Mühendislik Bilimleri ve Tasarım Dergisi
8(1), 285 – 304, 2020
e-ISSN: 1308-6693

Journal of Engineering Sciences and Design
DOI: 10.21923/jesd.530295

Araştırma Makalesi Research Article

NESNELERİN İNTERNETİ İÇİN HİBRİT UYGULAMA KATMANI PROTOKOL TASARIMI

Erdal ÖZDOĞAN[1*], O. Ayhan ERDEM[2]

[1] Gazi Üniversitesi, Bilişim Enstitüsü, Bilişim Sistemleri Anabilim Dalı, Ankara, Türkiye
[2] Gazi Üniversitesi, Teknoloji Fakültesi, Bilgisayar Mühendisliği Bölümü, Ankara, Türkiye

Anahtar Kelimeler
Nesnelerin İnterneti,
IoT Protokolleri,
Hibrit IoT Protokolü,
IoT Uygulama Protokolü,
IoT Protokol Tasarım.

Öz

Nesnelerin İnternetinde, iletişim başarımını etkileyen önemli faktörlerden biri kullanılacak olan mesajlaşma protokolüdür. MQTT, XMPP ve AMQP merkezi yapıda ve sunucu aracılığıyla haberleşen uygulama protokolleridir. DDS ve CoAP ise özellikle gerçek zamanlı uygulamalarda doğrudan iletişim için kullanılan uygulama protokolleridir. Nesnelerin İnternetinin giderek yaygınlaşması ve kullanım senaryolarının farklı gereksinimlere sahip olması, veri iletişiminde yeni yaklaşımların geliştirilmesini gerekli kılmaktadır. Bu çalışmada hem merkezi sunucu aracılığıyla hem de doğrudan iletişim sağlayabilen UDP tabanlı hibrit bir protokol tasarlanmıştır. Geliştirilen protokolün çalışma mantığı, paket yapısı ele alınmış ve işlevsellik yönünden MQTT protokolü ile karşılaştırılmıştır.

HYBRID APPLICATION LAYER PROTOCOL DESIGN FOR INTERNET OF THINGS

Keywords
Internet of Things,
IoT Protocols,
Hybrid IoT Protocol,
IoT Application Protocol,
IoT Protocol Design.

Abstract

One of the important factors affecting communication performance in the Internet of Things is the messaging protocol. MQTT, XMPP and AMQP are centralized application protocols that communicate through the server. DDS and CoAP are application protocols that can communicate directly, especially in real-time applications. As the Internet of Things is becoming more widespread and usage scenarios have different requirements, new approaches to data communication are required. In this study a UDP based hybrid protocol is designed which can communicate both directly and through central server. Operating logic and packet structure of the developed protocol is examined and compared with MQTT protocol with respect to their functionality.

Alıntı / Cite

Özdoğan, E., Erdem, O.A., (2020). Nesnelerin İnterneti İçin Hibrit Uygulama Katmanı Protokol Tasarımı, Mühendilkik Bilimleri ve Tasarım Dergisi, 8(1), 285-304.

Yazar Kimliği / Author ID (ORCID Number)
E. Özdoğan, 0000-0002-3339-0493
O. A. Erdem, 0000-0001-7761-1078

Makale Süreci / Article Process	
Başvuru Tarihi / Submission Date	21.02.2019
Revizyon Tarihi / Revision Date	09.09.2019
Kabul Tarihi / Accepted Date	27.11.2019
Yayım Tarihi / Published Date	20.03.2020

1. Giriş (Introduction)

Başlangıcından günümüze kadar İnternetin geçirdiği büyük ve öngörülemeyen gelişimi, önümüzdeki yıllarda İnternet teknolojisindeki gelişmelerin ve yeniliklerin dünya çapında büyük yankılar uyandıracağını göstermektedir. 2012 yılında, internete bağlı cihazların sayısı, yeryüzünde yaşayan insanların sayısını aşmış ve 2020 yılında 26 ile 50 milyar arasında nesnenin internete bağlı olması beklenmekte ve bu cihazlardan milyonlarca gigabyte veri üreteceği tahmin edilmektedir (Anjum vd, 2014). Heterojen yapıda üretilecek olan bu devasa ölçüdeki veri yığınının organize edilmesi, anlamlandırılması ve enformasyona dönüştürülmesi Nesnelerin İnternetinin (Internet of Things- IoT) karşı karşıya kaldığı önemli problemlerden biri olarak görülmektedir (Atzori, vd., 2010). Yakın gelecekte her cihazın veya nesnenin doğrudan ya da dolaylı olarak bir IP adresine sahip

[*] İlgi yazar / Corresponding author: erdalozdogan@hotmail.com, +90-312-296-9566

olacağı öngörüsü, IP desteği olan protokollerin geliştirilmesi gerekliliğini ortaya koymaktadır (Höller vd., 2014). Mevcut İnternet protokollerinin bu amacı karşılamadaki yetersizlikleri yeni protokollerin geliştirilmesine neden olmaktadır. Bu amaçla, nesnelerin internetinde veri transferi sağlamak amacıyla Message Queuing Telemetry Transport (MQTT), Constrained Application Protocol (CoAP), Advanced Message Queuing Protocol **(**AMQP), Data Distribution Service **(**DDS) ve Extensible Messaging and Presence Protocol (XMPP) protokolleri geliştirilmekte ya da IoT gereksinimlerine göre uyarlanmaktadır (Gubbi vd., 2013). Diğer taraftan ITU-T'nin 2014 yılında yayınladığı "*M2M service layer: APIs and protocols overview*" isimli raporda, gelişen teknolojileri ve kullanım senaryolarını karşılamak amacıyla yeni protokollerin geliştirilmesi gerekliliği belirtilmektedir (ITU-T, 2014).

MQTT, AMQP ve XMPP protokolleri veri transferinde sunucu kullanan, merkezi protokollerdir (Talaminos-Barroso vd., 2016) . Bu yaklaşımda veriye erişmek isteyen istemci ile veri kaynağı arasında doğrudan bir iletişim bulunmamaktadır. CoAP ve DDS gibi protokoller ise, merkezi olmayan yaklaşım ile çalışan, doğrudan iletişim sunan protokollerdir (Bellavista & Zanni, 2016; Dizdarevic vd., 2018). Aracı sunucu kullanan merkezi yaklaşım ile merkezi olmayan yaklaşımın her ikisinin de çeşitli avantajları ve dezavantajları bulunmaktadır. Merkezi olmayan yaklaşımın en önemli avantajlarından biri doğrudan iletişim kurabilmesidir (Collina vd., 2014). Ancak yönetimsel güçlükler bu mimarinin başta gelen dezavantajıdır (Fortino vd., 2014). Merkezi iletişimin en önemli avantajlarından biri ise, iletişimin sunucu tarafından kontrol edilebilir olmasıdır. Ancak sunucunun kullanılması, tek bir hata noktasının oluşmasına neden olmaktadır (Bellavista & Zanni, 2016; H. Kim, 2017; Su vd., 2014; Thota & Kim, 2016; Pathania, 2017). Sunucu devre dışı kaldığında veri iletişimi kesilmektedir.

Bu çalışma kapsamında, veri iletişiminde hem sunucu tabanlı çalışan hem de doğrudan iletişim yeteneğine sahip, hibrit bir uygulama katmanı protokolü (hIoT) tasarlanmıştır. Tasarlanan protokol kısıtlı kaynaklara sahip cihazlarda, düşük bant genişliğine sahip ortamlarda, sensörlerden üretilen düşük boyutlu verilerin transferini sağlamak amacıyla geliştirilmiştir. Geliştirilen protokolde sensörler tarafından sunulan hizmetlerin, doğrudan ya da sunucu tabanlı iletişim seçimi yapılabilmektedir. Bu sayede farklı amaçlar ve ihtiyaçlar doğrultusunda her iki iletişim yöntemi kullanılabilecek ve bu iki yöntem arasında dinamik geçiş yapılabilmektedir.

Makalenin ikinci bölümünde, IoT protokollerine yönelik yapılan akademik çalışmalar ele alınmıştır. Üçüncü bölümde, yaygın olarak kullanılan CoAP ve MQTT protokolleri incelenmiştir. Makalenin dördüncü bölümünde, tasarlanan uygulama protokolünün genel çalışma prensibi, paket yapıları ve protokol bileşenleri incelenmiştir. Beşinci bölümde, geliştirilen protokolün deneysel başarım değerlendirmeleri ele alınmış ve yaygın olarak kullanılan MQTT protokolü ile karşılaştırılmıştır. Makalenin son bölümünde, çalışmanın özetine ve önerilen protokolün avantajlarına yer verilmiş, gelecekteki çalışmalara ışık tutmak amacıyla önerilerde bulunulmuştur.

2. İlgili Çalışmalar (Related Work)

IoT'nin eksiksiz olarak hayata geçirilmesinde protokoller önemli bir rol oynamaktadır. IoT protokolleri ile ilgili son yıllarda birçok akademik çalışma yapılmaktadır. Mevcut IoT protokolleri üzerine yapılan çalışmalarda, bu protokollerin birbirlerine göre farklı alanlarda üstünlükler sağlayabildiği görülmektedir. Bu bölümde, IoT uygulama protokollerinin bant genişliği, gecikme yönünden başarım karşılaştırmalarına yönelik çalışmalar ve ortak çalışabilirlik yönünden yapılan araştırmalar ele alınmıştır.

MQTT ve CoAP protokollerinin birlikte çalışabilmesini sağlamak amacıyla ara katmanın geliştirildiği Thangavel ve arkadaşlarının çalışmasında (Thangavel vd., 2014) MQTT ve CoAP protokolleri uçtan uca gecikme ve bant genişliği tüketimi yönünden ele alınmıştır. Bu çalışmaya göre, %20'den daha düşük paket kayıplarının yaşandığı ortamlarda MQTT'nin; yüksek paket kayıplarında ise CoAP'ın daha yüksek başarım gösterdiği ifade edilmektedir. Paket kayıplarının olmadığı varsayımının yapıldığı ideal bir hücresel ağ ortamında başarım değerlendirilmesinin yapıldığı çalışmada (Lars, 2015) ise protokollerin ek yüklerine göre karşılaştırmaları yapılmakta ve 1024 byte'dan büyük ek yük durumlarında CoAP protokolünün performans kayıpları yaşadığı ortaya konmaktadır. Mobil ve stabil olmayan ağlarda yapılan çalışmada (Luzuriaga vd., 2015) MQTT ile AMQP protokolleri bant genişliği kullanımı, gecikme ve jitter etkisi yönlerinden karşılaştırılmakta, her iki protokolde de bu açılardan anlamlı bir farklılığın olmadığı ancak güvenlik açısından bakıldığında AMQP'nin; enerji verimliliği açısından ise MQTT'nin daha başarılı olduğu belirtilmektedir. Aynı verinin taşınmasında CoAP ile MQTT protokolünün karşılaştırıldığı diğer bir çalışmada (Shin vd., 2017) CoAP protokolünün daha verimli olduğu ifade edilmektedir. Yüksek ağ trafiğinin bulunduğu bir ortamda gerçekleştirilen çalışmada, MQTT protokolünün CoAP'a göre daha başarılı olduğu, daha yüksek net genişlik değerine ve daha düşük gecikmeye sahip olduğu belirtilmektedir (Collina vd., 2014).

Nesnelerin İnternetinde servis keşif problemini ele alan bir çalışmada, M.Kirsche ve arkadaşları, MQTT ve CoAP protokollerinin uçtan-uca iletişim sağlayamadığı, mesajların dönüştürülmesinden kaynaklanan gecikmelerin

yaşandığı ve keşif sürecinin karmaşık yapıda olduğu belirtmektedirler. Bu nedenle mDNS ve DNS-SD kombinasyonunu kullanan, sadeleştirilmiş bir yapı geliştirmişlerdir (Klauck & Kirsche, 2012) (Jara vd., 2012).
IoT Protokollerinin geliştirilmesi başlığı altında incelenebilecek (Choi vd., 2017) bir çalışmada MQTT ve CoAP protokollerinin nesnelerin interneti uygulamalarında yaygın kullanıldığına değinilmekte, ancak her iki protokolün de çok sayıda sensörün bulunduğu büyük ağlarda ölçeklenebilirlik problemleri ile karşı karşıya olduğu belirtilmektedir. İlgili çalışmada ölçeklenebilirliği sağlamak ve bant genişliğini verimli kullanmak amacıyla CoAP protokolünde iyileştirmeye gidilmiştir. Buna göre, sensörlerin bir küme oluşturacak şekilde gruplandırılması ve kümenin bir temsilci ile veri gönderilmesi sağlanmakta, sonuç olarak %18 daha az bant genişliği tüketimine ulaşıldığı ifade edilmektedir.

IoT sisteminin karmaşıklaşmaya başlaması, ağa bağlanan cihazların sayı ve türlerindeki artış, çözüm olarak hibrit yöntemlerin ele alınmasına neden olmaktadır. Protokollerin birlikte kullanımına yönelik olarak Bellavista ve arkadaşlarının çalışmasında (Bellavista & Zanni, 2016) CoAP ve MQTT'nin birlikte kullanımını hedefleyen bir mimari tasarlanmıştır. Çalışma, yüksek yoğunluklu ağlarda geliştirilen bu mimarinin ölçeklenebilirliği sağladığını ifade etmektedir.

Merkezi bir ara katman kullanılarak birden çok protokolün birlikte çalışmasını sağlamak amacıyla yapılan çalışma önemli boyutta performans kayıpları olmadan protokollerin birlikte çalışabilirliğini ortaya koymaktadır (Huo, 2014). "*Custom UDP*" olarak adlandırılan bir uygulama protokolü tasarımında ise önerilen protokol çeşitli IoT protokolleri ile karşılaştırılmış ve deneysel sonuçlara göre, tahmin edilemeyen paket kayıplarının bulunduğu ortamlarda göreli olarak düşük enerji ve bant genişliği tüketimi sunduğu ifade edilmektedir (Chen & Kunz, 2016).

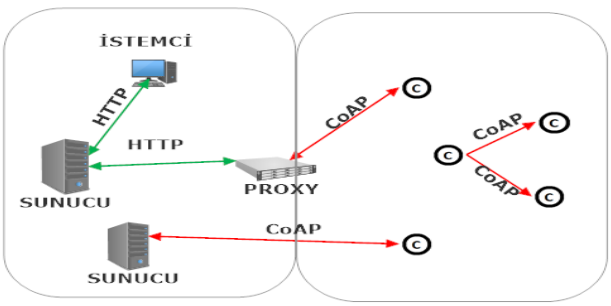
Görüldüğü üzere Nesnelerin İnternetinde belirli bir protokolün diğerine göre her alanda bir üstünlüğü olduğunu ifade etmek çok güçtür. Kullanılan ağ topolojisi, güvenlik ihtiyacı, ölçeklenebilirlik, bant genişliği kullanımına göre farklı protokoller kullanılabilir. Ayrıca ara katman yazılımlarının çeşitliliği, IoT cihazlarına bağlantı kurulmasını ve toplanan verilerin yorumlanmasını güçleştirmektedir (Ngu, Gutierrez, Metsis, & Sheng, 2017). IoT ekosisteminin heterojen yapısı, giderek artan oranda bir büyüme eğilimi farklı protokol ihtiyacını da beraberinde getirmektedir.

3. IoT Uygulama Katmanı Protokolleri (IoT Application Layer Protocols)

IoT iletişimin başarımını belirleyen en önemli faktörlerden biri kullanılacak olan mesajlaşma protokolünün belirlenmesidir. Organizasyonlar için en uygun protokolün belirlenebilmesi, bu protokollerin özelliklerinin ve gereksinimlerinin anlaşılmasına bağlıdır (Naik, 2017). IoT protokolleri ile ilgili akademik çalışmalara konu olan ve yaygın olarak kullanılan protokoller özellikleri açısından incelendiğinde uygulama katmanı protokolleri, servis keşif protokolleri ve altyapı protokolleri olmak üzere üç kategori altında toplandığı görünmektedir (Florea vd., 2017). Bu bölümde uygulama katmanı protokollerinden yaygın olarak kullanılan MQTT ve CoAP ele alınmıştır.

3.1. Constrained Application Protocol (CoAP)

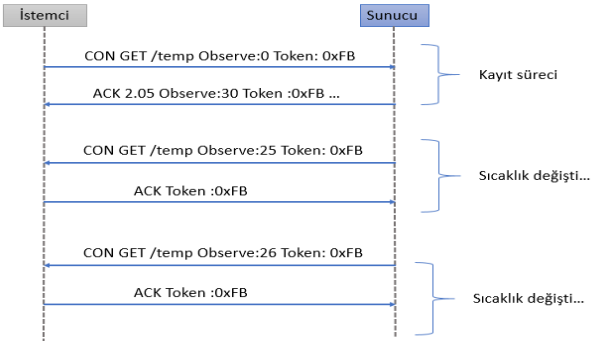
İnternet Mühendisliği Görev Gücü (IETF) tarafından tasarlanan Kullanıcı Datagram Protokolü (User Datagram Protokolü-UDP) tabanlı bu protokol (Lars, 2015), nesnelerin internetinin bir türü olan Nesnelerin Web'inde makineler arası iletişimde (M2M) kullanılmak üzere tasarlanmıştır (Florea vd., 2017; Naik, 2017; Shin vd., 2017). CoAP yapısında her bir nesne, basit bir sunucu görevi görmektedir. Kısıtlı düğümler arasında ve kısıtlı ağlarda iletişim kurmak için sunucu – istemci mimarisinde, istek / cevap mesajları ile haberleşen bir yapıda geliştirilmiştir. CoAP protokolünde cihazlar ve bu cihazların rolleri Şekil 1'de gösterilmiştir.



Şekil 1. CoAP protokolünde cihazlar ve roller (Devices and their roles in CoAP)

İstemciler sunucuda verilen hizmetlere GET, PUT, POST ve DELETE yöntemleri ile erişebilmektedir (Gündoğan vd., 2018). Uç noktalar arasındaki iletişimde kullanılan CoAP iletilerinde, aşağıdaki gibi kaynak cihaz erişim bilgilerini içeren URI yapısı kullanılmaktadır (Shelby vd., 2014; Ishaq vd., 2013):
GET coap :// [<IoT cihaz IP adresi>]:[<port numarası>]/[kaynak URI adresi]

CoAP protokolün en önemli avantajlarından biri, http protokolüne kolay entegre olabilme yeteneğidir (Naik, 2017). İnternet ortamında çoğu uygulama http protokolünü kullandığından, CoAP mesajlarını http mesajlarına dönüştüren bir vekil sunucu kullanılabilmektedir (M. Koster & Keranen, 2015).



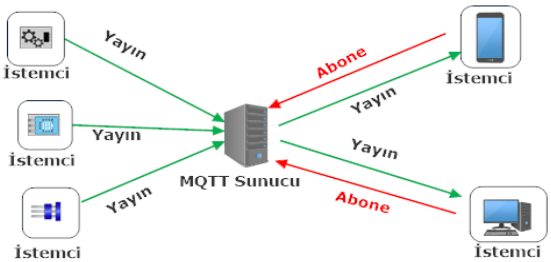
Şekil 2. CoAP iletişiminin adımları (Communication steps of CoAP)

CoAP protokolünde veri kaynağında her değişiklik olduğunda istemciye bilgi gönderilir. Böylece ortam değişikliği durumunun istemciye bildirilmesi sağlanır. CoAP protokolünün bu işlem adımları örnek olarak Şekil 2'de gösterilmiştir. CoAP protokolünde, onay gerektiren, onay gerektirmeyen, onay ve sıfırlama olmak üzere dört tür paket bulunmaktadır (Thangavel vd., 2014). UDP protokolü kullanıldığından, mesajların hedefe ulaşıp ulaşmadığının doğruluğu, onay gerektiren paketler ile sağlanmaktadır.

3.2. Message Queuing Telemetry Transport (MQTT)

M2M iletişimi için tasarlanan, sunucu istemci mimarisi ile çalışan IBM kökenli bir protokol olup abonelik ve yayın mesajları yapısında kullanılmaktadır (Chen & Kunz, 2016). Kısıtlı bant genişliği sunan konumlar arasında veri transferi sağlamak amacıyla geliştirilmiş İletim Denetim Protokolü (Transmission Control Protocol – TCP) tabanlı bir protokoldür (Anjum vd., 2014). Ağda bulunan birçok cihazdan veri toplamak ve verileri ihtiyaç duyan cihazlara ya da yazılımlara sunmak üzere tasarlanmıştır.

Bu uygulama katmanı protokolünde abone, yayıncı ve sunucu olmak üzere üç rol bulunmaktadır (Schmitt, Carlier, & Renault, 2018). İstemci rolünde bulunan abone ile yayıncı arasındaki tüm iletişimin koordinesi sunucunun sorumluluğundadır (Pathania, 2017). Yayıncı bir "konu" (topic) ile ilişkilendirilmiş bir mesaj gönderdiğinde, sunucusu mesajı alır, mesajın konusuna abone olan kullanıcıyı sorgular ve mesajı iletir (Tantitharanukul vd., 2016). MQTT protokolünün çalışma sistemi Şekil 3'te gösterilmiştir.



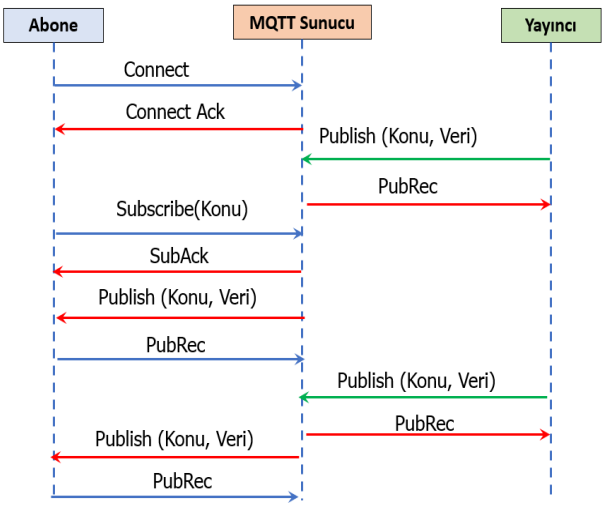
Şekil 3. MQTT protokolünde cihazlar ve cihazların görevleri (Devices and their roles in MQTT protocol)

Abonelik sisteminde kullanılan "konu" kavramı (Sasaki vd., 2018), hiyerarşik bir yapıda konum ve hizmet bilgisini içeren aşağıdaki gibi bir diziden oluşmaktadır (Menyah, 2017).

BinaA/Kat3/DerslikB/SICAKLIK

Bu örnekte A binası 3.katındaki, B dersliğinin sıcaklık bilgisi "konu" olarak ifade edilmiştir.

MQTT iletişiminde kullanılan mesajların paket başlığı küçük boyutlu olup 14 farklı paket türü kullanılmaktadır (Kimsey vd., 2015). İstemci ile sunucu arasındaki paketlerin akışı Şekil 4'te gösterilmiştir.



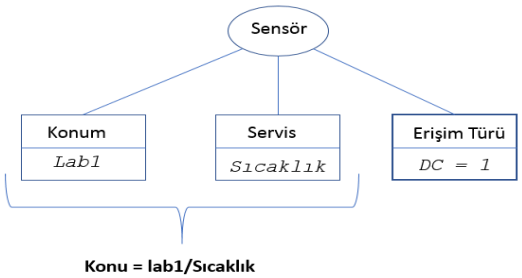
Şekil 4. MQTT paketlerinin akışları (Packet flows of MQTT)

MQTT yapısında mesajın ulaşıp ulaşmadığının garantisinin düzeyini tanımlamak üzere üç düzey hizmet kalitesi (QoS) bulunmaktadır (Sch vd., 2017). QoS0 yapısında ileti sadece gönderilir, mesajın alındığına ilişkin geri bildirim bulunmamaktadır. QoS1 düzeyinde ise mesajın alındığına dair abone cihazdan yayıncıya PUBACK geri bildirim mesajı gönderilir. MQTT'nin en güvenli iletim şekli QoS2 düzeyindedir. Buna göre yayıncı, QoS2 düzeyinde mesajına sahip olduğunu bildiren bir ileti (PUBLISH) gönderir. Abone alıcı, iletiyi almaya hazır olduğunu gösteren PUBREC mesajı göndermektedir. Yayıncı PUBREL mesajını, aboneye iletir. Alıcı bu mesajı aldığında, işlemi PUBCOMP mesajı ile onaylamaktadır (Gündoğan vd., 2018).

IoT ekosisteminde ağa bağlı cihazların çeşitli ve farklı gereksinimleri olduğundan, tek bir protokolün baskın bir şekilde kullanılması olası değildir. Hangi protokolün daha uygun olacağı, sistemde kullanılan uygulamalara ve kullanıcı gereksinimlerine bağlıdır.

4. Geliştirilen Uygulama Katmanı Protokolü (Developed Application Layer Protocol)

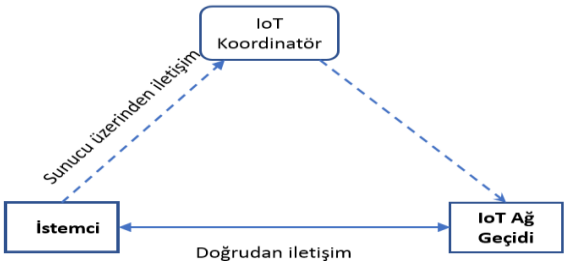
Nesnelerin İnterneti kapsamında son yıllarda güvenlik, kaynak keşfi, ortak çalışabilirlik, uyumluluk ve performans geliştirmeleri gibi problem durumlarına yönelik çeşitli araştırmalar ve akademik çalışmalar yapılmaktadır. Bu çalışma kapsamında bu problem durumlarının tümüne odaklanılmamış ve bazı varsayımlar dikkate alınmıştır. Buna göre, her ne kadar güvenlik kavramı IoT için kritik bir öneme sahip olsa da bu çalışmada ortamın güvenli olduğu varsayılmış ve veri güvenliği kapsam dışında tutulmuştur. Sensörlerin kayıt ettirilmesi ve kaynak keşfi sürecinde, MQTT ve CoAP protokollerine uyum sağlamak amacıyla sensöre ait konum ve hizmet bilgisi dikkate alınmış, sensör üreticisi, sensör tipi gibi parametreler dikkate alınmamıştır. Sensörden sunulan hizmetin belirlenmesinde konum ve servis bilgisi, MQTT yapısında kullanılan "*konu*" ile uyumluluk sunmak amacıyla Şekil 5'de gösterildiği gibi tasarlanmıştır.



Şekil 5. Geliştirilen protokolde kullanılan sensör özellikleri (Sensor features used in the developed protocol)

Bu çalışmada sensörlerden elde edilen verilerin tek bir veri paketi içinde taşınabildiği, IP paketlerinin parçalara ayrılmadığı varsayılmıştır.

IoT sisteminde kullanılan cihazlar kısıtlı kaynaklara sahip olduklarından, yüksek işlem gücü, bellek ve bant genişliği tüketimi gerektiren uygulamalar, ihtiyacı karşılamada yetersiz ve verimsiz olmaktadır. Bu nedenle geliştirilen protokol, düşük kaynaklarda çalışabilecek esneklikte ve basit yapıda olacak şekilde tasarlanmıştır.

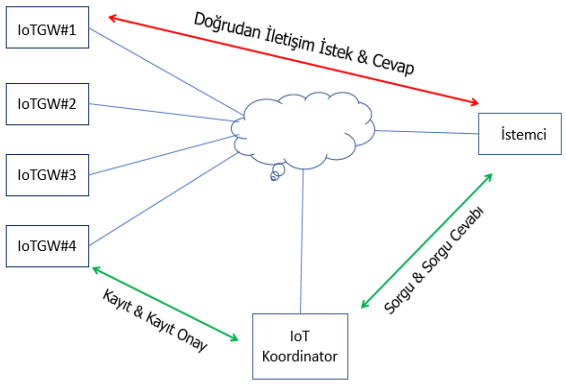


Şekil 6. Hibrit Uygulama Protokolü doğrudan iletişim ve sunucu tabanlı iletişimi destekler (Hybrid Application Protocol supports both direct and server based communication)

Çalışma kapsamında hem merkezi sunucu ile çalışan hem de doğrudan erişim destekleyen, Şekil 6'da gösterildiği gibi hibrit bir uygulama katmanı protokolü (hIoT) geliştirilmiştir. Sensör türüne göre, doğrudan erişim ya da sunucu aracılığı ile erişim yöntemlerinden biri seçilebilir. Sunucu devre dışı kaldığında ise, doğrudan erişim yöntemine geçilebilen dinamik bir yapı tasarlanmıştır. Böylece sunucu tabanlı çalışan protokollerde yaşanan tek bir hata noktasının önüne geçilmesi amaçlanmıştır.

4.1. Genel Çalışma Prensibi (General Working Principle)

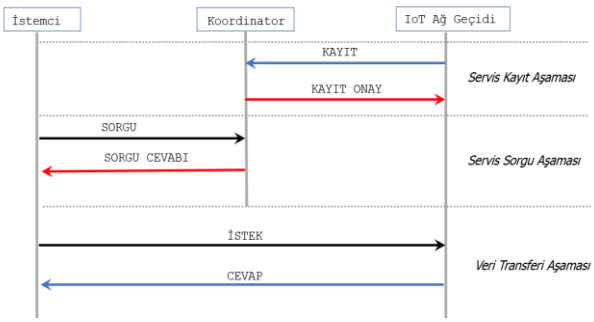
Tasarlanan sistemin doğru bir şekilde çalışabilmesi için, Nesnelerin İnterneti ekosisteminde cihazların rollerinin kesin bir şekilde tanımlanması gerekmektedir. Belirlenen roller sayesinde ağ iletişiminin modeli tanımlanabilmektedir.



Şekil 7. Geliştirilen protokolde cihazlar ve iletişim paketleri (Devices and communication packets in developed protocol)

Geliştirilen protokolde Şekil 7'de gösterildiği gibi, IoT Koordinatör, IoT Ağ Geçidi (IoTGW) ve İstemci olmak üzere üç farklı bileşen tanımlanmıştır. Bu bileşenlerin görevleri 4.6 başlığında ayrıntılı olarak ele alınmıştır.

Protokolün çalışması Şekil 8'de gösterildiği gibi, Servis Kayıt Aşaması, Servis Keşif Aşaması ve Veri Transfer Aşaması olmak üzere üç aşamada incelenmektedir.



Şekil 8. Geliştirilen protokolün temel aşamaları (The main phases of the developed protocol)

Servis kayıt aşaması, IoT Ağ Geçidi üzerinde çalışan servislerin veri tabanına kayıt ettirilmesi aşamasıdır. ***Servis keşif aşamasında***, istemcilerin ilgilendikleri servislere nasıl erişileceği sorgulanır. Koordinatör tarafından servise nasıl erişeceği bilgisi istemciye cevap olarak gönderilir. ***Veri transferi aşamasında***, koordinatör tarafından verilen erişim bilgisine bağlı olarak, istemci ile ağ geçidi arasında doğrudan iletişim ya da sunucu aracılığı ile iletişim gerçekleştirilir.

4.2. Paket Türleri (Packet Types)

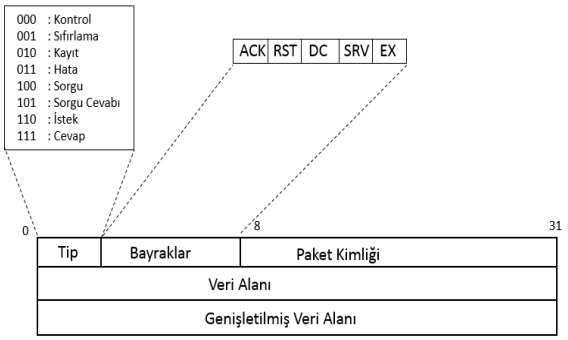
Tasarlanan IoT protokolü sekiz farklı paket içermektedir. Paket tipleri ve işlevleri Tablo 1'de gösterilmiştir.

Tablo 1. Paket Tipleri ve Görevleri (Packet types and their functions)

Paket Tipi	Açıklama
Kontrol (Control)	Erişilebilirlik kontrolü amacıyla kullanılır.
Sıfırlama (Reset)	Erişim yöntemini sıfırlar.
Kayıt (Register)	IoTGW tarafında sağlanan servisin sunucuya kayıt ettirilmesini sağlar
Hata (Error)	Hata durumunda gönderilir.
Sorgu (Query)	İstemcinin servis keşfi için yaptığı sorgu paketidir.
Sorgu Cevabı (Reply)	Servis keşfi paketine cevap olarak gönderilir.
İstek (Request)	IoTGW tarafından sunulan servisten bilgi isteme paketidir.
Cevap (Response)	İstek paketine cevap olarak gönderilen ve veri içeren pakettir.

4.3. Paket Yapısı ve Başlık Bilgisi (Packet Structure and Packet Header)

Geliştirilen protokolün başlık bilgisi, kolay kullanım sağlamak amacıyla, Şekil 9'da gösterilen alanlardan oluşan bir yapıda tasarlanmıştır.



Şekil 9. Hibrit IoT Protokolünün paket başlığı yapısı (Packet header of the Hybrid IoT Protocol)

Toplamda 32-bit olan başlık bilgisindeki 3-bitlik "Tip" alanı, paket türünü göstermektedir. 5-bitlik "*Bayraklar*" alanı iletişimin sağlıklı çalışması için gerekli olan bayrakları saklar. 24-bitlik "*Paket Kimliği*" alanı paketin benzersiz kimlik bilgisini içermektedir.

Paket içerisinde yer alan "*Veri Alanı*" ise, uygulamalarda sensörlerden elde edilen verilerin taşınmasını sağlayan 32-byte uzunluğundaki alandır. "*Genişletilmiş Veri Alanı*" ise daha büyük boyutlarda veri transferini desteklemek amacıyla geliştirilmiş ve "EX" bayrağına bağlı olarak çalışan, 1024 baytlık ek veri taşıma alanını ifade etmektedir.

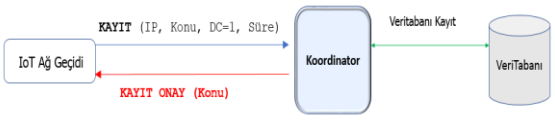
ACK bayrağı, iletişimdeki onay paketlerinde kullanılmaktadır. RST bayrağı, *Sıfırlama* paketi ile birlikte kullanılan, IoT Ağ Geçidi aracılığıyla doğrudan erişim yetkisini sıfırlamak amacıyla kullanılmaktadır. RST bayrağı set edilmiş paketi alan IoT Ağ Geçidi, doğrudan erişim yetkisini kaybedecektir. Böylece servisten bilgi almak için koordinatör sunucusu aracı olarak kullanılacaktır. DC bayrağı, "1" ise ilgili servise doğrudan erişim sağlanabilir. SRV bayrağı, sunucu kaynaklı trafikleri gösterir. Son bayrak olan EX ise Genişletilmiş Veri Alanının kullanılıp kullanılmayacağını gösterir.

4.4. Protokolün Çalışma Aşamaları (Working Phases of the Protocol)

4.4.1. Servis Kayıt Aşaması (Service Registration Phase)

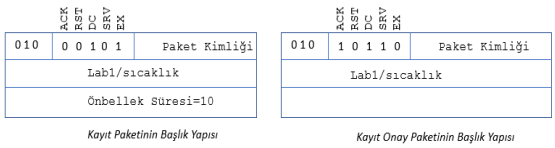
Bu aşamanın amacı, IoT Ağ Geçitleri tarafından sunulan servislerin ve bu servislere erişim bilgilerinin tutulduğu yerel veri tabanının oluşturulmasıdır. Sisteme kayıt edilmesi gereken sensör sayısı az olduğu durumlarda, elle kayıt yöntemi ya da yarı-otomatik kayıt sistemi kullanılabilir. Ancak bu sayı çok daha fazla olduğu durumlarda, elle kayıt yöntemi verimsiz, pahalı ve iş gücü gerektirecek; çoğu durumda uygulanması imkânsız olacaktır (Kang vd., 2017). Bu nedenle çalışma kapsamında IoT Ağ Geçidine bağlı sensörler tarafından sağlanan hizmetlerin yarı otomatik kayıt edilmesini sağlayan bir mekanizma geliştirilmiştir. Sensör aracılığıyla sunulan sıcaklık, ışık miktarı, nem gibi servisler, konum bilgisi ile birleştirilip IoT Koordinatör yerel veri tabanına kaydettirilir.

IoT Ağ Geçidine bağlı sensörün yerel Koordinatör cihazına otomatik kaydedilmesi bu aşamada gerçekleşmektedir. Bu aşamada IoT Ağ Geçidi cihazından sağlanan hizmet, IP adresi, uçtan-uca erişim destekleyip desteklemediği (DC) bilgisi ve önbellekleme süresi koordinatör cihaza gönderilmektedir.



Şekil 10. Geliştirilen protokolün servis kayıt süreci (Service registration process of the developed protocol)

Lab1 konumunun Sıcaklık servisi için örnek *Kayıt* paketinin yapısı ve bu pakete verilen *Kayıt Onay* cevabının yapısı Şekil 10'da gösterilmiştir.

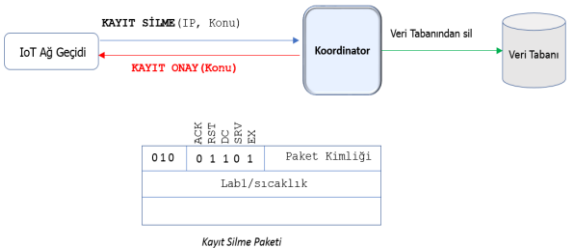


Şekil 11. Kayıt ve Kayıt Onay paketlerinin başlık yapıları (Header of Registration and Registration Ack packets)

Şekil 11'deki örneğe göre, "*Lab1/Sıcaklık*" servisi, doğrudan erişimi desteklemekte ve bu servis tarafından gönderilen bilgi koordinatör sunucunun ön belleğinde 10 saniye boyunca tutulmaktadır. Bu süre içerisinde aynı servis için gelen tüm veri talepleri, tekrar sensör düğümüne sorgulanmadan doğrudan koordinatör önbelleğinden alınarak istemciye gönderilir.

Konu, önbellek süresi ve DC bilgisini alan sunucu, bu bilgileri ve IoT Ağ Geçidinin IP bilgisini veri tabanına kaydeder. Kayıt işlemimin ardından, koordinatör tarafından IoT Ağ Geçidinde *Kayıt Onay* paketi gönderilir.

Geliştirilen protokolün servis kayıt sürecinde tüm bilgiler Koordinatörde yer alan veri tabanında tutulmaktadır. Bu durum tek bir hata noktasının oluşmasına neden olduğundan, koordinatörün devre dışı kalması durumunda servis keşif sorgularına cevap verilemeyecektir. Bu problemin önüne geçilebilmesi için IoT Ağ Geçidi cihazlarda *Otomatik Kayıt* özelliği geliştirilmiştir. Buna göre, kayıt paketine belirli bir zaman diliminde cevap verilmediği takdirde, IoT Ağ Geçidi, servis keşif isteklerine doğrudan cevap verecek şekilde kendisini ayarlamaktadır.

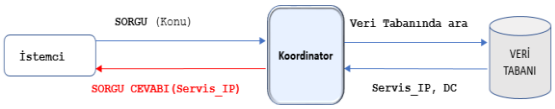


Şekil 12. Kayıt silme süreci ve paket yapısı (De-register process and its packet)

IoT Ağ Geçidi tarafından sunulan servisin devre dışı kalması durumunda, kayıt işleminin iptal edilmesi gerekir. Bu durumda ağ geçidi, ilgili servise ilişkin RST biti set edilmiş özel *Kayıt* paketi gönderir. Paketi alan Koordinatör, bu servise ait erişim bilgilerini veri tabanından siler ve IoT Ağ Geçidine *Kayıt Onay* paketi ile cevap verir. Şekil 12'deki örnekte "lab1/Sıcaklık" servisine ilişkin kayıt silme paketinin yapısı gösterilmektedir.

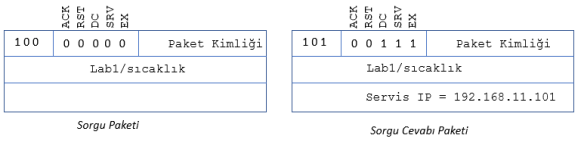
4.4.2. Servis Keşif Aşaması (Service Query Phase)

IoT'nin başarılı bir şekilde uygulanmasına yönelik potansiyelinden tam olarak yararlanmak için, mevcut kaynakların sorunsuz ve otomatik olarak keşfedilmesine ve IoT cihazının erişim bilgisinin dinamik olarak bulunmasına ihtiyaç vardır. MQTT yapısında servis keşif özelliği olmamasına karşın, CoAP protokolünde servis keşfi için Tekdüzen Kaynak Tanımlayıcıları (Uniform Resource Identifiers-URI) yapısı kullanılmaktadır (Kim vd., 2016). Önerilen protokolde, istemci erişmek istediği servise ilişkin "*konu*" bilgisini "*Sorgu*" paketleri ile sorgular. İstemci ile sunucu arasındaki iletişimde, istemci tarafından ihtiyaç duyulan konuma ve servise ilişkin sorgu Şekil 13'de gösterilmiştir.



Şekil 13. Servis keşif sürecinin yapısı (Structure of the service query process)

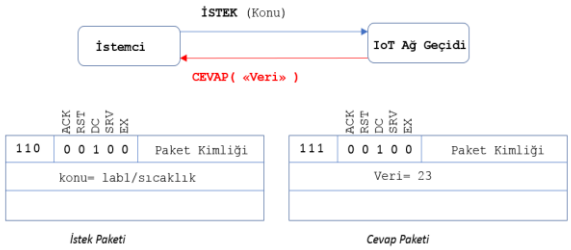
İstemci tarafından sorgulanan konuya ilişkin veri, koordinatör veri tabanında sorgulanır ve servisin hangi IP tarafından sağlandığı istemciye bildirilir. İlgili servis doğrudan iletişimi destekliyorsa IoT Ağ Geçidinin IP adresi; doğruda iletişim desteklenmiyor ise Koordinatörün IP adresi cevap olarak gönderilir. Şekil 14'deki örnekte istemci tarafından gönderilen "*Sorgu*" ve Koordinatör tarafından cevap olarak verilen "*Sorgu Cevabı*" paketlerinin başlık yapısı gösterilmiştir.



Şekil 14. Sorgu paketi ve sorgu cevap paketinin yapısı (Query and reply packet structure)

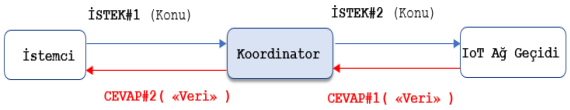
4.4.3. Veri transferi aşaması (Data Transfer Phase)

Bu aşamada "konu" bilgisine göre veri talebi gerçekleşmektedir. Önerilen protokolün hibrit yapısı gereği, veri talebi doğrudan sensör düğümü olan Ağ Geçidinden sağlanabildiği gibi (Şekil 15), Koordinatör sunucu aracılığı ile de (Şekil 16) gerçekleşebilmektedir.



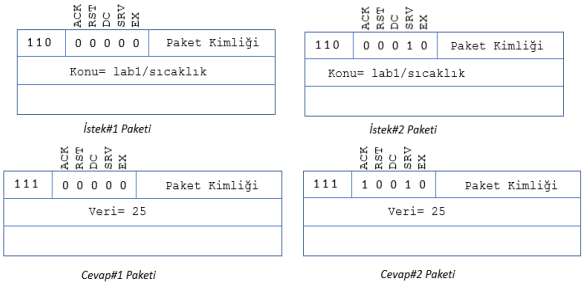
Şekil 15. Sensör düğümüne doğrudan erişim süreci ve kullanılan paketler (Process of direct access to the sensor node and related packets)

Hangi erişim yönteminin desteklendiği, *Servis Kayıt* aşamasında belirlenmektedir. Doğrudan IoTGW erişimi için paket yapısı ve örnek içeriği Şekil 15'de verilmiştir. İstemcinin gönderdiği *İstek* mesajına, içerisinde sorgulanan servise ilişkin veri barındıran *Cevap* paketi ile geri bildirimde bulunulur. Hibrit protokolün desteklediği diğer bir yöntem olan sunucu tabanlı iletişimin modellemesi ise Şekil 16'da gösterilmiştir.



Şekil 16. Sunucu aracılığı ile veri transferi süreci (Data transfer process via server)

Sunucu tabanlı iletişimde, istemcinin doğrudan IoT Ağ Geçidine ulaşmasına izin verilmemektedir. Bu nedenle servisten veri talebi koordinatör aracılığı ile yapılır. Koordinatör, "*konu*" bilgisine göre istemciden gelen *İstek* paketini alır ve IoT Ağ Geçidine gönderir. IoT Ağ Geçidinden gelen *Cevap* paketi de yine koordinatör aracılığıyla istemciye gönderilmektedir. Sunucu tabanlı iletişimde sensör düğümü yalnızca paket başlığında "*SRV*" bayrağı "0" olarak işaretlenmiş paketleri dikkate almaktadır.



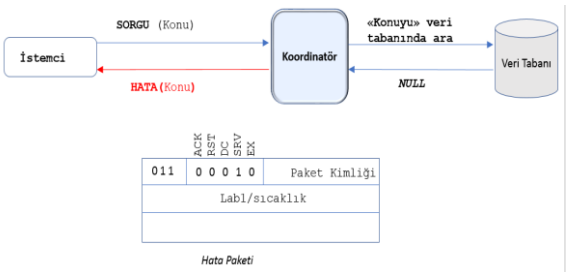
Şekil 17. Sunucu tabanlı iletişimde kullanılan istek ve cevap paketleri (Request and response packets used in server based communication)

Sunucu tabanlı iletişimlerde kullanılan paketlerin başlık bilgisi ve örnek içeriği Sekil 17'de verilmiştir. Koordinatör sunucu aracılığı ile iletişimde, istemci bir servise erişmek istediğinde sunucuya bağlanmak zorundadır. Bu servis ile ilgili tüm iletişim koordinatör aracılığıyla gerçekleşmektedir. Servis kayıt sürecinde belirlenen ön bellek süresine göre, verinin ne kadar süre ile geçerli olacağını belirlenmiştir. Koordinatör önbellek süresi boyunca veriyi saklamaktadır. Bu süre içerisinde aynı servis için gelen diğer *İstek* mesajları, sensör düğümüne gönderilmeden doğrudan ön bellekten alınarak cevaplandırılmaktadır.

4.5. Yardımcı Paketler (Utility Packets)

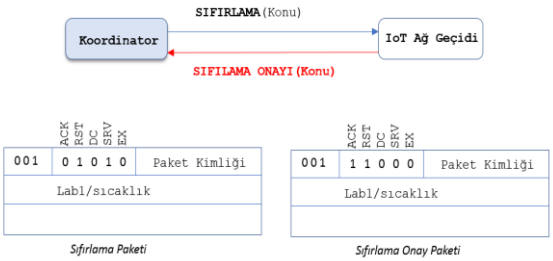
Ana işlevselliği sunan beş paketin yanında işlevselliği arttırmak amacıyla *Hata*, *Sıfırlama* ve *Kontrol* paket türleri kullanılmaktadır.

Hata Paketi (Error packet); Sunucuda yer alan yerel kaynak veri tabanında sorgulanan *konuya* ilişkin bir kayıt bulunmadığı, servise erişim olmaması, veri tabanına kayıt yapılmadı gibi hata durumlarında istemciye gönderilmektedir. Her bir hataya ilişkin "hata kodları" yine paket içerisinde gönderilir. Hata paketinin örnek işleyişi ve paket başlık yapısı Şekil 18'de verilmiştir.



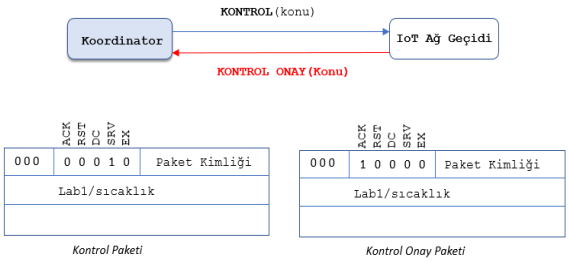
Şekil 18. Hata süreci ve kullanılan paket yapısı (Error process and its packet structure)

Sıfırlama Paketi (Reset Packet): IoT Ağ Geçidi cihazının hangi erişim türünü desteklediği, Servis Kayıt aşamasında belirlenmekte ve bu değer sürekli olarak sabit kalmaktadır. Ancak ağın anlık durumuna göre doğrudan erişim yönteminin iptal edilmesi gerekebilir. Bu durumda koordinatör tarafından, erişim yönetimin değiştirilmesi için *Sıfırlama* paketi gönderilmektedir. *Sıfırlama* paketinin iletişim şekli ve paket yapısı Şekil 19'da gösterilmiştir. Bu paketi alan IoT Ağ Geçidi, doğrudan erişim yöntemini kaybetmektedir.



Şekil 19. Sensör düğümünün erişim yönteminin değiştirilmesi (sıfırlama) süreci ve paket yapısı (The process of changing (reset) the access method of the sensor node and the its packet structure)

Kontrol Paketi (Control Packet): IoT Ağ Geçidi üzerinde yer alan servisin erişilebilir olup olmadığını kontrol etmek için kullanılan doğrulama mesajlarıdır. Şekil 20'de paket yapısı gösterilen kontrol paketlerine kontrol onayı mesajları ile cevap verilmektedir.



Şekil 20. Kontrol paketi ve başlık yapısı (Control packet and its packet header)

Kontrol mesajlarının diğer bir avantajı da gönderilen mesaja verilen onay mesajları aracılığıyla servise erişim süresinin de hesaplanabilmesidir. Kontrol mesajlarına cevap alınmadığı takdirde, ilgili servisin devre dışı kaldığı varsayılır ve koordinatörün veri tabanından silinir. Aynı şekilde, Kontrol mesajları IoT Ağ Geçitlerine ulaşmadığında koordinatörün devre dışı kaldığı varsayılır ve sensör düğümleri doğrudan erişim durumuna geçmektedir.

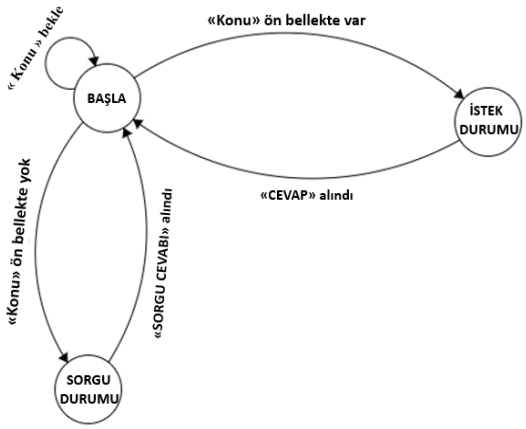
4.6. Geliştirilen Protokoldeki Bileşenler (Components of the Developed Protocol)

Daha önceki bölümde değinildiği gibi, protokolün önerilen mimarisinde Koordinatör, İstemci yazılımı ve IoT Ağ Geçidi düğümü olmak üzere üç bileşen bulunmaktadır. Bu bölümde bu bileşenlerin çalışma prensipleri akış diyagramları ile ifade edilmektedir.

4.6.1. İstemci Yazılımı (Client Software)

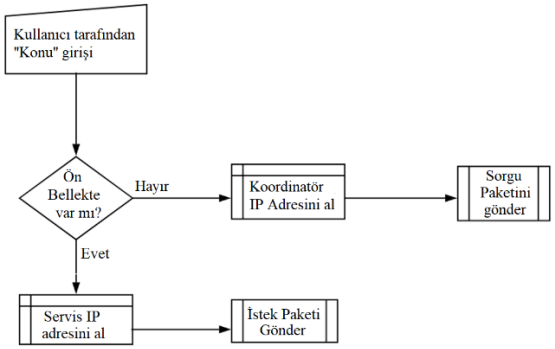
Ağdaki bir konuma ve servise ilişkin veri talep eden, sensör ile etkileşim kurmak isteyen uygulamadır. MQTT ve AMQP yapısındaki herhangi bir konuya abone olan istemciye benzetilebilir. MQTT yapısında, trafiği başlatan taraf yayıncıdır ve genellikle abonenin o an bilgiye ihtiyacı olup olmamasına bakılmaksızın sensöre ilişkin veri aboneye gönderilir. Ancak, önerilen mimaride trafiği başlatan taraf, istemci tarafıdır. Veriye ihtiyaç duyulduğu an veri talebi, istemci tarafından başlatılmaktadır.

İstemci "konu" ile belirlenen servise nasıl erişebileceğine ilişkin sorguyu yapar. Koordinatörden gelen *Sorgu Cevabında* servis kaynağının erişim bilgisi bulunmaktadır. İstemci "konu" ile erişim bilgilerini eşleştirir ve ön bellekte saklar. Daha sonra bu servise ilişkin veri talebi yapılacağı zaman tekrar Sorgu yapılmasına gerek kalmaksızın doğrudan veri isteme aşamasına geçmekte ve sensör düğümünden cevabının gelmesini beklemektedir. Şekil 21'de istemci yazılımının akış diyagramı gösterilmiştir.



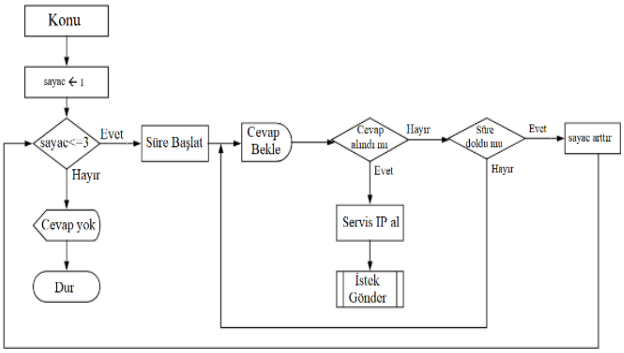
Şekil 21. İstemci yazılımı akış diyagramı (Client software flow diagram)

İstemci yazılımı, veri talep edilen "konu" bilgisinin ön bellekte bulunup bulunmama ve koordinatörden *Sorgu cevabı* paketinin gelip gelmemesi durumuna göre, *sorgu* durumuna veya *istek* durumuna geçmektedir. Kullanıcı tarafından sorgulanmak istenen "konu", öncelikli olarak ön bellekte aranır. Ön bellekte sorgulanan konu bilgisine ilişkin bir kayıt varsa, servisi veren donanım aygıtının IP adres bilgisi ön bellekten alınmaktadır. Bu durumda veri talebi aşaması olan *İstek* durumuna geçilmektedir. Ancak ön bellekte istenen konu için kayıt yoksa, hizmeti sunan IoT Ağ Geçidinin IP adresini sorgulaması gerekmekte ve *sorgu* durumuna geçmektedir.



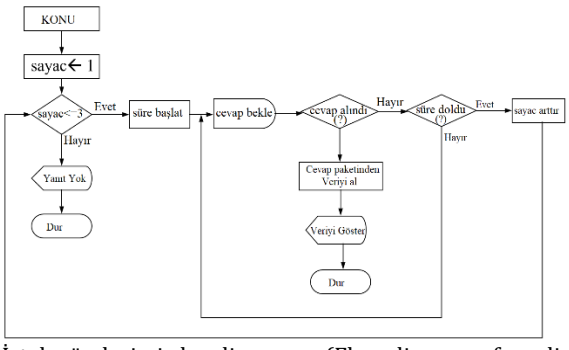
Şekil 22. İstemci yazılımı başlangıç durumu akış diyagramı (Initial state flow diagram of client software)

İstemci yazılımının başlangıç durumuna ilişkin akış diyagramı Şekil 22'de verilmiştir. Diyagramda yer alan *Sorgu Paketini Gönder* ve *İstek Paketi Gönder* süreçleri modülerlik sağlamak amacıyla ayrı akış diyagramları olarak sunulmuştur.



Şekil 23. Sorgu gönderimi akış diyagramı (Flow diagram of sending query)

"Konu" bilgisini göre sorgulama süreci, istemciden gönderilen *Sorgu* paketi tarafından yapılmaktadır. Bu sürecin akış diyagramı Şekil 23'te gösterilmiştir. Sorguya verilen *Sorgu Cevabı* mesajında *Servis IP* adres bilgisi bulunmaktadır. Bu bilgiler, daha sonra tekrar kullanılmak amacıyla istemcinin ön belleğinde tutulmaktadır. Akış diyagramından da görüleceği üzere, sorgulama için belirli bir süre beklenmektedir. Bu süre içerisinde *Sorgu Cevabı* alınmazsa, süreç 2 kez tekrar edilmektedir. Toplamda 3 kez *Sorgu Cevabı* alınmadığı taktirde, cevabın alınamadığına ilişkin yazılım bilgilendirilmektedir. Cevap alındığında ise, istemci yazılımı *İstek Gönderim* durumuna geçmektedir.



Şekil 24. İstek gönderimi akış diyagramı (Flow diagram of sending request)

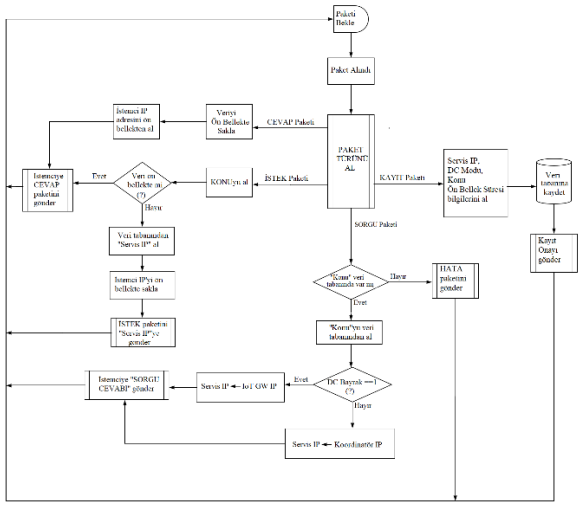
İstek durumunda, IoT Ağ Geçidi ya da Koordinatör'den gelen bilgiler *Cevap* paketlerinde yer almaktadır. *Sorgu* durumunda olduğu gibi, *İstek* durumunda da 3 kez işlem tekrar edilmektedir. Her bir işlem için 2 saniye bekleme süresi bulunmaktadır. Bu süreç içerisinde *Cevap* paketi alındığında, paket içerisinden veriler alınıp ve istemci yazılımına gönderilmektedir. Aksi durumda yazılıma "*yanıt yok*" bilgisi verilmektedir. Bu durum, Şekil 24'te akış diyagramı ile gösterilmiştir.

4.6.2. IoT Koordinatör (IoT Coordinator)

Önerilen protokoldeki en kritik görev, koordinatör bileşenindedir. Bu bileşen, IoT ekosisteminde kullanılabilir olan hizmetlerin listelenmesinden ve bu hizmetlere erişmek isteyen kullanıcılara aracılık etmekten sorumludur. Koordinatör bileşeninin dört temel amacı bulunmaktadır:

- Veri toplama ve veri transferinde aracılık yapmak,
- Servislerin erişim bilgilerini tutma ve yapılan keşif sorgularına yanıt vermek,
- Sunucu tabanlı iletişim ile doğrudan erişim yöntemleri arasında anahtarlama yapmak,
- Sensör düğümlerinin (IoT Ağ Geçidi) erişilebilirliğini kontrol etmek.

Nesnelerin İnternetinde servis keşfi sorgulaması için karşılaşılan önemli problemlerden biri de yerel düzeyde sorgulanan bir kaynak dizininin tanımlanması ihtiyacıdır (Jara vd., 2012). Tasarlanan protokolde koordinatör bu ihtiyacı karşılamak amacıyla, servis kaynaklarını ve bu servislere nasıl ulaşılacağı bilgisini tutmaktadır. İstemcilerden gelen servis keşif sorgularına göre istemci trafiğini servis kaynağına yönlendirmektedir. Bu cihaz, servis keşfi için kaynak olmanın yanı sıra, kayıt sürecinde tanımlanan belirli servis türleri için, aracılık görevi de üstlenmektedir. Buna göre, koordinatör cihazının fonksiyonlarını tanımlayan akış diyagramı Şekil 25'te gösterilmiştir.



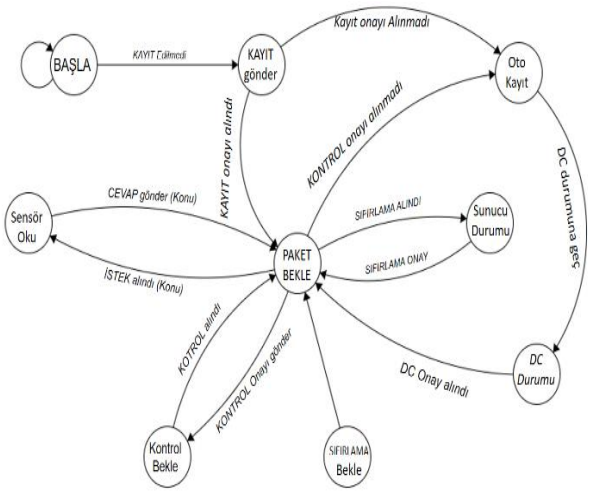
Şekil 25. IoT Koordinatörün akış diyagramı (Flow diagram of IoT Coordinator)

Koordinatöre gelen iletilerin belirlenmesi ve paket türüne göre yerine getirilecek görevler paket tipi analiz modülü ile sağlanmaktadır. Her bir paket türü için yapılması gereken görevler Şekil 25'teki akış diyagramında gösterilmektedir.

4.6.3. IoT Ağ Geçidi (IoT Gateway)

Nesnelerin İnternetinde sıklıkla kullanılan sensörler, yapıları itibari ile kısıtlı ya da hiç hesaplama yeteneği olmayan cihazlardır (Klauck & Kirsche, 2012). Bu nedenle sensörlerden elde edilen verilerin işlenebilmesi amacıyla, bu cihazlar ile iletişime geçebilen, sinyal gönderip alabilen, aldığı sinyalleri ağ ortamına sunabilecek *IoT Ağ Geçidi* olarak adlandırılan cihazlara ihtiyaç duyulmaktadır. IoT Ağ Geçidi kavramı Nesnelerin İnternetindeki kritik bileşenlerden biridir (Grygoruk & Legierski, 2016). Bu bileşen, sensör ağı ile uygulama katmanı arasında bir vekil olarak görev yapmaktadır. IoT ekosistemindeki nesnelerin dijital arayüzleri olan sensörler, IP ağı ile iletişime geçebilmesi için bu bileşenlere ihtiyaç duymaktadırlar.

Geliştirilen protokolde IoT Ağ Geçidi (IoTGW), sensörlerinden elde edilen verilerin, IP paketlerine kapsüllenip istemci yazılımına gönderilmesinden sorumludur. IoT Ağ Geçidi, hem doğrudan erişimi (DC) hem de koordinatör aracılığıyla erişimi desteklemektedir. Her bir servis için hangi erişim yönteminin desteklendiğini *Kayıt* paketi ile sağlanmaktadır. Kayıt işlemi gerçekleştikten sonra, istemciden gelen istekleri (*İstek Paketi*) cevaplamaya hazır hale gelmektedir. Şekil 26'da IoT Ağ Geçidinin sonlu durum akış diyagramı verilmiştir.

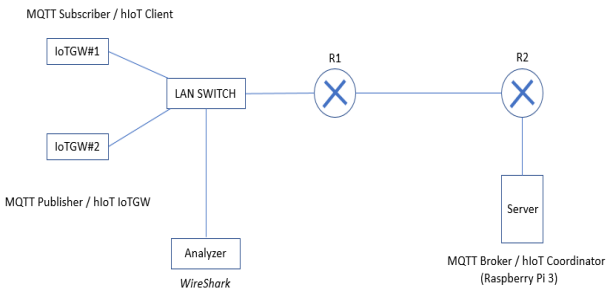


Şekil 26. IoT Ağ Geçidi akış diyagramı (Flow diagram of IoT Gateway)

Diyagramda belirtildiği gibi, IoT Ağ Geçidinin Koordinatör ile sürekli iletişim halinde olduğu *Kontrol* mesajları ile doğrulanmaktadır. Ancak Koordinatör, sunucu tabanlı iletişimde tek bir hata noktası oluşturmaktadır. Bu nedenle, sunucunun devre dışı kalıp kalmadığı sürekli olarak doğrulanmalıdır. Periyodik olarak gönderilen *Kontrol* mesajları sayesinde sunucunun erişilebilir olduğu kontrol edilmektedir. Sunucunun erişilebilirliği sağlanmazsa, IoT Ağ Geçidi *Otomatik Kayıt* durumuna geçmekte ve doğrudan erişimi (DC =1) desteklemektedir. Böylece istemci tarafından sorgulanan *Sorgu* ve *İstek* paketlerine de cevap verebilmektedir. Hibrit protokolün çalışma mantığı gereği, hizmet sunan IoT Ağ Geçidi doğrudan erişim durumunda (DC) olduğunda, koordinatörden gelen *Sıfırlama* paketi ile doğrudan erişim yetkisini kaybedip, sunucu bazlı çalışma moduna (Sunucu durumu) geçmektedir.

5. Başarım Değerlendirmesi (Performance Evaluation)

Geliştirilen protokolünün başarım değerlendirmesi için laboratuvar ortamında diğer ağ trafiklerinden izole edilmiş kapalı sistem bir topoloji hazırlanmış ve IoT uygulamalarında sıklıkla kullanılan MQTT protokolü ile karşılaştırılmıştır.



Şekil 27. Gerçek ortam değerlendirmesinin yapıldığı topoloji (Experimental topology developed for performance testing)

Şekil 27'de belirtilen topolojide, IoT Ağ Geçidi cihazları olarak Arduino Uno; sunucu olarak Raspberry Pi 3 donanımları kullanılmıştır. Sunucu, hIoT protokolünün mimarisinde *koordinatör* olarak; MQTT mimarisinde ise sunucu (MQTT Broker) olarak kullanılmıştır. MQTT broker sunucusunda açık kaynak "*Mosquitto*" kullanılmıştır (Johnsen vd., 2018). Hibrit IoT protokolünde ise sunucuda Python betikleri, veri tabanı olarak ise SQLite kullanılmıştır. Topolojide, farklı bant genişliklerinin etkisini yansıtmak amacıyla, sunucu ile IoT Ağ geçidi cihazları arasında R1 ve R2 ile gösterilen yönlendiriciler kullanılmıştır.

5.1. Bant Genişliğine Bağlı Gecikme Karşılaştırması (Delay Comparison by bandwidth)

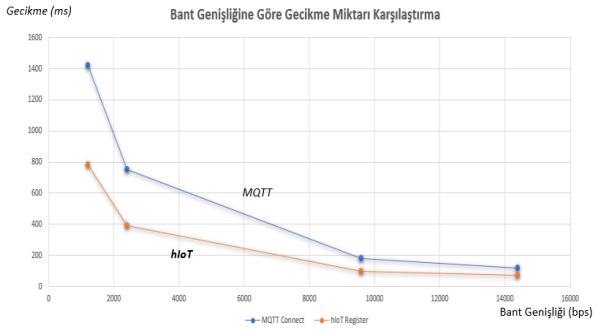
Geliştirilen Hibrit IoT protokolünün *Kayıt* aşaması ile, MQTT yapısındaki *Connect* aşaması, her ikisinde de sunucuya kayıt işlemi gerçekleştiğinden, işlevsellik açısından benzerlik göstermektedir. Şekil 27'de geliştirilen topolojide, R1-R2 arasındaki bant genişliği her senaryo için değiştirilmiş ve her bir bant genişliği için MQTT-Connect aşamasındaki ortalama gecikme değeri ile hIoT – Kayıt aşamasının ortalama gecikme değerleri hesaplanmıştır. Topolojide IoTGW cihazlarından gönderilen ve alınan tüm trafiklerin bir kopyası WireShark yazılımı ile yakalanmış ve yakalanan paketlere göre ortalama değerler elde edilmiştir. WireShark ile yakalanan

paketlerden elde edilen bilgilere göre, altı farklı bant genişliği için ortalama süreler milisaniye cinsinden Tablo 2'de verilmiştir.

Tablo 2. MQTT-Connect ve hIoT-Kayıt süreçlerinin farklı bant genişliklerindeki gecikme değerleri (Delay values of MQTT-Connect and hIoT-Register processes at different bandwidths)

Bant Genişliği (bps)	Gecikme Süreleri (ms)	
	MQTT (Connect)	Hibrit IoT (Kayıt)
1200	1423	782
2400	753	390
9600	182	100
14400	121	68
64000	26	41
128000	16	11

MQTT protokolünde sunucuya kayıt için öncelikle TCP 3-Yollu El Sıkışma işleminin (3-way handshake) işleminin gerçekleşmesi gerekmektedir. Bu işlemin ardından Connect ve Connect-ACK paketleri ile sunucuya bağlantı sağlanmaktadır. MQTT *Connect* aşaması ile, hIoT protokolünde bu aşamaya benzer olan "*Kayıt*" (hIoT-Register) aşamalarının gecikme süreleri Şekil 28'de verilmiştir.



Şekil 28. Bant genişliğine göre gecikme değerleri karşılaştırması (Delay comparison by bandwidth)

Grafikte gösterilen MQTT ve hIoT iletişiminden görüleceği üzere, düşük bant genişliklerinde hIoT protokolü daha düşük gecikme değerine sahiptir. Önerilen protokolde UDP kullanılması ve paket başlığının sade yapısı nedeniyle, kısıtlı kaynaklara sahip cihazlarda kullanımı daha uygun olmaktadır. Bu durum, iletişimin daha hızlı olmasını sağlamaktadır. IoT ekosisteminde hız gerektiren uygulamalarda hIoT protokolü, MQTT protokolüne göre hız açısından daha yüksek performans göstermektedir. Ancak bant genişliği arttığında, her iki protokolün gecikme süreleri arasındaki fark kapanmaktadır.

5.2. Yük Miktarına Göre Gecikme Değerleri (Delays by Payload)

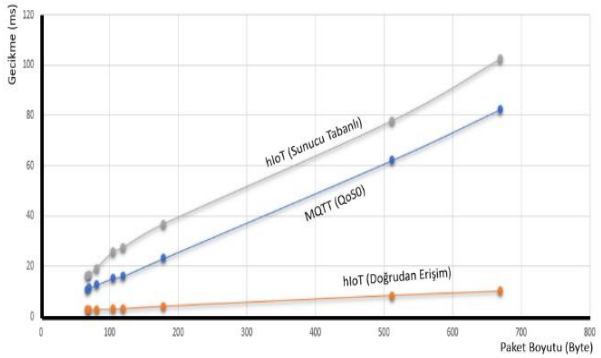
Aynı topoloji üzerinde veri transferi başarım değerlendirmesi için yapılan karşılaştırmada ise, farklı yük değerleri için ortalama gecikmeler hesaplanmıştır. Buna göre IoTGW#1 MQTT "Abone" olarak; IoTGW#2 ise MQTT "Yayıncı" olarak tasarlanmıştır. Aynı topolojide IoTGW#1 hIoT İstemcisi, IoTGW#2 ise Ağ Geçidi olarak tasarlamıştır. Farklı paket boyutları için MQTT yayıncıdan gönderilen 30 mesajın, MQTT aboneye ulaşma süreleri WireShark yazılımı ile yakalanmış ve ortalama süre hesaplanmıştır.

Aynı şekilde, *İstek* paketinin hIoT istemcisinden gönderilmesi ile *cevap* paketinin istemciye ulaşması arasındaki süreler 30 paket için ortalama olarak hesaplanmıştır. hIoT protokolünde, doğrudan erişim için ortalama süre ile sunucu üzerinden gerçekleşen iletişim için ortalama süreler ayrı ayrı hesaplanmış ve milisaniye cinsiden Tablo 3'te verilmiştir.

Tablo 3. Farklı yük miktarlarına göre veri transferi karşılaştırması (Data transfer comparison by different payloads)

Yük Miktarı (Bayt)	Gecikme Süreleri (ms)		
	MQTT (Publish-Subscribe)	hIoT (İstek-Cevap) Doğrudan erişim	hIoT (İstek-Cevap) Sunucu Tabanlı
68	10,7	2,5	15,8
70	11,3	2,5	16,2
81	12,5	2,7	19,1
105	15,2	2,8	25,5
120	16	3	27,5
179	23,1	4	36,7
512	62	8,2	77,8
670	82,2	9,9	102,4

Tablodaki değerlere göre yük miktarı ve gecikme değerleri Şekil 29'daki grafikte verilmiştir.



Şekil 29. Taşınan yük miktarlarına göre veri transferinde yaşanan gecikme değerleri (Delays in data transfer according to the payload)

Görüleceği üzere, geliştirilen hIoT protokolünde doğrudan erişim MQTT QoS0'a göre daha düşük gecikme değerlerine sahiptir. MQTT protokolünde, yayıncıdan gönderilen mesaj öncelikle sunucuya gönderilmekte, ardından sunucu tarafından aboneye iletilmektedir. Sunucu, abone ile yayıncı arasında ek bir sekme oluşturduğundan gecikme artmaktadır. Oysa hIoT protokolünde doğrudan iletişim sağlandığından, sunucudan kaynaklanan gecikmeler iletişime yansımamaktadır. Geliştirilen protokolün sunucu tabanlı iletişiminde ise MQTT'ye oranla daha yüksek gecikmeler ortaya çıkmaktadır.

Gecikmelerin hassas olduğu servisler için doğrudan erişim yöntemi tercih edilebilir. Gecikmenin göreli olarak önemsiz olduğu ancak sunucu üzerinden erişim kontrolünün yapılması gerektiği durumlarda ise sunucu tabanlı iletişim daha uygun olacaktır.

5.3. Servis Keşif Süreci Karşılaştırma (Service Query Process Comparison)

IoT ekosisteminde kullanılan cihazlar, birçok nesne ve fiziksel ortam ile etkileşim halindedir. Böyle bir ortamda istenilen servislerin keşfedilmesi için efektif bir keşif mekanizmasına ihtiyaç duyulmaktadır. Bu ihtiyacı karşılamak amacıyla Klauck ve Kirsche tarafından, kısıtlı kaynaklara sahip cihazlar için mDNS ve DNS-SD kombinasyonu bir sistem geliştirilmiştir. Adı geçen çalışmada MQTT ve CoAP protokolünün uçtan-uca iletişim sağlayamadığı, mesajların dönüştürülmesi gerektiği ve bu nedenle gecikmeye neden olduğu, hata olasılığının bulunduğu ve işlemin karmaşık olduğu belirtilmiştir.

Bu çalışma kapsamında geliştirilen Hibrit IoT protokolünün önemli özelliklerinden biri de ek bir protokol kullanımına ihtiyaç olmadan *"Servis Keşif"* mekanizmasını barındırmasıdır. hIoT protokolünün "*Kayıt*" aşamasında koordinatör sunucuya kayıt ettirilen servisler, servis keşif aşamasındaki "*Sorgu*" paketleri ile sorgulanır. Şekil 27'de gösterilen topolojide farklı bant genişlikleri için servis keşif sorgularına verilen cevaplara ait gecikme süreleri milisaniye cinsinden Tablo 4'te verilmiştir.

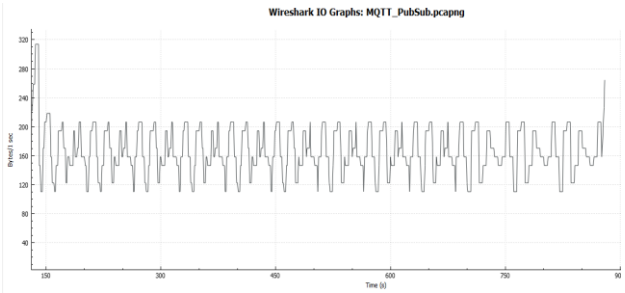
Tablo 4. Farklı bant genişlikleri için servis keşif süreci gecikme değerleri (Service query delays for different bandwidths)

Bant Genişliği (bps)	Servis Keşif Süresi (ms)
1200	890
2400	384
9600	99
14400	67
64000	18
125000	15

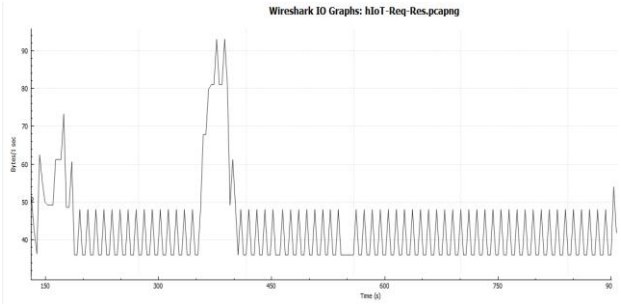
Klauck ve Kirsche'nin geliştirmiş oldukları sistemde (Klauck & Kirsche, 2012), 2 sekme (hop count) için gecikme değeri 1954 ms olduğu belirtilmiştir. Geliştirilen hIoT protokolünde ise servis keşfi en az 890 ms olarak ölçülmüştür. hIoT protokolü, kampüs ağları gibi sınırlı alanlarda "konu" bazlı olarak çalışacak şekilde tasarlanmıştır. Bu sınırlılık nedeniyle hIoT internet ortamında kullanılamasa da, karmaşık DNS çözümlerine göre yerel ağlarda kullanımı uygun olacaktır.

5.4. Bant Genişliği Kullanımı Karşılaştırması (Bandwidth Usage Comparison)

Hazırlanan topoloji üzerinde, MQTT ve hIoT protokolü için sabit aralıklara veri transferi yapılmış ve gönderilen tüm paketler WireShark ile yakalanmıştır. Her iki protokolün bu süre içerisinde toplam bant genişliği kullanımı Şekil 30 ve Şekil 31'de gösterilmiştir.



Şekil 30. MQTT Bant genişliği kullanımı (Bandwidth usage in MQTT)



Şekil 31. hIoT bant genişliği kullanımı (Bandwidth usage in hIoT)

Hibrit IoT protokolünün MQTT'ye göre veri transferinde daha az bant genişliği kullanarak, sensör düğümler ile iletişime geçtiği şekildeki grafiklerden görünmektedir. MQTT protokolü TCP tabanlı olduğundan, veri iletişiminde daha fazla paket akışı gerçekleşmektedir. Diğer taraftan, geliştirilen hibrit protokol UDP tabanlı olduğundan, TCP'ye göre daha küçük başlık bilgisi taşımaktadır. Başlık bilgisinin boyutunun daha küçük olması ve kullanılan paket akışlarının daha az olması, daha düşük bant genişliği kullanımını sağlamaktadır. Daha az bant genişliği kullanımı, sensör düğümlerinin de verimli kullanımını sağlamaktadır.

6. Sonuç ve Tartışma (Result and Discussion)

IoT ekosisteminde nesnelerden gelen veriler, uygulamalara göre farklılık göstermektedir. Ancak IoT uygulamalarında transfer edilen veriler genellikle düşük ve sınırlı boyuttadır (Nóbrega vd., 2019). Sensörlerden elde edilen düşük boyuttaki verilerin taşınması için geliştirilen hIoT protokolü, UDP tabanlı ve küçük paket başlığına sahiptir. Thangavel ve arkadaşlarının çalışmasında UDP tabanlı protokolün daha düşük ek yük getirdiği ifade edilmiştir (Thangavel vd., 2014). Nesnelerin İnternetinde kullanılan cihazlar, kısıtlı bellek, işlemci ve güç gibi

düşük kaynaklara sahip olduğu göz önüne alındığında, düşük ek yük, kaynakların verimli kullanımına da katkı sağlamaktadır.

Gecikme sürelerinin düşük olması, özellikle gerçek zamanlı uygulamalarda önemli bir kriter olarak görülebilir. Jürgen ve arkadaşlarının çalışmasında IoT ortamlarında UDP tabanlı iletişimlerin TCP tabanlı iletişimlere oranla daha düşük RTT değerine sahip olduğu ifade edilmektedir (Lars, 2015). Geliştirilen hibrit IoT protokolü, doğrudan iletişim yönteminde, MQTT protokolüne oranla daha düşük gecikme değerlerine sahip olmaktadır.

Yaygın kullanılan IoT protokollerinde, sensörler aracılığıyla nesnelerden elde edilen bilgiler sunucuya periyodik olarak gönderilmektedir. Sunucu aracılığıyla da abonelere bu bilgi tekrar yayınlanmaktadır. Bu durum sensörün sürekli kullanımı ve sensör ömrünün göreli olarak kısalması ve güç tüketiminin artması anlamına gelmektedir. Geliştirilen protokolün kullanılacağı mimaride ise isteğe bağlı iletişim amaçlanmıştır. Böylece güç tüketiminin göreli olarak düşürülmesi hedeflenmektedir.

Bu çalışma kapsamında önerilen hibrit protokolde, veri iletişiminde sürekli olarak sunucuya bağımlılık bulunmamakta, ağdaki cihazlar birbirleri ile doğrudan haberleşebilmektedir. Bu sayede tek bir hata noktasının oluşmasını önlenmekte ve trafik yükü dağıtılarak darboğaz oluşumunun önüne geçilmektedir. Yine, merkezi sunucu kullanılan senaryoları da desteklemek amacıyla, sunucu tabanlı da çalışabilecek yapıda tasarlamıştır. Veriye erişimin hassas ve anlık olması gerektiği durumlarda, sensörler aracılığı ile nesnelere anlık olarak uçtan uca bir şekilde iletişim sağlanabilmektedir.

Heterojen yapılardaki IoT sistemlerde, her servise erişim aynı derecede öneme sahip değildir. Kimi servisler doğrudan iletişim gerektiren kritik öneme sahip iken, kimi servislerde ise bu gecikme telafi edilebilir. Aynı şekilde bazı servislere doğrudan erişim güvenlik ve performans riskleri barındırabileceğinden, aracı yapıların kullanılması daha uygun olabilir. Bu durum ihtiyaçlara, kullanım senaryolarına, güvenlik politikalarına göre değişkenlik göstermektedir. Bu çalışmada, IoT sistemlerinde yukarıda bahsi geçen çeşitli kullanım senaryolarını desteklemek amacıyla hibrit bir protokol tasarlanmıştır.

Bu çalışmada, servisin doğrudan erişimi destekleyip desteklemediği, servis kayıt aşamasında belirlenmekte ve sürekli olarak sabit kalmaktadır. Ancak, belirlenen trafik kriterlerine göre anlık karar verme algoritmaları geliştirilebilir. Yine çeşitli parametrelere göre, trafik tahmini yapılıp, sunucu bazlı ya da uçtan-uca iletişim arasında dinamik geçişler sağlanmasına yönelik çalışmalar yapılabilir.

IoT ekosisteminde kullanılan cihazların büyük bir kısmı sınırlı kapasitelere sahip olduğundan, protokolün tasarımı sürecinde şifreleme gibi yüksek bellek ve işlemci gücü gerektiren güvenlik özellikleri bu çalışmada kapsam dışına alınmıştır. Teknolojinin gelişmesi ile bu sınırlılıkların giderek azalacağı göz önüne alındığında, protokolde güvenli iletişim sağlamak amacıyla şifreleme, veri bütünlüğü ve kimlik doğrulama çalışmaları yapılabilir.

Teşekkür (Acknowledgement)

Bu çalışma doktora tez çalışmasından türetilmiştir.

Çıkar Çatışması (Conflict of Interest)

Yazarlar tarafından herhangi bir çıkar çatışması beyan edilmemiştir. No conflict of interest was declared by the authors.

Kaynaklar (References)

Anjum, A., Ilyas, M. U., Gorden, D., Gar, C., & Guo vd., 2014. Internet of Things – From Research and Innovation to Market Deployment. (O. Vermesan & P. Friess, Eds.)Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics) (Vol. 6). River Publishers.

Atzori, L., Iera, A., & Morabito, G., 2010. The Internet of Things: A Survey. Computer Networks: The International Journal of Computer and Telecommunications Networking, 54(15), 2787–2805. http://doi.org/10.1016/j.comnet.2010.05.010

Bellavista, P., & Zanni, A., 2016. Towards better scalability for IoT-cloud interactions via combined exploitation of MQTT and CoAP. 2016 IEEE 2nd International Forum on Research and Technologies for Society and Industry Leveraging a Better Tomorrow, RTSI 2016. http://doi.org/10.1109/RTSI.2016.7740614

Chen, Y., & Kunz, T., 2016. Performance evaluation of IoT protocols under a constrained wireless access network. In 2016 International Conference on Selected Topics in Mobile and Wireless Networking, MoWNeT 2016. http://doi.org/10.1109/MoWNet.2016.7496622

Choi, D., Jung, J., Kang, H., & Koh, S., 2017. Cluster-based CoAP for Message Queueing in Internet-of-Things Networks, 584–588.

Collina, M., Bartolucci, M., Vanelli-Coralli, A., & Corazza, G. E., 2014. Internet of Things application layer protocol analysis over

error and delay prone links. In 2014 7th Advanced Satellite Multimedia Systems Conference and the 13th Signal Processing for Space Communications Workshop, ASMS/SPSC 2014 (s. 398–404). http://doi.org/10.1109/ASMS-SPSC.2014.6934573

Dizdarevic, J., Carpio, F., & Jukan, A., 2018. Survey of Communication Protocols for Internet-of-Things and Related Challenges of Fog and Cloud Computing Integration. CoRR, 1(1), 1–27.

Florea, I., Rughinis, R., Ruse, L., & Dragomir, D., 2017. Survey of Standardized Protocols for the Internet of Things. Proceedings - 2017 21st International Conference on Control Systems and Computer, CSCS 2017, 190–196. http://doi.org/10.1109/CSCS.2017.33

Fortino, G., Guerrieri, A., Russo, W., & Savaglio, C., 2014. Integration of agent-based and Cloud Computing for the smart objects-oriented IoT. Proceedings of the 2014 IEEE 18th International Conference on Computer Supported Cooperative Work in Design, CSCWD 2014, 493–498. http://doi.org/10.1109/CSCWD.2014.6846894

Grygoruk, A., & Legierski, J., 2016. IoT gateway – implementation proposal based on Arduino board. In Proceedings of the Federated Conference on Computer Science (Vol. 8, s. 1011–1014). http://doi.org/10.15439/2016F283

Gubbi, J., Buyya, R., Marusic, S., & Palaniswami, M., 2013. Internet of Things (IoT): A vision, architectural elements, and future directions. Future Generation Computer Systems, 29(7), 1645–1660. http://doi.org/10.1016/j.future.2013.01.010

Gündoğan, C., Kietzmann, P., Lenders, M., Petersen, H., Schmidt, T. C., & Wählisch, M., 2018. NDN, CoAP, and MQTT: A Comparative Measurement Study in the IoT. http://doi.org/arXiv:1806.01444v2

Höller, J., Tsiatsis, V., Mulligan, C., Karnouskos, S., Avesand, S., & Boyle, D., 2014. Part III. IoT Use Cases. From Machine-To-Machine to the Internet of Things, 233–235. http://doi.org/10.1016/B978-0-12-407684-6.00033-4

Huo, C., 2014. A Centralized IoT Middleware System for DevicesWorking Across Application Domains Using Self-descriptive Capability Profile. Doktora Tezi, California Üniversitesi.

Ishaq, I., Carels, D., Teklemariam, G., Hoebeke, J., Abeele, F., Poorter, E., vd., 2013. IETF Standardization in the Field of the Internet of Things (IoT): A Survey. Journal of Sensor and Actuator Networks (Vol. 2). http://doi.org/10.3390/jsan2020235

ITU-T. (2014). Focus Group on M2M service layer: APIs and protocols overview. Çevrimiçi, https://www.itu.int/opb/publications.aspx?parent=T-FG&selection=6&sector, Son Erişim 18.02.2019

Jara, A. J., Martinez-Julia, P., & Skarmeta, A., 2012. Light-weight multicast DNS and DNS-SD (lmDNS-SD): IPv6-based resource and service discovery for the web of things. Proceedings - 6th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, IMIS 2012, 731–738. http://doi.org/10.1109/IMIS.2012.200

Johnsen, F. T., Landmark, L., Hauge, M., Larsen, E., & Kure, O., 2018. Publish/Subscribe Versus a Content-Based Approach for Information Dissemination. In MILCOM 2018 - 2018 IEEE Military Communications Conference (MILCOM) (s. 1–9). http://doi.org/10.1109/MILCOM.2018.8599786

Kang, B., Kim, D., & Choo, H., 2017. Internet of Everything: A Large-Scale Autonomic IoT Gateway. IEEE Transactions on Multi-Scale Computing Systems, 3(3), 206–214. http://doi.org/10.1109/TMSCS.2017.2705683

Kim, H., 2017. Securing the Internet of Things via Locally Centralized , Globally Distributed Authentication and Authorization.

Kim, S. M., Choi, H. S., & Rhee, W. S., 2016. IoT home gateway for auto-configuration and management of MQTT devices. 2015 IEEE Conference on Wireless Sensors, ICWiSE 2015, 12–17. http://doi.org/10.1109/ICWISE.2015.7380346

Kimsey, T., Jeffords, J., Moghaddam, Y., & Rucinski, A., 2015. An IoT Based Service System as a Research and Educational Platform. New Trends in Intelligent Information and Database Systems, 598, (s. 249–257). http://doi.org/10.1007/978-3-319-16211-9_26

Klauck, R., & Kirsche, M., 2012. Bonjour Contiki: A Case Study of a DNS-Based Discovery Service for the Internet of Things. In X.-Y. Li, S. Papavassiliou, & S. Ruehrup (Eds.), Ad-hoc, Mobile, and Wireless Networks (pp. 316–329). Berlin, Heidelberg: Springer Berlin Heidelberg.

Lars, D., 2015. Performance Evaluation of M2M Protocols Over Cellular Networks in a Lab Environment. In Conference: 18th International Conference on Intelligence in Next Generation Networks (ICIN), 2015, Paris, Fransa (s. 70–75). http://doi.org/http://dx.doi.org/10.1109/ICIN.2015.7073809

Luzuriaga, J. E., Perez, M., Boronat, P., Cano, J. C., Calafate, C., & Manzoni, P., 2015. A comparative evaluation of AMQP and MQTT protocols over unstable and mobile networks. In 2015 12th Annual IEEE Consumer Communications and Networking Conference, CCNC 2015 (s. 931–936). http://doi.org/10.1109/CCNC.2015.7158101

M. Koster, & Keranen, A., 2015. Message Queueing in the Constrained Application Protocol (CoAP). Online: https://tools.ietf.org/html/draft-koster-core-coapmq-00, Son Erişim: 20.02.2019

Menyah, N., 2017. A Real Time Demonstrative Analysis Of Lightweight Payload Encryption in Resource Constrained Devices Based on MQTT. Yüksek Lisans Tezi, Sakarya Üniversitesi.

Naik, N., 2017. Choice of Effective Messaging Protocols for IoT Systems : MQTT , CoAP , AMQP and HTTP.

Ngu, A. H., Gutierrez, M., Metsis, V., & Sheng, Q. Z. (2017). IoT Middleware : A Survey on Issues and Enabling Technologies. IEEE Internet of Things Journal, 4(1), 1–20. http://doi.org/10.1109/JIOT.2016.2615180

Nóbrega, L., Gonçalves, P., Pedreiras, P., Pereira, J., Nóbrega, L., Gonçalves, P., … Pereira, J., 2019. An IoT-Based Solution for Intelligent Farming. Sensors, 19(3), 603. http://doi.org/10.3390/s19030603

Pathania, N., 2017. Traffic Prioritization in an MQTT Gateway. In International Journal of Computer Applications (Vol. 164, s. 32–38).

Sasaki, Y., Yokotani, T., & Mukai, H., 2018. Proposals on IoT Communication through MQTT over L2 Network and their Performance Evaluation. In 2018 International Conference on Innovations in Information Technology (IIT) (s. 30–35).

Sch, B., Bauer, J., & Aschenbruck, N., 2017. Improving Energy Efficiency of MQTT-SN in Lossy Environments using Seed-based Network Coding. 42nd Annual IEEE Conference on Local Computer Networks. http://doi.org/10.1109/LCN.2017.87

Schmitt, A., Carlier, F., & Renault, V., 2018. Dynamic bridge generation for IoT data exchange via the MQTT protocol. Procedia Computer Science, 130, 90–97. http://doi.org/10.1016/j.procs.2018.04.016

Shelby, Z., Hartke, K., & Bormann, C., 2014. The Constrained Application Protocol (CoAP). Online: http://doi.org/10.17487/rfc7252, Son Erişim: 20.02.2019

Shin, I.-J., Song, B.-K., & Eom, D.-S., 2017. International Electronical Committee (IEC) 61850 Mapping with Constrained Application Protocol (CoAP) in Smart Grids Based European Telecommunications Standard Institute Machine-to-Machine (M2M) Environment. Energies, 10(3), 393. http://doi.org/10.3390/en10030393

Su, P. H., Shih, C., Hsu, J. Y., Lin, K., & Wang, Y., 2014. Decentralized Fault Tolerance Mechanism for Intelligent IoT / M2M Middleware. In 2014 IEEE World Forum on Internet of Things (WF-IoT) (s. 45–50).

Talaminos-Barroso, A., Estudillo-Valderrama, M. A., Roa, L. M., Reina-Tosina, J., & Ortega-Ruiz, F., 2016. A Machine-to-Machine protocol benchmark for eHealth applications - Use case: Respiratory rehabilitation. Computer Methods and Programs in Biomedicine, 129, 1–11. http://doi.org/10.1016/j.cmpb.2016.03.004

Tantitharanukul, N., Osathanunkul, K., Hantrakul, K., Pramokchon, P., & Khoenkaw, P., 2016. MQTT-Topics Management System for sharing of Open Data, 1–6.

Thangavel, D., Ma, X., Valera, A., Tan, H. X., & Tan, C. K. Y., 2014. Performance evaluation of MQTT and CoAP via a common middleware. In IEEE ISSNIP 2014 - 2014 IEEE 9th International Conference on Intelligent Sensors, Sensor Networks and Information Processing, Conference Proceedings. http://doi.org/10.1109/ISSNIP.2014.6827678

Thota, P., & Kim, Y., 2016. Implementation and Comparison of M2M Protocols for Internet of Things. In 2016 4th Intl Conf on Applied Computing and Information Technology/3rd Intl Conf on Computational Science/Intelligence and Applied Informatics/1st Intl Conf on Big Data, Cloud Computing, Data Science Engineering (ACIT-CSII-BCD) (s. 43–48). http://doi.org/10.1109/ACIT-CSII-BCD.2016.021