istemci; pembe renkle gösterilen 4 düğümü ise sunucuyu temsil etmektedir. Düğümler arasındaki kırmızı oklar ise mesaj trafiğinin yönünü göstermektedir. Benzetim kontrol arayüzü ile program başlatılıp durdurulabilirken her düğümün zaman çizelgesini gösteren arayüzle mesajların gönderilme aralığı görülebilmektedir. Düğüm çıktı arayüzüyle ise hangi düğümlerin ne zaman hangi mesajı gönderip aldığı gibi bilgiler ayrıntılı bir şekilde izlenebilmektedir. Ayrıca düğüm tipi bilgileri panosunda "Cooja mote type #1" olarak tanımlanan istemcilerin "udp-client" kütüphanesini, "Cooja mote type #2" olarak tanımlanan sunucunun ise "udp-server" kütüphanesini kullandığı bilgisine ulaşılabilmektedir. Burada bahsi geçen kütüphaneler Contiki'nin örnek UDP kütüphaneleridir.

2.3 Düşük Güç ve Kayıplı Ağlar için IPv6 Yönlendirme Protokolü



Şekil 2.3: RPL iletim yolu mimarileri.

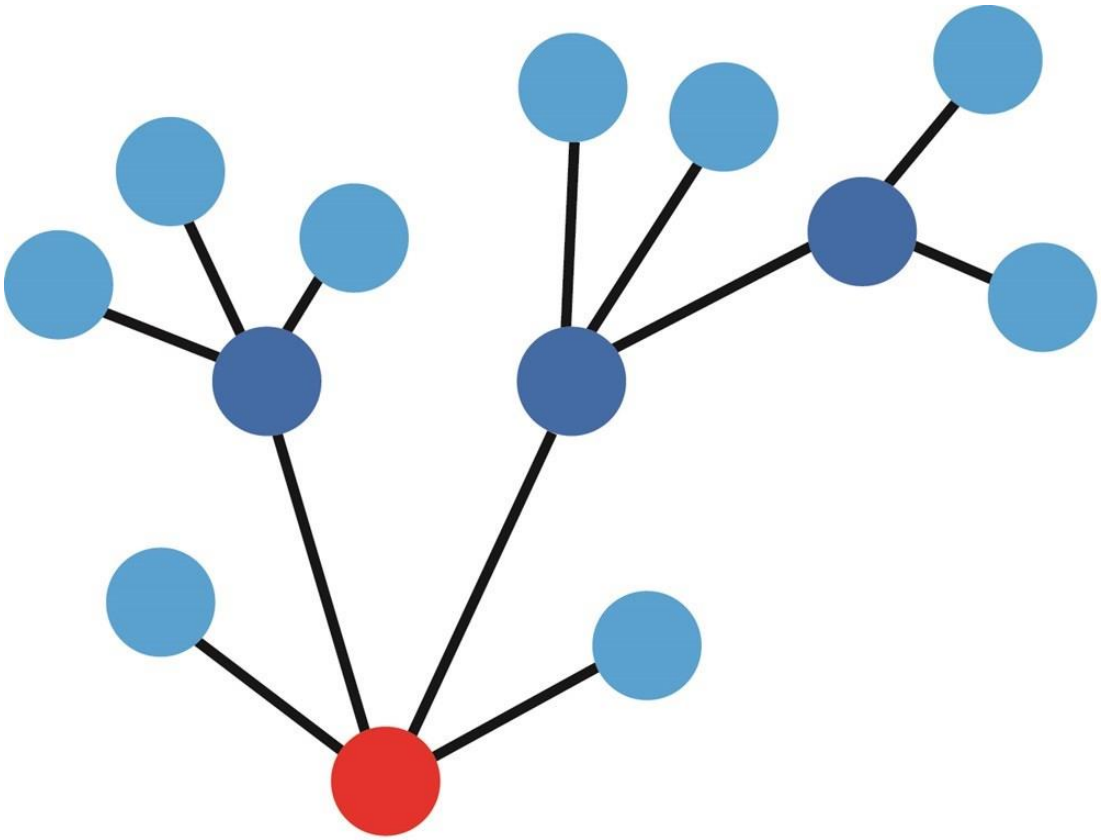
RPL çeşitli bağlantı katmanı mekanizmalarının üzerinde çalışmak üzere tasarlanmış bir mesafe vektörü ve bir kaynak yönlendirme protokolüdür[11]. Düğümlerin merkezi bir noktaya periyodik ölçümler gönderdiği ve bu merkezi noktanın aygıtlara çoklu gönderim yaptığı ağlarda çalışmayı hedeflemektedir, ayrıca merkezi bir düğümün olmadığı ağlardaki iletimi de desteklemektedir.

RPL enerji ve bant genişliği açısından kısıtlı kaynaklı, dolayısıyla paket kayıplarının yüksek olduğu düşük güç ve kayıplı ağlar (Low-power and Lossy Network – LLN)[27]

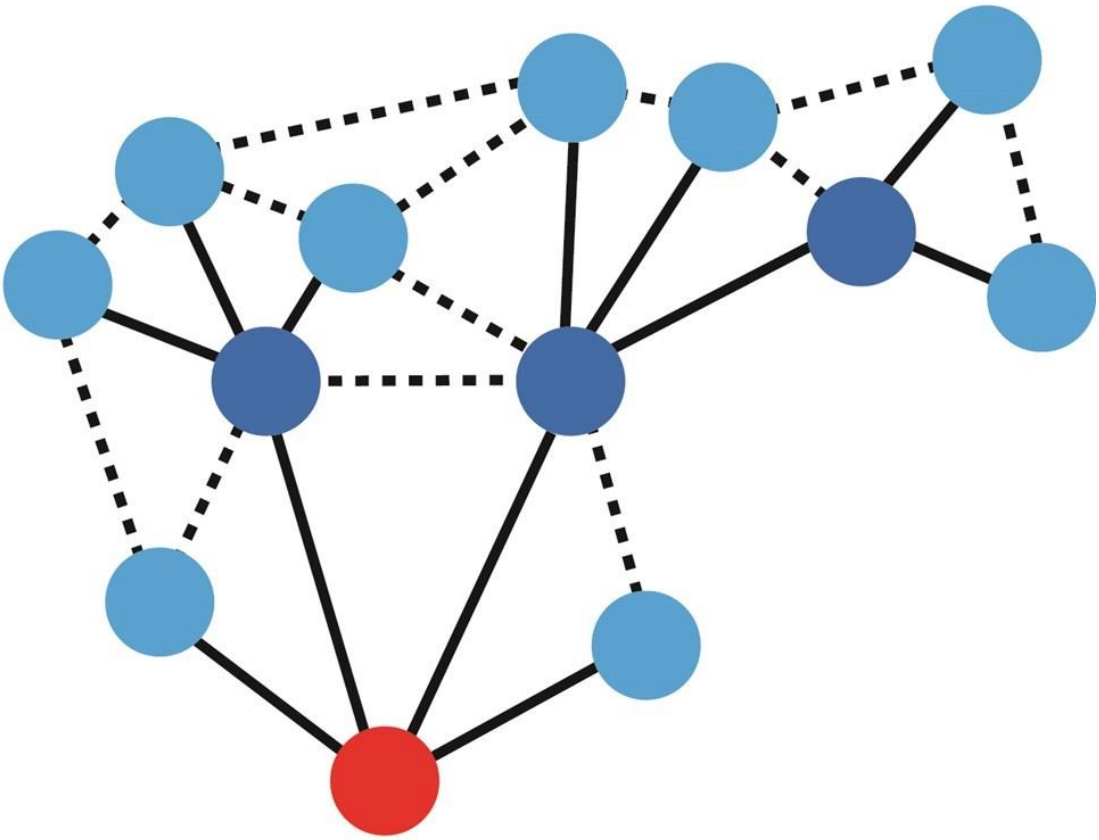
için bir çözüm olarak sunulmuştur[11]. Kayıplı bir ağ, yüksek bir bit hata oranı ve düğümlere erişilemeyen sürelerin büyüklüğüyle tanımlanabilir ve bunlar yönlendirme protokolü tasarımını oldukça etkilemektedir. RPL ağdan kopmuş düğümlerin kısa sürede farkına varıp alternatif yollar sağlamak temeline dayanmaktadır, bu sayede paket ve enerji kaybı ciddi anlamda azaltılmaktadır.

RPL temelli sistemlerde iletim yolları Yönlü Çevrimsiz Grafikler (Directed Acyclic Graphs – DAG)[11] ve Hedefe Yönelik Yönlü Çevrimsiz Grafikler (Destination Oriented DAG – DODAG)[11] mimarileri ile oluşturulmaktadır. Şekil 2.3'te bu mimariler gösterilmektedir[28].

RPL ağ kurulumunda ağaç ve örgü topolojilerini bir arada kullanabilmektedir[28]. Örnek bir ağaç topolojisi Şekil 2.4'te, örgü topolojisi ise Şekil 2.5'te görülebilmektedir. Söz edilen ağ topolojilerinin DAG ve DODAG mimarileri içinde birlikte kullanılması sayesinde RPL tabanlı LNN'ler çevre düğümleri ve üst dizin düğümlerini keşfederek otomatik bir yönlendirme yapabilmekte, herhangi bir düğümün kopması durumunda yeni bir yol üretebilmektedir.



Şekil 2.4: Ağaç topolojisi.



Şekil 2.5: Örgü topolojisi.

RPL Çok Noktadan Noktaya (Multi-Point-to-Point – MP2P)[11], Noktadan Çok Noktaya (Point-to-Multi-Point – P2MP)[11] ve Noktadan Noktaya (Point-to-Point – P2P)[11] iletişim paradigmalarını desteklemektedir[27]. P2P ağdaki herhangi iki düğüm noktası arasında kullanılan iletim yolu iken MP2M yönteminden DODAG düğümüne doğru ileri yönlü aktarımda yararlanılır; P2MP ise DODAG düğümünün aşağı yönlü yönlendirmesinde kullanılmaktadır. Çoklu Gönderim yöntemi de RPL tarafından desteklenmektedir. Çoklu gönderimi destekleyen aygıtlar ağa yönlendirici düğümler olarak katılabilirken desteklemeyen aygıtlar sadece uç düğüm olabilmektedirler.

Kayıplı ağlarda bir düğümün tüm üst düğümlerle bağlantısının kopması sonucu ebeveyn olarak alt düğümlerden birini seçmesi ve böylece mesajın bir döngü içinde dolaşıp DODAG düğümüne çıkamaması çok yaygın bir problemdir[29].

RPL döngü oluşumunu bir düğümün iletim yapabileceği maksimum bir ebeveynlik derecesi belirleyerek olabildiğince azaltmaktadır, döngü oluşumunun engellenemediği

durumlar için de döngü saptaması yaparak en kısa sürede ağ güncellemesi yapmaktadır[28].

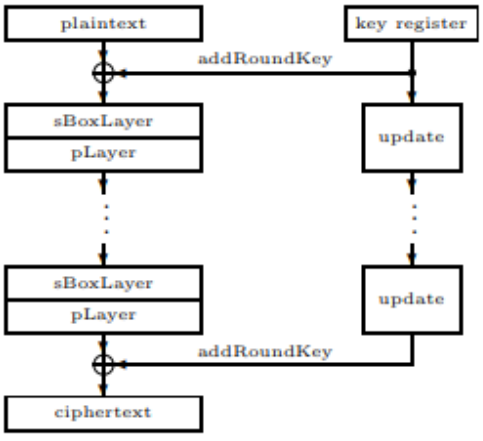
2.4 Şifreleme Algoritması

Blok Şifreleme, 31 turdan oluşan bir yerine koyma – yer değiştirme (substitution-permutation – SP)[6] ağ örneğidir; klasik bir SP ağ örneği herbir turda, önceden belirlenmiş s kutusu (sBox) denilen bir girdiye göre değişiklikler yapan bir yerine koyma katmanı (sLayer) ve sBox'ın çıktısına yine önceden belirlenmiş bir permütasyon işlemi uygulayan bir yer değiştirme katmanından (pLayer) oluşur[30]. SP ağlar istenilen uzunluklarda metin ve anahtar girişlerini sLayer ve pLayer katmanlarından geçirerek istenilen uzunluklarda bir çıktı oluşturma prensibine dayanmaktadırlar.

PRESENT ise 64 bit uzunluğunda bir şifrelenmemiş metin ve 80 ya da 128 bit uzunluğunda bir anahtarı girdi olarak alıp XOR işlemiyle 64 bitlik bir tur anahtarı (round key) üretmekte ve daha sonra bu ranahtarı sLayer ve pLayer'dan geçirerek bir turunu tamamlamaktadır; her turun sonunda ise anahtarını güncellemektedir. Bütün turların sonunda da 64 bit uzunluğunda şifrelenmiş bir metin üretmektedir[6]. Şifrelemenin algoritması Şekil 2.6'da gösterilmektedir.

Bu projede anahtar uzunluğu olarak 80 bit seçilmiştir.

```
generateRoundKeys()
for i = 1 to 31 do
    addRoundKey(STATE,K_i)
    sBoxLayer(STATE)
    pLayer(STATE)
end for
addRoundKey(STATE,K_32)
```



Şekil 2.6: PRESENT şifreleme algoritması.

2.4.1 Anahtar ekleme katmanı (addRoundKey)

PRESENT'te her tur anahtar ekleme katmanıyla başlar, bu katmanda bir XOR işlemi gerçekleştirilir[6]. İlk turda şifrelenmemiş metin ile hiç güncellenmemiş anahtar

XOR'lanırken ilerleyen turlarda sLayer ve pLayer'ın çıktısıyla güncellenmiş anahtar XOR'lanır. 31 turun sonunda ise ayrıca bir XOR işlemi yapılarak şifrelenmiş metin elde edilir. Anahtar ekleme işlemi denklem 2.1'de gösterilmiştir. Burada b şifrelenmemiş metni, k ise anahtarı temsil etmektedir.

$$b_j \rightarrow b_j \oplus k_j^i \tag{2.1}$$

$$b = b_{63}b_{62} \dots b_0 \tag{2.2}$$

$$k_i = k_{63}^i k_{62}^i \dots . k_0^i \tag{2.3}$$

$$0 \leq j \leq 63 \tag{2.4}$$

2.4.2 Yerine koyma katmanı (sLayer)

Bu katman 4 bit girişe 4 bit çıkış sunan bir yerine koyma katmanıdır. Bu nedenle öncelikle 64 bitlik girdiler 4 bitlik 16 elemanlı dizilere dönüştürülerek sBox'a uygun hale getirilmelidir[6]. Yerine koyma işlemi ise Şekil 2.7'de gösterilmektedir. Buna göre eski dizinin 12. elemanı yeni dizinin 0. elemanı; eski dizinin 0. elemanı yeni dizinin 5. elemanı olacaktır.

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S[x]$	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

Şekil 2.7: Yerine koyma katmanı tablosu.

2.4.3 Yer Değiştirme katmanı (pLayer)

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$P(i)$	0	16	32	48	1	17	33	49	2	18	34	50	3	19	35	51
i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$P(i)$	4	20	36	52	5	21	37	53	6	22	38	54	7	23	39	55
i	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
$P(i)$	8	24	40	56	9	25	41	57	10	26	42	58	11	27	43	59
i	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
$P(i)$	12	28	44	60	13	29	45	61	14	30	46	62	15	31	47	63

Şekil 2.8: Yer değiştirme tablosu.

Yer değiştirme katmanı sLayer çıktılarını Şekil 2.8'de verilen tabloya uygun bir permütasyona sokan katmandır[6]. Burada girişlerin i. biti çıkışların P(i). bitine

gitmektedir; yani çıkışın 1. biti girişin 4. biti olacakken; çıkışın 15. biti girişin 60. biti olacaktır.

2.4.4 Anahtar güncelleme katmanı (updateKey)

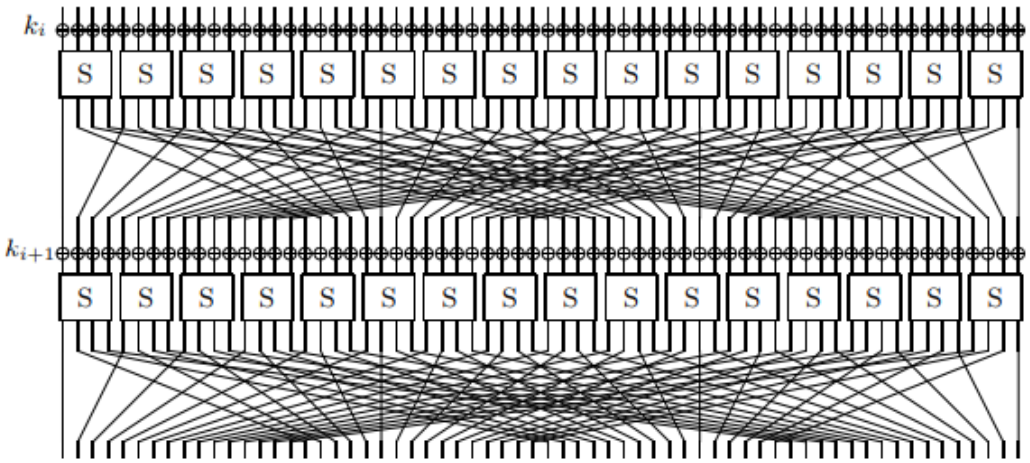
80 bitlik bir anahtar denklem 2.3'teki gibi gösterilmektedir[6]. Fakat 64 bitlik girdilerle XOR'landığı için tur anahtarları 64 bitliktir. Yani i. turdaki anahtar denklem 2.5'te gösterildiği gibi olmak üzere en düşük değerli 64 biti kullanacağı için öncelikle 16 bitlik bir dairesel sağa kaydırma işlemi yaptırılır ve bu işlemin sonucunda denklem 2.6 elde edilir.

$$k_i = k_{63}k_{62} \ldots . k_0 \tag{2.5}$$

$$k_{63}k_{62} \ldots . k_0 = k_{79}k_{78} \ldots . k_{16} \tag{2.6}$$

$$k_{79}k_{78} \ldots . k_1k_0 = k_{18}k_{17} \ldots . k_{20}k_{19} \tag{2.7}$$

Daha sonra 61 bitlik bir dairesel sola kaydırma işlemi yapılır ve yeni anahtar denklem 2.7 şeklini almış olur[6]. Bu işlemin ardından en soldaki 4 bit sBox'a sokulur ve elde edilen çıktının $k_{19}k_{18}k_{17}k_{16}k_{15}$'inci bitleri tur sayısı değerini tutan round_counter değişkeniyle XOR'lanır ve Şekil 2.9'da verilen anahtar güncelleme tablosundaki sonuçlar elde edilmiş olur.



Şekil 2.9: Anahtar güncelleme tablosu.

2.5 Kimlik Doğrulama Algoritması

Nyberg yöntemi, simetrik anahtar tabanlı, tek yönlü bir kimlik doğrulama yöntemidir[7]. Bu mekanizma genel özet fonksiyonları ve bit tabanlı işlemler ile gerçeklenmektedir.

$$Z = H^{\text{Nyb}}(K_Z, Y) = K_Z \odot \prod_{i=1}^{m} \alpha_r(y_i)$$
$$= K_Z \prod_{i=1}^{m} \alpha_r(h(x_i))$$

Şekil 2.10: Nyberg algoritması.

Gönderilmek istenen mesaj öncelikle bir özet fonksiyonu yardımıyla cihaza ve anahtara özgü bir özet fonksiyonu formuna sokulmakta, bu özet fonksiyonu mesajı daha sonra pencereleme yöntemiyle alt parçalara ayırmaktadır[7]. Alt parçalar Nyberg anahtarıyla lojik VE işleminden geçirilerek, kimlik doğrulamada kullanılacak kod elde edilmektedir.

Nyberg kimlik doğrulama algoritmasının genel çalışma prensibi Şekil 2.10'da gösterilmektedir.

2.5.1 Özet Fonksiyonu

Burada kullanılan özet fonksiyonu anahtar tabanlı genel özet fonksiyonlarından biri olup, kimlik doğrulama için kullanılacak cihaza özgü kod bloğunun üretilmesi için kullanılmaktadır[8]. Özet fonksiyonu kodunun uzunluğu denklem 1 ile hesaplanmaktadır.

$$\boldsymbol{l = 2^d * e * t * d} \quad (1)$$

Burda *l* hash uzunluğunu, *t* güvenlik seviyesini ifade etmektedir. *d* sayısı ise pozitif bir tamsayı olup denklem 2 ile hesaplanmaktadır.

$$\boldsymbol{N = 2^d} \quad (2)$$

Burda *N ise* kimlik doğrulama mesajında kullanılacak maksimum blok sayısıdır.

2.5.2 Alpha Fonksiyonu

Alpha fonksiyonu bir önceki adımda üretilen hash kodlarını *d* bitlik *r* tane alt parçaya ayırmaktadır[7]. Bu alt parçanın tüm bitlerinin 0 olması durumunda bunun karşılığı olarak 0, aksi durumda 1 değeri verilmektedir. Böylece bu alt parçalardan *r* uzunluklu bir kimlik doğrulama kodu elde edilmektedir.

2.5.3 Lojik VE

Mesajdaki tüm bloklar Alpha işleminden geçtikten sonra, üretilen yeni alt bloklara lojik VE işlemi uygulanmakta ve böylece bir tane kimlik doğrulama kodu üretilmektedir[7]. Bu kod son olarak kimlik doğrulama anahtarıyla lojik VE işlemine sokulup, kod son halini almaktadır.

Özet fonkisoyonunda cihaza özgü, gizli bir anahtar kullandığı durumlarda, aynı mesaj için bile üretilecek her özet fonksiyon kodu farklı olacağından, kimlik doğrulama anahtarı tercihe bağlı olarak kullanılmayabilir.

2.6 Pil Yönetimi

Kablosuz duyarga ağları küçüklüğü ve ucuzluğu gibi sebeplerden ötürü son yıllarda pek çok alanda kullanılmış ve olabildiğince yaygınlaşmıştır[31]. Fakat bu ağlardaki duyargalar, bataryalarında sınırlı bir güç depolanmış cihazlar oldukları için enerji kapasiteleri çok kısıtlıdır; dolayısıyla yaşam süreleri oldukça kısadır. Bu durum literatürde kendine bir yer edinmiş ve cihaz ömrünü arttırmak için birçok araştırma yapılmıştır.

İlk olarak minimum güç kullanarak maksimum yaşam süresi edinebilmek için güç tüketimleri farklı olan modlar geliştirilmiştir, bu sayede düğümlerin ihtiyaç duyulmadığında bütün işlevlerini kullanıp gereksiz enerji tüketmesinin önüne geçilmiştir[31]. Fakat bu tek başına yeterli olmamıştır çünkü güç tüketimi azaltılsa bile bir ağ kurulduktan sonra düğüm bataryalarının ne zaman biteceği, dolayısıyla pillerin ne zaman değiştirileceği bilgisine de ihtiyaç duyulmaktadır. Bu nedenle ağın çalışma parametrelerine göre çeşitli ölçümler yapılarak, düğümlerin işlevlerine göre harcadıkları ortalama güç hesaplanır ve bu hesapları içeren bir modelleme yapılarak cihazların tükettikleri güç tahminlerinde bulunulur. Modelleme sonucu elde edilen tahminlerde önceden belirlenmiş bir eşik değerinin altına inildiğinde kullanıcılara bir

uyarı gönderilerek bahsi geçen düğümün pillerinin değiştirilmesi gerektiği bilgisi verilir.

Bu projede fiziksel gerçekleme yapma imkanı bulunmadığı için düğümlerin çalıştırdıkları fonksiyonlara göre harcadıkları ortalama güç hesaplaması yapılamamıştır, bu nedenle temsili olarak basit bir tahmin algoritması kullanılmıştır. Bu algoritmada şifreleme yaparak bir mesaj göndermek ya da alınan mesajın şifresini çözmek 3 birim, RPL protokolü gereğince kendinden önceki düğümün gönderdiği mesajı bir sonraki düğüme iletmek ise 2 birim güç tüketiyor ön kabulüyle hareket edilmiştir.

3. PROJENİN GERÇEKLENMESİ

3.1 Şifreleme Algoritmasının Gerçeklemesi

3.1.1 Şifreleme

PRESENT ile şifreleme yapılırken C programlama dili 1 bitlik bir değişken tanımlamaya olanak vermediği için 64 bitlik veri blokları, tipi 8 bitlik işaretsiz tamsayıları ifade eden uint8_t olan 8 elemanlı diziler şeklinde tanımlanacaktır. Aynı şekilde 80 bitlik anahtar ise uint8_t cinsinden 10 elemanlı bir dizi olarak tanımlanacaktır. Bu bölümde, bölüm 2.4'te anlatılan katmanlar sırasıyla gerçeklenecektir ve sunulacak bütün kodlar bu gösterim üzerinden şekillenecektir.

3.1.1.1 Anahtar ekleme katmanı (addRoundKey)

Bu katmanda 80 bitlik anahtar, 16 bit sağa kaydırılma amacıyla dizi indisi 2 arttırılarak XOR işlemine sokulmuştur; böylece anahtarın $k_{79}k_{78} \ldots . k_{16}$'ıncı bitleri kullanılmış olmaktadır. Şekil 3.1'de bu katmanın gerçeklemesi görülmektedir.

```
void add_round_key(uint8_t *cipher, uint8_t *round_key)
{
    uint8_t i;

    for(i = 0; i < 8; i++){
        cipher[i] = cipher[i] ^ round_key[i+2];
    }
}
```

Şekil 3.1: Şifreleme anahtar ekleme katmanı.

3.1.1.2 Yerine koyma katmanı (sLayer)

```
const uint8_t sBox[] =  {0x0C,0x05,0x06,0x0B,0x09,0x00,0x0A,0x0D,
                         0x03,0x0E,0x0F,0x08,0x04,0x07,0x01,0x02};
```

Şekil 3.2: PRESENT sBox tanımlaması.

Şekil 2.7'de tanımlanan yerine koyma katmanı tablosu .c dosyası başlangıcında Şekil 3.2'de gösterildiği gibi tanımlanmıştır.

Bu katmanda sBox 4 bitlik bir yerine koyma yaptığı için öncelikle 8 bitlik cipher elemanlarını uygun forma getirmek için Şekil 3.3'de gösterilen bit işlemleri

yapılır;bunun sonucunda sBox'a 8 bitlik bir giriş sunularak 8 bitlik bir çıkış elde edilmiş olur.

```
void sLayer(uint8_t* cipher)
{
    uint8_t i;

    for (i = 8; i > 0; i--){
        cipher[i-1] = sBox[cipher[i-1]>>4]<<4 | sBox[cipher[i-1] & 0x0F];
    }
}
```

Şekil 3.3: Şifreleme yerine koyma katmanı.

3.1.1.3 Yer değiştirme katmanı (pLayer)

Şifreleme yapılırken yer değiştirme katmanında ise birtakım matematik ve bit işlemleri sonucunda Şekil 2.8'de verilen yer değiştirme tablosuna uygun olarak bir çıktı alınmıştır. Bu katmanın gerçeklemesi Şekil 3.4'te gösterilmektedir.

```
void pLayer(uint8_t* cipher)
{
    uint8_t i;
    uint8_t temp_state[8] = {0};

    for(i = 0; i < 64; i++)
    {
        uint8_t permutation_position = (16*i) % 63;
        if(i == 63 || i == 0)
            permutation_position = i;
        uint8_t element_index = i / 8;
        uint8_t bit_place_in_element = i % 8;
        uint8_t position_index = permutation_position / 8;
        uint8_t bit_destination = permutation_position % 8;
        temp_state[position_index] = temp_state[position_index] |
                                     ((cipher[element_index]>>bit_place_in_element) & 0x01) << bit_destination;
    }

    for(i = 0; i < 8; i++) {
        cipher[i] = temp_state[i];
    }
}
```

Şekil 3.4: Şifreleme yer değiştirme katmanı**.**

3.1.1.4 Anahtar güncelleme katmanı (updateKey)

Bu katmanda Şekil 2.9'daki anahtar güncelleme tablosuna uygun olarak anahtar öncelikle 16 bit sağa kaydırılarak en değerli 64 bitlik kısmı tutulmuş olur. Daha sonra ise 61 bit dairesel sola kaydırma işlemi yapılır, en düşük değerli 16 bitlik kısım ise yeniden kullanılacağı için 2 elemanlı bir temp dizisi oluşturularak korunur. Sonrasında dizinin son elemanının yine sBox'a uygun hale getirilme amacıyla bit işlemleriyle 4 bitlik kısmı alınarak yerine koyma işlemi yapılmış olur. Son olarak ise sBox'a gönderilmiş anahtarın $k_{19}k_{18}k_{17}k_{16}k_{15}$'inci bitleri alınarak tur sayısını tutan round_counter değişkeniyle XOR'lanır. Burada round_counter değişkeni 0'dan başlatıldığı için tur sayısının doğru değerini yansıtması amacıyla 1 eklenerek

kullanılır. Anahtar güncelleme katmanında yapılan işlemler Şekil 3.5'te görülmektedir.

```
void update_key(uint8_t *round_key, uint8_t round_counter)
{
    uint8_t i;
    uint8_t temp[2];

    memcpy(temp, round_key, 2u);

    for(i = 0; i < 8; i++){
        round_key[i] = round_key[i+2];
    }

    round_key[8] = temp[0];
    round_key[9] = temp[1];

    temp[0] = round_key[0] & 0x07;

    for(i = 0; i < 9; i++){
        round_key[i] = round_key[i] >> 3 | round_key[i+1] << 5;
    }

    round_key[9] = round_key[9] >> 3 | temp[0] << 5;

    round_key[9] = sBox[round_key[9] >> 4] <<4 | (round_key[9] & 0x0F);

    round_key[2] ^= (round_counter + 1) >> 1u;
    round_key[1] ^= (round_counter + 1) << 7u;
}
```

Şekil 3.5: Şifreleme anahtar güncelleme katmanı.

3.1.1.5 Şifreleme katmanı (encryptionMemory)

Bu bölüme kadar anlatılan katmanlar şifreleme katmanı denilen bir üst katmanda sırasıyla çağrılarak kullanılmaktadır. Bu projede öncelikle mikrodenetleyicilerin düşük hafıza problemi göz önüne alındığı için tur anahtarlarının hafızada tutulmak yerine her turda tekrardan üretildiği bir şifreleme algoritması kullanılmıştır. Şekil 3.6'da şifreleme katmanının gerçeklemesi gösterilmektedir.

```
void encryption_memory(uint8_t const *state, uint8_t const *key, uint8_t *cipher)
{
    uint8_t round;
    uint8_t round_key[10] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};

    memcpy(cipher, state, 8u);
    memcpy(round_key, key, 10u);

    for(round = 0; round < 31; round++)
    {
        add_round_key(cipher, round_key);
        sLayer(cipher);
        pLayer(cipher);
        update_key(round_key, round);
    }

    add_round_key(cipher, round_key);
}
```

Şekil 3.6: Şifreleme üst katmanı.

Şifreleme yapılırken şifrelenmemiş metin ve anahtar daha sonra tekrar kullanılacağı için değiştirilmemeleri, sabit bir parametre olarak alınmaları gerekmektedir; bu nedenle constant olarak tanımlanmışlardır. Şifreleme işlemlerini yapabilmek için öncelikle bir cipher ve round_key isimli değişkenler oluşturulmuş, şifrelenmemiş metin ve anahtar bu değişkenlere kopyalanmıştır. Daha sonra güncelleme işlemleri için gerekli katmanlarda bu değişkenler kullanılmıştır.

Bu katmanda 31 tur boyunca şifre ekleme, yerine koyma, yer değiştirme ve anahtar güncelleme alt katmanları çağrılarak gerekli güncellemeler yapılmıştır. Ve son olarak Şekil 2.6'da gösterilen algoritmaya uygun olarak turların bitiminde bir anahtar ekleme işlemi daha yapılmıştır.

3.1.2 Şifre çözme

Şifre çözme katmanında şifreleme yapılırken kullanılan katmanların tersi denilebilecek ters yerine koyma, ters yer değiştirme ve ters anahtar güncelleme katmanları bulunmaktadır.

3.1.2.1 Şifre çözme yerine koyma katmanı (sLayer_dec)

```
const uint8_t invsBox[] = {0x05,0x0E,0x0F,0x08,0x0C,0x01,0x02,0x0D,
                           0x0B,0x04,0x06,0x03,0x00,0x07,0x09,0x0A};
```

Şekil 3.7: PRESENT ters sBox tanımlaması.

Bu katmanda Şekil 3.7'de tanımlanan invsBox'a göre yerine koyma yapılır. Yerine koymanın gerçeklenmesinin ise şekil 3.8'de görüldüğü üzere şifreleme yerine koyma katmanından bir farkı yoktur.

```
void sLayer_dec(uint8_t* state)
{
    uint8_t i;

    for (i = 8; i > 0; i--){
        state[i-1] = invsBox[state[i-1]>>4]<<4 | invsBox[state[i-1] & 0x0F];
    }
}
```

Şekil 3.8: Şifre çözme yerine koyma katmanı.

3.1.2.2 Şifre çözme yer değiştirme katmanı (pLayer_dec)

Şekil 3.9'da görüldüğü üzere şifre çözme yer değiştirme katmanıyla, şifreleme yer değiştirme katmanı ufak farklılıklar dışında aynıdır.

```
void pLayer_dec(uint8_t* state)
{
    uint8_t i;
    uint8_t temp_state[8] = {0};

    for(i = 0; i < 64; i++)
    {
        int permutation_position = (4*i) % 63;
        if(i == 63 || i == 0)
            permutation_position = i;
        int element_index = i / 8;
        int bit_place_in_element = i % 8;
        int position_index = permutation_position / 8;
        int bit_destination = permutation_position % 8;
        temp_state[position_index] = temp_state[position_index] |
                                     ((state[element_index]>>bit_place_in_element) & 0x01) << bit_destination;
    }

    for(i = 0; i < 8; i++) {
        state[i] = temp_state[i];
    }
}
```

Şekil 3.9: Şifre çözme yer değiştirme katmanı.

3.1.2.3 Şifre çözme anahtar güncelleme katmanı (updateKey_dec)

Şifre çözme anahtar güncellemesi, şifreleme anahtar güncelleme katmanının küçük değişikliklerle tersten gerçekleştirildiği katmandır. Bu amaçla öncelikle round_counter'la XOR'lama işlemi gerçekleştirilir. Daha sonra en yüksek değerli bitler invsBox'a sokularak yerine koyma işlemleri yapılmadan önceki hallerine dönüştürülür. Bu işlemden sonra önce 56 bit, sonrasında ise 61 bit sağa kaydırma yapılır, böylece tur anahtarları ilk hallerine dönmüş olur. Bu katmanın gerçeklenmesi Şekil 3.10'da anlatılmaktadır.

```
void update_key_dec(uint8_t *round_key, uint8_t round_counter)
{
    uint8_t i;
    uint8_t temp[7];

    round_key[2] ^= round_counter >> 1u;
    round_key[1] ^= round_counter << 7u;

    round_key[9] = invsBox[round_key[9] >> 4] <<4 | (round_key[9] & 0x0F);

    memcpy(temp, round_key, 7u);

    for (i = 0; i < 3; i++) {
        round_key[i] = round_key[i + 7];
    }

    for (i = 0; i < 7; i++) {
        round_key[i + 3] = temp[i];
    }

    temp[0] = round_key[0];

    for (i = 0; i < 9; i++) {
        round_key[i] = (round_key[i + 1] << 3u) | (round_key[i] >> 5u);
    }

    round_key[i] = (temp[0] << 3u) | (round_key[i] >> 5u);
}
```

Şekil 3.10: Şifre çözme anahtar güncelleme katmanı.

3.1.2.4 Şifre çözme katmanı (decryptionMemory)

Şifreleme esnasında her bir turda yeni bir anahtar üretilmektedir. Şifre çözme esnasında ise bu anahtarların çözülerek son tura gelindiğinde başlangıçta constant olarak tanımlanan anahtarın elde edilmesi gerekmektedir. Bu aşamada her turda üretilen anahtarı daha sonra çözmek amacıyla bellekte tutmak iyi bir çözümdür fakat mikrodenetleyicilerle çalışılan sınırlı ağlarda hafıza ciddi bir problem oluşturabilir. Bu nedenle şifre çözme esnasında her bir anahtar önce yeniden üretilir, daha sonra çözülür.

Şifre çözme katmanında öncelikle tur anahtarları yeniden üretilir. Sonrasında şifreleme katmanında turların bitiminde gerçekleştirilen anahtar işlemi gerçekleştirilir. Bu işlemden sonra ise 31 turluk bir döngü gerçekleştirilir. Bu döngüde sırasıyla şifre çözme yer değiştirme katmanı, şifre çözme yerine koyma katmanı, şifre çözme anahtar güncelleme katmanı ve anahtar ekleme katmanları gerçekleştirilir. Döngünün sonunda ise şifrelenmemiş ilk metne ve güncellenmemiş ilk anahtara ulaşılır

```
void decryption_memory(uint8_t const *cipher, uint8_t const *key, uint8_t *state)
{
    uint8_t round;
    uint8_t round_key[10] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};

    memcpy(state, cipher, 8u);
    memcpy(round_key, key, 10u);

    for(round = 0; round < 31; round++) {
        update_key(round_key, round);
    }

    add_round_key(state, round_key);

    for(round = 31; round > 0; round--)
    {
        pLayer_dec(state);
        sLayer_dec(state);
        update_key_dec(round_key, round);
        add_round_key(state, round_key);
    }
}
```

Şekil 3.11: Şifre çözme üst katmanı.

Şekil 3.11'de gösterilen şifre çözme katmanının sonunda ise alıcı gönderilen şifreli mesajı doğru bir şekilde çözmüş olur.

3.2 Kimlik Doğrulama Algoritmasının Gerçeklenmesi

3.2.1 Özet Fonksiyonu

Bu uygulamada kullanılan özet fonksiyonu, PRESENT algoritması kullanılarak üretilen çıktıların birbirleriyle XOR'lanması sonucu bağımsız bir kod üretmektedir.

PRESENT fonksiyonun 64 bitlik bir girdi beklemesi ve özet fonksiyonuna aktarılan mesajın 64 bitten daha küçük olma ihtimaline karşın, özet fonksiyonu mesajın uzunluğunu kontrol etmekte ve gerekli durumlarda mesaja gerektiği kadar 0 ekleyip mesajı 64 bite tamamlayarak PRESENT girdisine uygun hale getirmektedir.

Özet fonksiyonu Şekil 3.12'de gösterildiği gibi uyarlanmıştır.

```
while (0u != byte_remain) {
    if (byte_remain < HASH_BLOCK_SIZE) {
        buff_offset = HASH_BLOCK_SIZE - byte_remain;

        memcpy(block_buff, p_text, byte_remain);
        memset(&block_buff[byte_remain], 0u, buff_offset);

        hash_block(block_buff, p_key, p_hash);
        byte_remain = 0u;
    }
    else {
        memcpy(block_buff, p_text, HASH_BLOCK_SIZE);

        hash_block(block_buff, p_key, p_hash);
        byte_remain -= HASH_BLOCK_SIZE;
    }

    p_text += HASH_BLOCK_SIZE;
    hash_xor_block(p_text, p_hash, block_buff);
}
```

Şekil 3.12: Özet fonksiyonu.

Burada kullanılan *hahs_block* fonksiyonu bölüm 3.3.1.5'de anlatılan *encryptıon_memory* fonksiyonuyla aynıdır.

3.2.2 Alpha Fonksiyonu

```
for (bit_left = NYB_SIZE; 0u != bit_left; bit_left--) {
    if (0u == bit) {
        p_alpha[byte] = 0u;
    }

    mask = NYB_SUBS_WINDOW_MASK << offset;

    if (0u != (*p_hash & mask)) {
        p_alpha[byte] |= (1u << bit);
    }

    bit    += 1u;
    offset += NYB_SUBS_WINDOW_SIZE;

    if (BYTE_SIZE == offset) {
        p_hash += 1u;
        offset = 0u;
    }

    if (BYTE_SIZE == bit) {
        bit    = 0u;
        byte += 1u;
    }
}
```

Şekil 3.13: Alpha fonksiyonu.

C dili bit büyüklüğündeki değişkenlere izin vermediğinden, fonksiyonun gerçekleştirilmesi sırasında maskeleme ve bit kaydırma işlemleri kullanılmıştır. Alpha fonksiyonunun gerçeklemesi ise Şekil 3.13'te görülebilmektedir.

Bu fonksiyonda özet fonksiyonu değeri maskelenerek, sadece işlem yapılan alt parçaya ait bitlerin değerleriyle işlem yapılması sağlanmaktadır. Eğer bu değer 0'a eşit değilse alt parçanın indisi kadar ötelenen 1 sayısı alpha çıktısıyla lojik VEYA işlemine sokulmaktadır; böylece o indise denk düşen bit değeri verilmiş olmaktadır. Aksi durumda lojik VEYA işlemi yapılmayıp sadece indis değeri arttırılmakta ve bit değeri 0 olarak kalmaktadır.

3.2.3 Lojik VE

Kimlik doğrulama için kullanılacak kod, C standart değişkenlerinden daha uzun olabileceği için lojik VE işlemi bir döngü içersinde en küçük değişken tipi kullanılarak yapılmıştır ve gerçeklemesi Şekil 3.14'te gösterilmektedir.

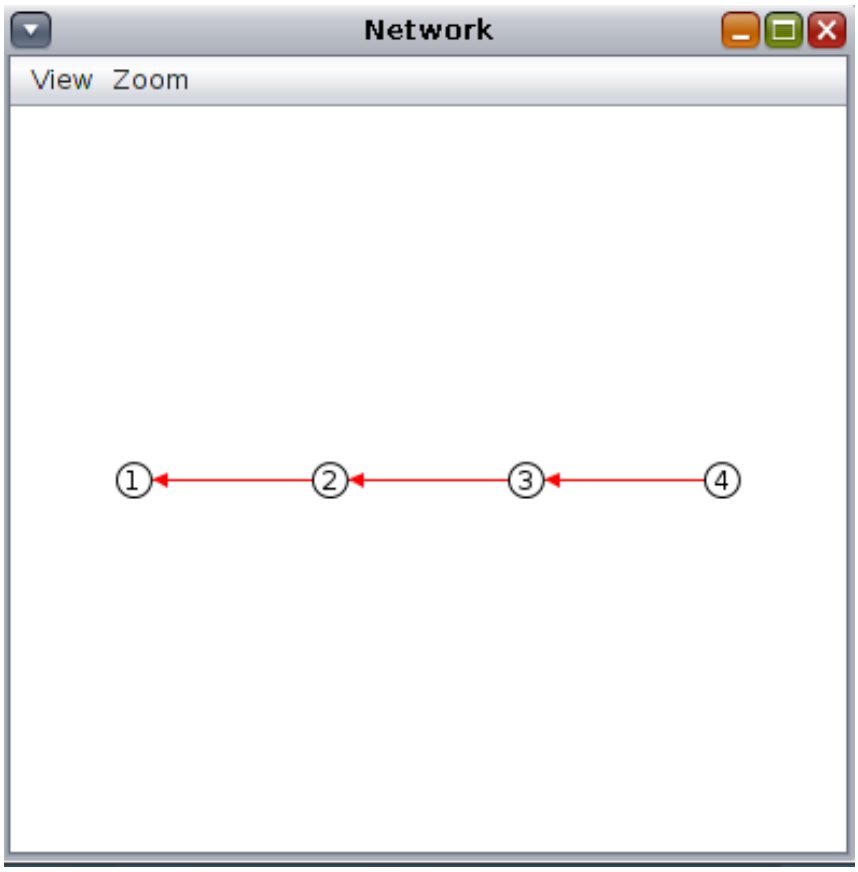
```
static void
nyb_and_subs(byte_t const *p_subs0, byte_t const *p_subs1, byte_t *p_out)
{
    uint_fast8_t byte;

    ASSERT((NULL != p_subs0) && (NULL != p_subs1));
    ASSERT(NULL != p_out);

    for (byte = 0u; byte < NYB_BUFF_SIZE; byte++) {
        p_out[byte] = p_subs0[byte] & p_subs1[byte];
    }
}
```

Şekil 3.14: Lojik VE fonksiyonu.

4. GERÇEKLEME ANALİZLERİ

4.1 Mesaj Gönderim Süreleri

Projenin ilk adımı olarak Cooja simülasyon ortamında bir ağ oluşturulmuştur. Kurulan ağ atlamalı bir yapıda olup her düğümün iletişim menzili sadece kendinden bir önceki ve bir sonraki düğümle sınırlıdır.



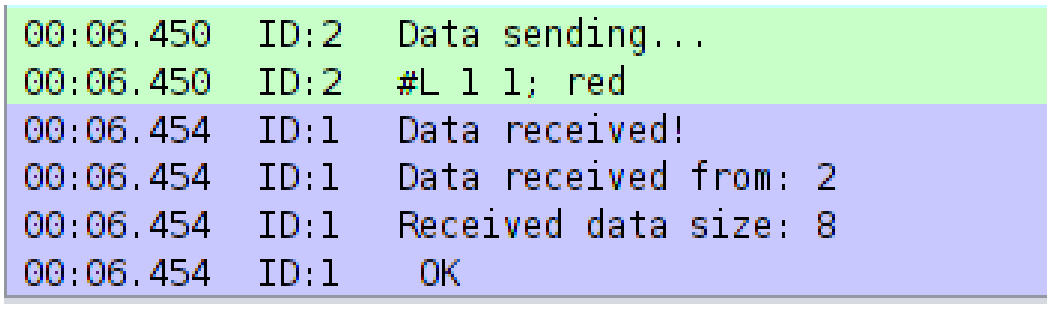
Şekil 4.1: RPL ağ topolojisi.

Şekil 4.1'deki 1 numaralı cihaz yönlendirici, diğer cihazlar düğüm görevindedir. Ağdaki tüm düğümler 1 numaralı yönlendirici aracılığıyla internet ağına bağlanabilir. Bu nedenle tüm düğümler 1 numaralı yönlendiriciye bağlanmak zorundadır.

Şekildeki oklar Cooja simülasyon ortamında düğümlerin birbirleriyle ilişkilerini göstermektedir. Ok yönü düğümlerin mesaj göndermek için seçtikleri ata düğümleri işaret eder.

4.1.1 Çiğ veri gönderimi

Ağdaki her bir düğümünden sırayla 1 numaralı yönlendiriciye mesaj gönderilmiş ve haberleşme süreleri kayıt edilmiştir.



Şekil 4.2: Çiğ veri gönderimi.

Şekil 4.2'de 2 numaralı düğümün mesaj gönderme anındaki Cooja çıktısı verilmiştir. Haberleşmede önce 8 baytlık bir veri şifreleme ve kimlik doğrulama olmadan gönderilmiştir. 00:06.450 anında gönderme işlemi başlamış ve 00:06.454 anında 1 numaralı yönlendiriciye ulaşmıştır. Haberleşmenin toplamda 4 ms sürdüğü gözlemlenmiştir.

Yönlendirci diğer düğümlerin iletişim menzilinde olmadığından, haberleşmek için düğümler mesajları birbirleri üzerinden atlatarak göndermelidir. Her düğümün haberleşmesi için gereken süreler ölçülmüş ve ölçülen değerler Şekil 4.3'te verilmiştir.

Düğüm Numarası	Haberleşme Süresi(ms)
2	4
3	9
4	15

Şekil 4.3: Çiğ veri gönderim süreleri.

4.1.2 Güvenli veri gönderimi

İkinci aşamada haberleşme, şifreleme ve kimlik doğrulama kullanılarak güvenli hale getirilmiştir. Ağ topolojisi Şekil 4.1'deki gibidir.

Haberleşmede iletilecek mesaj ilk aşamadakiyle aynı olup 8 bayt uzunluğundadır. İletilecek pakete aynı zamanda 8 bayt uzunluğunda kimlik doğrulama verisi de eklenmiştir.

Şekil 4.4'te 2 numaralı düğümün mesaj gönderme anındaki Cooja çıktısı verilmiştir. Haberleşmede önce 8 baytlık veri, 8 bayt kimlik doğrulama koduyla birlikte şifrelenmiş şekilde gönderilmiştir. 01:11.540 anında gönderme işlemi başlamış ve 01:11.544 anında 1 numaralı yönlendiriciye ulaşmış ve mesaj doğrulanmıştır. Haberleşmenin toplamda 4 ms sürdüğü gözlemlenmiştir.

```
01:11.540  ID:2   Data sending...
01:11.540  ID:2   #L 1 0; red
01:11.540  ID:2   #L 1 1; red
01:11.544  ID:1   Data received!
01:11.544  ID:1   Data received from: 2
01:11.544  ID:1   Received data size: 16
01:11.544  ID:1    OK
```

Şekil 4.4: Güvenli veri gönderimi.

Aynı işlem ağdaki diğer düğümler için de tekrarlanmış ve sonuçlar Şekil 4.5'te gösterilmektedir.

Düğüm Numarası	Haberleşme Süresi(ms)
2	4
3	14
4	16

Şekil 4.5: Güvenli veri gönderim süreleri.

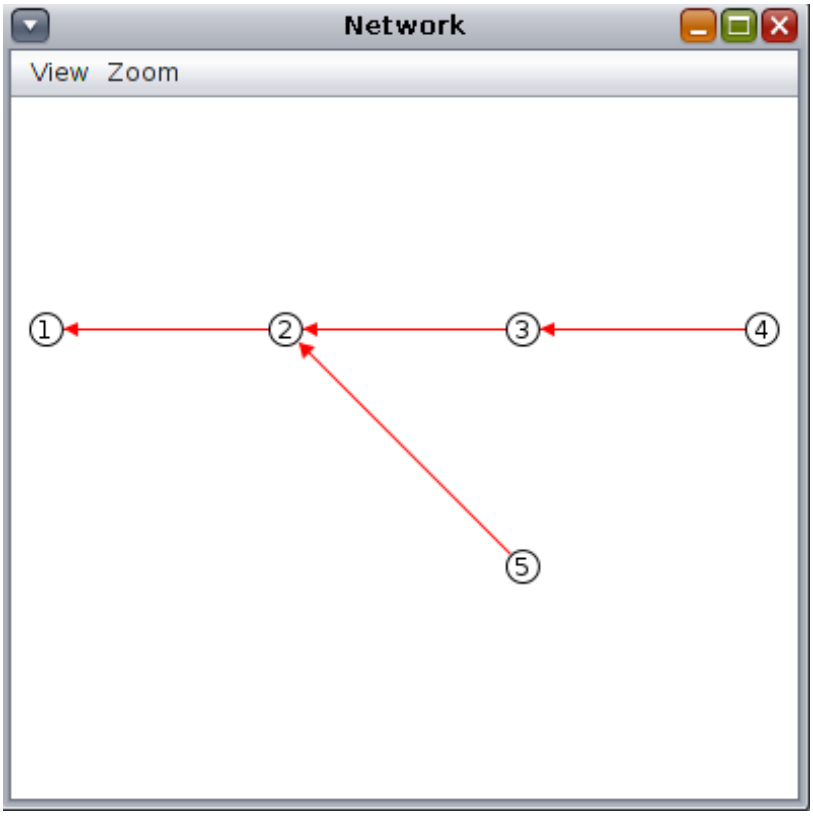
4.2 Pil Yönetimi

Son aşamada düğümlerin pil ömrünü ve pil bitmesi durumunda alınacak önlemleri test etmek için Cooja benzetim ortamında yeni bir ağ oluşturulmuştur. Ağ yapısı Şekil 4.1'dekine benzer olup alternatif bir rota oluşumuna olanak sağlamak amacıyla paralel bir düğüm eklenmiştir.

Şekil 4.6'da Şekil 4.1'de olduğu gibi; 1 numaralı cihaz yönlendirici, diğer cihazlar düğüm görevindedir. Fakat bu ağda 3. düğümle aynı seviyede bir düğüm daha vardır ve 4 numaralı düğüm için alternatif bir yol görevi görecektir.

Bu senaryoda 3 numaralı düğümün işlem yükü arttırılarak pil ömrünün hızlıca

tükenmesi sağlanmıştır. Böylece pil ömrü kritik seviyenin altına indiğinde 3 numaralı düğüm, 1 numaralı yönlendiriciye ağdan düştüğü mesajını yollayacak ve kendi RPL ata düğüm görevini sonlandırarak sadece kendi ölçüm değerlerini iletmeye devam edecektir.



Şekil 4.6: Pil test topolojisi.

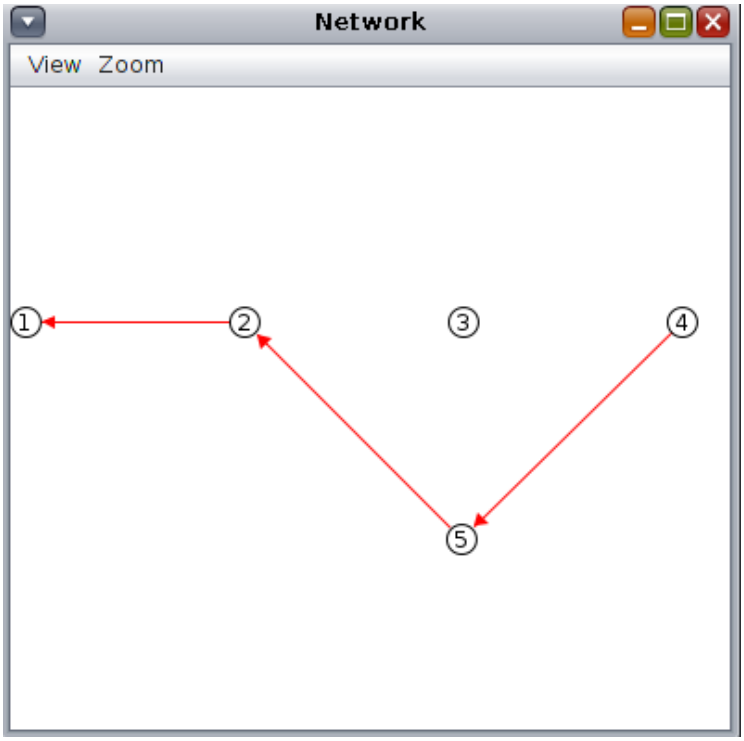
Şekil 4.7'de 3 numaralı düğümün ağdan kopmadan önce gönderdiği mesajın çıktısı görülmektedir.

```
02:15.898  ID:3   Data sending...
02:15.898  ID:3   #L 2 0; red
02:15.898  ID:3   #L 2 1; red
02:15.902  ID:2   #L 1 0; red
02:15.902  ID:2   #L 1 1; red
02:15.907  ID:1   Data received!
02:15.907  ID:1   Data received from: 3
02:15.907  ID:1   Received data size: 8
02:15.907  ID:1    Node 3 dropped!
```

Şekil 4.7: Pil uyarı mesajı.

3 numaralı düğümün ağdan kopması sonucu, bu düğüme bağlantılı olan diğer düğümler kendilerine yeni bir rota oluşturmak zorundadır. Bu amaçla 4 numaralı düğüm iletişim menzilinde olan diğer düğümlerden birini kendine ata düğüm olarak

seçecek ve haberleşmeyi bu düğüm üzerinden yapmaya devam edecektir. Şekil 4.6'daki gibi bir ağda 4 numaralı düğümün veri aktarımına 5 numaralı düğüm üzerinden devam ettiği görülecektir.



Şekil 4.8: Yeni haberleşme hattı.

Şekil 4.8'de 3 numaralı düğüm hattan düştükten sonra oluşan yeni haberleşme hattı verilmiştir. Burada 3 numaralı düğüm kendini haberleşmeye kapatmış olmasına karşın gerekli olan durumlarda 2 numaralı düğüm üzerinden yönlendiriciyle haberleşebilecektir.

5. SONUÇLAR

5.1 Çalışmanın Uygulama Alanları

Kablosuz duyarga ağlarının en temel sorunu olan gizlilik ve güvenlik sorununu, kablosuz duyarga cihazlarının en belirleyici özelliği olan hafıza limitini etkilemeden ve güç tüketimini göz önünde bulundurarak hem çok fazla işlemci gücü gerektirmeden hem de güç problemi için alternatif bir çözüm sunan proje internete bağlanabilen cihazların bulunduğu her alanda kullanıma uygundur.

Proje çıktılarından her biri özel bir ihtiyacı karşılayabilecek kadar kapsamlıdır. Örneğin, duyargaların herhangi bir kopma olmadan belirli aralıklarla ölçüm yapması gereken büyük sistemlerde pil tüketim tahmini algoritması ile kullanıcılara vaktinde uyarı gönderilerek sistemde herhangi bir aksaklık olmasının önüne geçilebilinecektir. Ya da gönderilen mesajların üçüncü bir kişi, cihaz ya da uygulama tarafından okunmaması gereken gizli haberleşmelerde şifrelenmiş metinler bir şekilde dışarıya sızsa bile okunup anlamlandırılamayacaklardır. Ayrıca ağ saldırılarının yüksek olduğu bir sistemde ise istenmeyen bir cihazın ağa katılıp verilere ulaşması da kimlik doğrulama algoritması sayesinde sisteme fazladan bir yük eklemeden engellenebilmektedir.

Proje bir bütün olarak da güvenli bir ağ, gizli bir haberleşme, hafif bir protokolle çalışma imkanlarını bir arada sunduğu için pek çok ihtiyaca aynı anda cevap verebilmektedir.

5.2 Gerçekçi Tasarım Kısıtları

5.2.1 Maliyet

Mikrodenetleyicilerle çalıştırılarak fiziksel bir ağ kurulması planlanan çalışma fiziksel yetersizliklerden ötürü benzetim ortamında devam ettirilmiştir, bu nedenle mikrodenetleyiciler için herhangi bir harcama yapılmamıştır. Kullanılan benzetim ortamları açık-kaynak olduğu için maliyeti yoktur ve referans verilen makalelere İTÜ Eduroam aracılığıyla ulaşıldığı için bu bağlamda da herhangi bir ücret ödenmemiştir.

Projenin tek maliyeti saatlik 20₺ üzerinden haftada 8 saatlik bir çalışma için kişi başı aylık 640₺ civarında bir işçilik maliyetidir.

5.2.2 Standartlar

Çalışma süresince yapı için 6LowPAN standardı; kimlik doğrulama içinse IPv6 standardı esas alınmıştır.

5.2.3 Sosyal, çevresel ve ekonomik etki

IoT standartları henüz güvenlikle ilgili bir önlem alınmasını şart koşmamaktadır. Güvenlik önlemleri mevcut olmadığı için de IoT kullanımı yaygın hale gelememektedir ve veri ve kimlik bilgilerinin çalınması gibi mahremiyeti kötü etkileyen sonuçlar ortaya çıkmaktadır. Güvenli bir IoT ağının kurulabilmesi ile IoT'nin ekonomide kullanımı artacak ve insanların güvenleri de sarsılmayacaktır.

5.2.4 Sağlık ve güvenlik riskleri

Çalışılan proje tamamlanma sürecinde herhangi bir güvenlik tehlikesi oluşturmamıştır, aksine IoT'nin bilinen güvenlik açıklarını kapatmayı amaçlamıştır. Sağlık hususunda ise bilinen bir tehlikesi yoktur.

5.3 Geleceğe Yönelik Öneriler

Nesnelerin İnterneti her geçen gün daha yaygın ve daha ulaşılabilir olmaktadır, bu amaçla yapılan binlerce proje mevcuttur fakat henüz güvenlik konusunda standartlaşmış bir çözüm bulunmamaktadır. Bu nedenle bir süre sonra IoT ağlarının en büyük önceliği dışarıdan gelen saldırılara karşı korunmuş ve gizliliğinden şüphe duyulmayan bir haberşme ortamı olacaktır. Bunu sağlayabilmek için bu alanda çalışmalara hız kesmeden devam edilmeli, daha hafif ve daha güvenli bir haberleşme sistemi üzerinde çalışılmalıdır.

KAYNAKLAR

[1] **Cinkler, T., Hejazi, H., Lengyel, L. & Rajab, H.** (2018). Survey of platforms for massive IoT, *IEEE International Conference on Future IoT Technologies*, Eger, Hungary.

[2] **Jha, N. K. & Mosenia, A.** (2017). A comprehensive study of security of Internet-of- Things, *IEEE Transactions on Emerging Topics in Computing*, *5*(4), 586–602.

[3] **Hai, H. D., Rattanalerdnusorn, E., Thaenkaew, P. & Vorakulpipat, C.** (2018). Recent Challenges, Trends, and Concerns Related to IoT Security: An Evolutionary Study, *International Conference on Advanced Communications Technology*, Pathumthani, Thailand.

[4] **Dunkels, A., Gronvall, B. & Voigt, T.** (2004). Contiki - a lightweight and flexible operating system for tiny networked sensors, *29th Annual IEEE International Conference on Local Computer Networks*, Tampa, USA.

[5] **Dunkels, A., Erikkson, J., Finne, N., Osterlind, G. & Voigt, T.** (2006). Cross-level sensor network simulation with COOJA, *31st IEEE Conference on Local Computer Networks*, Tampa, USA.

[6] **Bogdanov, A., Knudsen, L. R., Leander, G., Paar1, C., Poschmann, A., Robshaw, M. J. B., Seurin, Y., & Vikkelsoe, C.** (2007). PRESENT: An Ultra-Lightweight Block Cipher, *Conference on Cryptographic Hardware and Embedded Systems*, Vienna, Austria.

[7] **Du, X., Han, X., Yao, X. & Zhou, X.** (2013). A Lightweight Multicast Authentication Mechanism for Small Scale IoT Applications, *IEEE Sensors Journal, 13(10).*

[8] **Grochol, D. & Sekanina, L.** (2017). Multi-objective evolution of hash functions for high speed networks, *2017 IEEE Congress on Evolutionary Computation*, San Sebastian, Spain.

[9] **Kernighan, B.** (1983). A programming language called C: The C programming language is claimed to be compact, efficient, and expressive, to the point of supplanting assembly language on Unix, *IEEE Potentials*, *2*(December), 26-30.

[10] **Chen, W. & Li, S.** (2013). Client-based Internet Protocol version 4–Internet Protocol version 6 translation mechanism for Session Initiation Protocol multimedia services in next generation networks, *IET Networks*, *2*(3), 115-123.

[11] **Alexander, R., Brandt, A., Hui, J., Kelsey, R., Levis, P., Pister, K., Struik, R., Thubert, P., Vasseur, J. & Winter, T.** (2012). RPL: IPv6 routing protocol for low-power and lossy networks, *IETF March 2012*.

[12] **Jonwal, S. U. & Shingare, P. P.** (2017). Advanced Encryption Standard (AES) implementation on FPGA with hardware in loop, *2017 International Conference on Trends in Electronics and Informatics,* Tirunelveli, India.

[13] **Liu, Y. & Yang, Xi.** (2018). Chipless radio frequency identification tag design with modified interdigital hairpin resonators, *2018 International Conference on Intelligent Transportation, Big Data & Smart City,* Xiamen, China.

[14] **IEEE** (2018). IEEE Draft Standard for Local and Metropolitan Area Networks: Media Access Control (MAC) Security (P802.1AE).

[15] **An, Q., Beux, S. L., O'Connor, I., Klein, J. O. & Zhao, W.** (2017). Arithmetic Logic Unit based on all-spin logic devices, *15th IEEE International New Circuits and Systems Conference*, Strasbourg, France.

[16] **Takiguchi, K.** (2014). Exclusive OR circuit for optical signals using combined technology of photonics and electronics, *Electronics Letters, 50*(9).

[17] **Dong, N., Li, D. & Wang, Y.** (2017). Cellular automata malware propagation model for WSN based on multi-player evolutionary game, *IET Networks, 7*(3), 129–135.

[18] **Akyildiz, I. F., Cayirci, E., Sankarasubramaniam, Y. & Su, Weilian.** (2002). A survey on sensor networks, *IEEE Communications Magazine, 40*(8), 102–114.

[19] **Azzouz, S. L., Kriaa, L. & Sabri, C.** (2017). Comparison of IoT constrained devices operating systems: a survey, *14th International Conference on Computer Systems and Applications,* Hammamet, Tunisia.

[20] **Texas Instruments** (2009). MSP430 hardware tools user's guide, SLAU278C datasheet.

[21] **Atmel** (2016). AVR instruction set manual, 0856L dahasheet.

[22] **Allan, R.** (1975). Semiconductor memories: More users are opting for RAMs, ROMs, and PROMs as chip function densities increase and price per bit drops, *IEEE Spectrum, 12*(8), 40-45.

[23] **Eskofier, B., Kugler, P. & Nordhus, P.** (2013). Shimmer, Cooja and Contiki: a new toolset for the simulation of on-node signal processing algorithms, *IEEE International Conference on Body Sensor Networks*, Cambridge, USA.

[24] **Hamilton, M. A.** (1996). Java and the shift to net-centric computing, *Computer, 29*(8), 31-39.

[25] **Tompkins, D., Zhang, J. & Zurfi, A.** (2014). Efficiency measurement of white LED devices, *40th Annual Conference of the IEEE Industrial Electronics Society*, Dallas, USA.

[26] **Artho, C., Hagiya, M., Sebih, N., Tanabe, Y., Yamamoto, M. & Weitl, F.** (2014). Software model checking of UDP-based distributed applications, *Second International Symposium on Computing and Networking,* Shizuoka, Japan.

[27] **Bahk, S., Kim, H., Kim, H. & Paek, J. (2016).** Load balancing under heavy traffic in RPL routing protocol for low power and lossy networks, *IEEE Transactions on Mobile Computing, 16*(4), 964 - 979.

[28] **Gaddour, O. & Koubâa, A.** (2012). RPL in a nutshell: a survey, *Computer Networks, 56*(14), 3163-3178.

[29] **Dunkels, A. & Vasseur, J. P.** (2010). *Interconnecting Smart Object with IP: The Next Internet.* Burlington, Morgan Kaufmann Press.

[30] **Menezes, A., Oorschot, P.C. & Vanstone, S.** (1996). *The Handbook of Applied Cryptography.* CRC Press.

[31] **Albea, C., Lesecq, S., Lombardi, W., Mokrenko, O. & Vergara-Gallego, M.** (2015). WSN power management with battery capacity estimation, *IEEE 13th International New Circuits and Systems Conference*, Grenoble, France.

ÖZGEÇMİŞ

Ad-Soyad **:** Büşra ÖZEN

Doğum Tarihi ve Yeri **:** 13.11.1995 - Artvin

E-posta **:** ozenbus@itu.edu.tr

Eğitim Durumu (Kurum ve Yıl)

Lise **:** Tekirdağ Ebru Nayim Fen Lisesi (2009-2013)

Lisans **:** İstanbul Teknik Üniversitesi – Elektronik ve Haberleşme Mühendisliği (2013-2018)

ÖZGEÇMİŞ

Ad-Soyad **:** Furkan KURT

Doğum Tarihi ve Yeri **:** 05.01.1993 - İstanbul

E-posta **:** kurtfu@itu.edu.tr

Eğitim Durumu (Kurum ve Yıl)

Lise **:** Vatan Mesleki ve Teknik Anadolu Lisesi (2007-2011)

Lisans **:** İstanbul Teknik Üniversitesi – Elektronik ve Haberleşme Mühendisliği (2012-2018)