T.C.
SAKARYA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

NESNELERİN İNTERNETİ UYGULAMA KATMANI HABERLEŞME PROTOKOLLERİNİN BAŞARIM ANALİZİ

YÜKSEK LİSANS TEZİ

Mehmet Ali EBLEME

Enstitü Anabilim Dalı : BİLGİSAYAR VE BİLİŞİM MÜHENDİSLİĞİ

Tez Danışmanı : Doç. Dr. Cüneyt BAYILMIŞ

Mayıs 2019

T.C.
SAKARYA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

NESNELERİN İNTERNETİ UYGULAMA KATMANI HABERLEŞME PROTOKOLLERİNİN BAŞARIM ANALİZİ

YÜKSEK LİSANS TEZİ

Mehmet Ali EBLEME

Enstitü Anabilim Dalı : BİLGİSAYAR VE BİLİŞİM MÜHENDİSLİĞİ

Bu tez 24/05/2019 tarihinde aşağıdaki jüri tarafından oybirliği / oyçokluğu ile kabul edilmiştir.

Doç. Dr.	**Doç. Dr.**	**Dr.Öğr.Üyesi**
Kerem KÜÇÜK	**Cüneyt BAYILMIŞ**	**Abdullah SEVİN**
Jüri Başkanı	**Üye**	**Üye**

BEYAN

Tez içindeki tüm verilerin akademik kurallar çerçevesinde tarafımdan elde edildiğini, görsel ve yazılı tüm bilgi ve sonuçların akademik ve etik kurallara uygun şekilde sunulduğunu, kullanılan verilerde herhangi bir tahrifat yapılmadığını, başkalarının eserlerinden yararlanılması durumunda bilimsel normlara uygun olarak atıfta bulunulduğunu, tezde yer alan verilerin bu üniversite veya başka bir üniversitede herhangi bir tez çalışmasında kullanılmadığını beyan ederim.

Mehmet Ali EBLEME
10.05.2019

ÖNSÖZ

Danışman hocam Doç. Dr. Cüneyt Bayılmış'a teşekkürü bir borç bilirim. Beni, IoT alanında çalışmaya sevk ederek yepyeni ve muhteşem tecrübeler kazanmama ve bu alanda iyi bir işe girmeme vesile oldu. Yüksek lisans dönemim boyunca bana hep destek oldu. Üzerimdeki emeklerinin karşılığını asla ödeyemem.

Yüksek lisans eğitimim ve tezimi hazırlama sürecim boyunca, her konuda bana destek olan, araştırmanın planlanmasından yazılmasına kadar tüm aşamalarında yardımlarını esirgemeyen, zor anlarımda beni rahatlatan Ünal Çavuşoğlu hocama teşekkürlerimi sunarım.

Yüksek lisans dönemim boyunca ders aldığım tüm hocalarıma ayrıca teşekkür ederim.

İÇİNDEKİLER

SİMGELER VE KISALTMALAR LİSTESİ

AMQP	: Advanced Message Queuing Protocol
CoAP	: Constrained Application Protocol
CONNACK	: Connection acknowledgement
CORE	: Constrained restful environment
CPU	: İşlemci
DDS	: Data Distribution Service
DTLS	: Datagram transport layer security
HTTP	: Hypertext transfer protocol
HTTPS	: Hypertext transfer protocol secure
IETF	: Internet engineering task force
IoT	: Internet of Things
IP	: Internet protocol
JSON	: JavaScript object notation
Kb	: Kilobyte
LAN	: Local area network
M2M	: Machine to machine
MAC	: Media access control
MHz	: Megahertz
MQTT	: Message queue telemetry transport
MQTT-SN	: Message queue telemetry transport for Sensor Networks
OASIS	: Organization for the advancement of structured information standards
PUBACK	: Publish acknowledgement
PUBCOMP	: Publish complete
PUBREC	: Publish received
PUBREL	: Publish released

QoS	: Quality of service
REST	: Representational state transfer
SAGAT	: Situation Awareness Global Assessment Technique
SSL	: Secured socket layer
SOAP	: Simple Object Access Protocol
SUBACK	: Subscribe acknowledgement
TCP	: Transmission control protocol
TLS	: Transport layer security
UDP	: User datagram protocol
UNSUBACK	: Unsubscribe acknowledgement
UTF	: Unicode transformation format
WSN	: Wireless sensor networks
XML	: Extensible markup language
XMPP	: Extensible Messaging and Presence Protocol

ŞEKİLLER LİSTESİ

TABLOLAR LİSTESİ

ÖZET

Anahtar kelimeler: MQTT, CoAP, WebSocket, Enerji tüketimi, AMQP, DDS, XMPP, SOAP, MQTT-SN, REST, iş çıkarım oranı, gecikme süresi

Bu çalışmada nesnelerin internet'i uygulamalarında sıkça kullanılan MQTT, MQTT-SN, CoAP, AMQP, WebSocket, XMPP, DSS, SOAP ve REST haberleşme protokollerinin genel olarak tanımı yapılıp çalışma modellerinden bahsedildikten sonra MQTT, CoAP ve WebSocket protokollerinin deneysel sonuçlar eşliğinde performans analizleri yapılmıştır.

Deney düzeneğinde gerçek dünya şartlarını simüle etmek amacıyla; istemci olarak sınırlı bir cihaz olan Wemos D1 Uno cihazı kullanılmış ve bu cihazdan sunucu veya aracılara 8, 16, 32, 64, 128, 256, 512, 1024 bayt olarak değişen mesaj yükleri ile her bir yük boyutu değeri için 10000 mesaj gönderilmiştir. Sunucu veya aracı cihaz olarak ise bir dizüstü bilgisayar kullanılmış ve gerekli sunucu yazılımları sıfırdan yazılmıştır. Haberleşme kablosuz bir ağ üzerinden gerçekleştirilmiştir. Deneyler sonucunda bu üç protokolün ortalama mesaj gecikme süresi, iş çıkarma oranı (throughput) ve enerji tüketimi değerlendirilmiştir.

Deney çıktıları eşliğinde MQTT, CoAP ve WebSocket hakkında analizler yapılmış ve bu protokoller birbirleriyle özellik ve deney sonuçları açısından karşılaştırılmıştır. Bu çalışmanın sonunda nesnelerin internet'i uygulamalarında kullanılacak haberleşme protokolünün seçimi hakkında tasarımcılara öneriler sunulmuştur.

EXAMINATION AND PERFORMANCE EVALUATION FOR INTERNET OF THINGS APPLICATION LAYER COMMUNICATION PROTOCOLS

SUMMARY

Keywords: MQTT, CoAP, WebSocket, Energy consumption, AMQP, DDS, XMPP, SOAP, MQTT-SN, REST, throughput, delay

In this study, MQTT, MQTT-SN, CoAP, AMQP, WebSocket, XMPP, DSS, SOAP and REST communication protocols, which are frequently used in internet of things (IoT) applications, have been defined in general. After mentioning the working models, performance experiments were performed for MQTT, CoAP and WebSocket protocols.

To simulate real World conditions, we used Wemos D1 Uno constrained devices as clients to send data with changing in the range of 8, 16, 32, 64, 128, 256, 512, 1024 bytes payload to servers or brokers. And send 10.000 messages in every experiment. We used a laptop as a server or broker and developed necessary applications from scratch. Communication between clients and servers/brokers performed with a mobile WiFi network. We examined average delay, throughput, and energy consumption parameters from the results of experiments.

Analysis of MQTT, CoAP and WebSocket were performed with the help of the experimental outputs and these protocols were compared with each other in terms of features and test results. At the end of this study, suggestions were presented to the designers about the communication protocol to be used in IoT.

BÖLÜM 1. GİRİŞ

Nesnelerin İnternet'i (Internet of Things, IoT) kişisel ve sosyal haberleşme, işletme ve hizmet izleme gibi farklı uygulama alanlarında, düşük kapasite ve işlem gücüne sahip "her şey" için her zaman, her yerde, herhangi bir şekilde bağlantı oluşturmayı hedefleyen bir paradigmadır [1].

Son yıllarda Nesnelerin İnternet'i alanında süre gelen yoğun araştırmalar sonucunda birçok sınırlı veya kapsamlı yeni cihaz ve sensör üretildi. Bunlarla beraber; ürettikleri verilerin saklanacağı, izlenip yorumlanacağı veri tabanı bazlı yönetim araçlarının sayıları dolaylı olarak giderek arttı ve sürekli geliştirildiler.

Güncel bazı tahminler IoT cihazlarının sayısının 2020'ye kadar 200 milyardan fazla aralıklı bağlantı ile 20-30 milyar bandında olacağını veya aşacağını ve 700 milyar avronun üzerinde bir pazar oluşturacağını öne sürmektedir [1][2][3]. Bununla beraber yine 2020'ye kadar kuruluşların %65'inin IoT ürünlerine adapte olacağı öngörülmektedir [1].

Potansiyel IoT ürünleri değişen aralıklarla ve sürekli veri üreten; akıllı telefonlar, bio-nano nesneler, vücut sensörleri, akıllı etiketler, giyilebilir cihazlar, gömülü sistemlerde nesneler, genel maksatlı sensörler, geleneksel elektronik aygıtlar vb. çok geniş bir uygulama alanına yayılmış cihazlardır.

IoT araştırmacılarının önemle üzerinde durduğu noktalar ise IoT ürünlerinden elde edilen –küresel düzeyde devasa boyutta olan– verilerin, yönetilmesi, bu verilerin boyutlarının, tüketilen enerjinin, harcanan bant genişliğinin, gereken işlemci kapasitesinin minimize edilmesi ve yönetim araçlarına iletim şeklinin belirlenmesidir.

Her defasında gönderilecek veri boyutu, enerji tüketimi, veriyi üretecek ve gönderecek cihazın sınırlı olması, veri güvenliği, kullanılacak bağlantı türüne bağlı olarak bant genişliği, verinin iletilemediği durumlardaki tepkiler vb. birçok etmenin veri iletiminde göz önünde bulundurulması gerekmektedir. Bu etmenlere bağlı olarak farklı kullanım senaryolarına sahip farklı haberleşme modelleri ve protokolleri oluşturulmuştur. MQTT, MQTT-SN, CoAP, AMQP, REST, WEBSOCKET, XMPP, SOAP, DDS IoT'de veri iletiminde kullanılan popüler protokollerden bazılarıdır.

Her bir IoT haberleşme protokolü geliştiricileri tarafından yeterince açıklansa da bunların –gerçek hayatta, gerçek cihazlarla yapılan– farklı senaryolardaki karşılaştırılmaları yeterince yapılmamış veya dünyaya sunulmamıştır.

Bu tez çalışmasında IoT dünyasında popüler ve yaygın kullanıma sahip veri iletim protokollerinin genel yapılarından, haberleşme modellerinden, farklı durumlarda davranış şekillerinden bahsederek aralarındaki farklar gösterilmeye çalışılmaktadır. Bununla beraber belirli birkaç IoT haberleşme protokolünün performans değerlendirmesi; iş çıkarma oranı (throughput), enerji tüketimi ve mesaj gecikme süreleri parametrelerine göre gerçekleştirilip sonuçlar grafiksel olarak sunulmakta ve aralarındaki istatiksel farkın daha açık görülebilmesi sağlanmaya çalışılmaktadır.

1.1. Literatür Özeti

IoT'de kullanılan haberleşme protokollerinin birbirinden farkları ve performans karşılaştırmaları üzerine literatürde birçok çalışma yapılmıştır.

Chaudhary ve arkadaşları [4] MQTT, CoAP ve AMQP protokolleri üzerinde kablolu, kablosuz ve 4G bağlantılarında deneyler gerçekleştirmişlerdir. Deneylerini bu protokollerde 10, 100, 1000 ve 10000 mesaj göndererek gerçekleştirmişler ve sonuç olarak bant genişliği kullanımı, saniyede iletilen mesaj sayısını, her bir deney için protokollerin ürettiği toplam paket sayılarını çıktı almışlar ve bu sonuçları açıklamışlardır. İstemci tarafı için farklı deneyler olmak üzere bir Rasperry Pi 3 ve bir dizüstü bilgisayar, sunucu tarafı içinse HP Z 820 marka ve model bir workstation

kullanmışlardır. Bu sunucu üzerinde protokoller hakkında performans değerlendirmelerini CoAP için libcoap kütüphanesini kullanarak gerçekleştirirlerken MQTT ve AMQP için Mosquitto ve RabbitMQ gibi gelişmiş bilgisayar yazılımları kullanmışlardır. Bununla beraber deneylerinde test.mosquitto.org gibi web servislerinden de yararlanmışlardır.

Srinivasan ve arkadaşları [5] yaptıkları çalışmada ise WebSocket ve HTTP protokolleri üzerinden veri akışı deneyleri gerçekleştirmişler ve bu protokollerin bant genişliği tüketimi, video tazelenme hızı, veri iletim süresi ve SAGAT (Situation Awareness Global Assessment Technique) parametreleri üzerinden karşılaştırmalarını yapmışlardır. Sonuç olarak teleoperasyon protokolü olarak kullanıldıklarında WebSocket'in HTPP'den çok daha üstün olduğu sonucuna varmışlardır.

Amaran ve arkadaşları [6] yaptıkları deneylerde CoAP ve MQTT-SN protokollerini UDP üzerinden test etmişlerdir. İstemci tarafında Rasperry Pi B (ARMv6 işlemcili) cihazını, sunucu için ise Intel Xeon işlemcili bir cihaz kullanmışlar ve veri iletişimini kablo bağlantılı bir LAN bağlantısı üzerinden gerçekleştirerek internet'ten yalıtılmış bir ortam sağlayarak deneylerini gerçekleştirmişlerdir. Sonuç olarak MQTT-SN protokolünün ortalama mesaj iletim zamanı parametresine göre CoAP'tan daha üstün olduğunu tespit etmişlerdir.

Sarafov [7] yaptığı çalışmada WebSocket, MQTT QoS 0 ile QoS 2 ve CoAP protokolünün performans analizlerini yapmıştır. Deneylerinde yerel bir WiFi üzerinden gerçekleştirmiş ve istemci tarafında Rasperry Pi B, sunucu tarafında ise i7 işlemcili bir dizüstü bilgisayar kullanmıştır. Sonuç analizinde matematiksel bir metot kullanarak bu protokollerin çeşitli paket kayıp oranı durumlarında gösterdikleri iş çıkarma oranlarını kıyaslamış ve protokollerin çerçeve boyutlarının deney sonuçlarına etkisini incelemiştir.

Naik [8] yaptığı karşılaşmada MQTT, CoAP, AMQP ve HTTP protokollerini ve özelliklerini anlattıktan sonra bu protokolleri mesaj ve çerçeve boyutları, enerji tüketimi ve kaynak gereksinimi, bant genişliği kullanımı ve gecikme, kararlılık ve

desteklenen platformlar, güvenlik, IoT'ye uygunluk ve standartlaşma özelliklerine göre grafiksel olarak karşılaştırmıştır. Çalışmasında herhangi bir deney ortamı ve çıktısı sağlamamış olsada edinimlerini birçok bilimsel araştırmaya dayandırarak sonuca varmıştır.

Gültunca [9] yaptığı çalışmada MQTT, CoAP, XMPP ve AMQP protokollerini tanıttıktan sonra bir Raspberry Pi 3 cihazı ve birde MacOSX işletim sistemine sahip bir dizüstü bilgisayarı TP-Link 1 Gbit bant geliştiğine sahip switch ile CAT5 kablolu bağlantı ile bir LAN oluşturarak hazırladığı test düzeneğinde bu protokoller üzerinden 50, 100, 150, 200 ve 250 bayt yük boyutlarıyla her seferde 1000 mesaj göndererek test etmiştir. Deneylerinde istemci ve sunucu tarafında Mousquitto, RabbitMQ, ejabbered XMPP sunucusu gibi güçlü yazılımlar kullanmış ve bu yazılımların deney sonuçlarına etkilerinden ve kıyaslamalarından bahsetmiştir. Yaptığı deneyler ile bu protokollerin verilen işi yerine getirme zamanlarını kıyaslamıştır. XMPP deneylerinde sadece 50, 100 ve 150 baytlık yük boyutuna sahip mesajlar ile yaptığı denemelerden sonuç alabildiğinden bahsetmiş, yüksek yük boyutlu denemelerde kullandığı XMPP sunucusunun hata verdiğinden bahsetmiştir.

1.2. Tez Çalışmasının Katkıları

Literatürde, IoT'de kullanılan haberleşme protokollerinin performans analizi üzerine yapılmış olan birçok deneysel çalışma internet'ten yalıtılmış yerel ağlarda, sınırlı cihaz sayılamayacak kadar güçlü cihazlar ile, yapımcıları farklı olan ve performanslarının birbirlerinden farklı olduğu hazır bilgisayar yazılımları veya online IoT araçlarıyla, az sayıda yük boyutu seçeneği veya az sayıda mesaj ile yapılmış ve az sayıda parametreyi çıktı olarak vermektedirler.

Bu çalışmada ise IoT'de sıkça kullanılan MQTT, CoAP ve WebSocket haberleşme protokollerinin performans analizleri deneysel bir çalışma ile sunulmaktadır. Deneyler IoT uygulamalarının asıl çalışma alanı olan mobil bir kablosuz ağ üzerinden gerçekleştirilerek protokollerinin gerçek bir mobil internet ortamında gösterdikleri performans incelenmektedir. Mesaj üreten ve gönderen cihaz olarak gerçekten sınırlı

bir cihaz sayılabilecek bir cihaz seçilerek protokollerin sınırlı işlem kapasitesindeki performansları incelenmektedir. Mesajı alan/sunucu kısmında ise tez çalışması kapsamında geliştirilen yazılım ile harici yazlımlar veya online araçların etkisinden yalıtılmış olarak elde edilen performans sonuçları sunulmaktadır. 8, 16, 32, 64, 128, 256, 512 ve 1024 bayt yük seçenekleri ve her bir seçenek için 10 bin mesaj iletimi yapılarak protokollerin farklı yük boyutlarındaki performansları ve çalışma kararlılıkları değerlendirilmektedir. Ayrıca yukarıda sayılan bu durumların her biri için sınırlı cihazın harcadığı enerji tüketim miktarı da tespit edilmektedir.

1.3. Motivasyon

IoT uygulamalarının birçoğu sınırlı kapasite ve işlem gücüne sahip cihazlarda çalışmak üzere geliştirilir. Sınırlı işlem gücüne sahip bu cihazlar gerçekleştirilmek istenen senaryo için sadece gerekli olan bileşenleri içerdiğinden düşük maliyetli olurlar ve yapısal olarak boyutları küçük olup kullanılan alandan tasarruf sağlarlar.

Sınırlı cihazların verilen görevi yerine getirebilmek için üzerine yüklenen yazılımı çalıştıracak sınırlı işletim sistemleri vardır. Sınırlı cihazların birçoğu Windows, Apple IOS, Linux vb. yüksek kapasite ve işletim gücü gerektiren işletim sistemlerini çalıştıramazlar. Bunun yerine bu cihazlarda kullanılmak üzere tasarlanan düşük kapasite ve işlem gücü gerektiren ve farklı ek modülleri (sensörler, GPS, wireless modülleri vb.) destekleyen işletim sistemleri geliştirilmiştir. Bu işletim sistemlerinden yaygın kullanıma sahip bazıları Arduino, Android, RIOT, CONTIKI olarak sayılabilir. IoT' de kullanılan haberleşme protokollerinin birçoğu ise sınırlı cihazlarda düşük bant genişliği kullanımı ve düşük enerji tüketimi amaçlanarak tasarlanmıştır. İstenen bilgiyi alıcıya iletebilmek için farklı veri iletim modellerine sahip bu protokoller yapısal ve temel mantık olarak birbirlerinden farklılaşırlar. Bu farklılaşmalar genel olarak istenen verinin alıcıya en hızlı, düşük maliyetli ve güvenli iletilmesi arasında yapılan tercihlerden ileri gelmektedir.

IoT haberleşme protokollerinin veri iletim modelleri geliştiricileri tarafından yeterli düzeyde açıklansa da bu protokollerin veri iletiminde enerji tüketimi, iş çıkarma oranı,

gecikme süreleri üzerinden veri iletimine yaptığı etki istatistiksel karşılaştırmalar olarak sunulmamaktadır. Bu yönde yapılan birkaç araştırma ise genel olarak ideal ortamda yani internet'ten yalıtılmış yerel ağlarda, yüksek bant genişliğine sahip yapılan testten başka trafik içermeyen kablolu/kablosuz bağlantılarda veya önceden geliştirilmiş, hazır, üst düzey işletim sistemlerinde çalışan programlarda yapılan testlerin sonucunu paylaşmaktadır. Gerçek dünya şartlarında yapılmayan bu testlerin sunduğu bilgilerin güvenirliği kesin değildir.

Bu tez çalışmasında ise, IoT'de kullanılan protokollerin performans karşılaştırma testleri Arduino işletim sisteminde ESP8266 modülünde çalışan sınırlı bir cihaz olan Wemos D1 cihazı ile mobil bağlantı kullanarak -özellikle aynı anda milyonlarca kişinin kullandığı bir ağ- IoT protokolüne özgü veri iletim modellerinin tüm özelliklerini en iyi derecede kullanmaya olanak sağlayacak bir senaryo ile gerçekleştirilmektedir. Bu testlerin sonucunda IoT protokollerinin iş çıkarma oranı, enerji tüketimi ve mesajlar arasında gerçekleşen toplam gecikme sürelerinin analizi yapılmaktadır. Bu verilerin karşılaştırılması grafiksel olarak gerçekleştirilmektedir. Yapılan testler sonucunda elde edilen verilerle gerçek dünya şartlarında IoT protokollerinin veri iletimine yaptığı etki gösterilmeye çalışılmaktadır. Bu sayede IoT uygulamalarında kullanılacak veri iletim protokolü hakkında özellikle geliştiricilerin, gerçek dünya şartlarına göre fikir edinmesi sağlanmaya çalışılmaktadır.

1.4. Tezin Organizasyonu

Bölüm 2'de Nesnelerin interneti kavramından genel olarak bahsedilerek kullanıldığı yerlerden örnekler verilmektedir. IoT uygulamalarında sıkça kullanılan MQTT, MQTT-SN, CoAP, AMQP, WebSocket, XMPP, DDS, SOAP ve REST haberleşme protokolleri incelenmekte ve bu protokollerin özellik karşılaştırmaları bir tablo halinde sunulmaktadır. Bölüm 3'te MQTT, CoAP ve WebSocket protokolleri için oluşturulan deney düzeneği tanıtılmaktadır. Deneylerde elde edilen sonuçlar her bir protokol için ayrı ayrı değerlendirilmekte ve ardından bu protokollerin başarım karşılaştırmaları yapılmaktadır. Bölüm 4'te ise tez çalışması kapsamında elde edilen sonuçlar değerlendirilmekte ve gelecekte bu çalışma alanına yönelik önerirler sunulmaktadır.

BÖLÜM 2. NESNELERİN İNTERNETİ VE UYGULAMA PROTOKOLLERİ

2.1. Nesnelerin İnterneti

Kavramsal olarak Nesnelerin İnternet'i, genel olarak nesneler fikrini, özellikle de RFID, kablosuz yerel ağ, geniş alan ağı veya başka yollarla internet üzerinden okunabilen, tanınabilir, konumlandırılabilir, adreslenebilir ve/veya kontrol edilebilir gündelik nesneleri ele alır [10]. IoT nesneleri genel olarak düşük kapasite ve işlemci gücüne sahip, belli bir amacı gerçekleştirmek üzere yalnızca gerekli bileşenleri içeren sınırlı cihazlardır. Bunların yanında bilgisayarlar gibi gelişmiş kapasite ve işlemci gücüne sahip cihazlarda IoT nesneleri alanında sayılabilir ve genellikle sunucu veya aracı (broker) görevini üstlenirler.

IoT nesneleri; akıllı telefonlar, bio-nano nesneler, vücut sensörleri, akıllı etiketler, giyilebilir cihazlar, gömülü sistemlerde nesneler, genel maksatlı sensörler, geleneksel elektronik aygıtlar vb. çok geniş bir uygulama alanına yayılmış cihazlardır.

Nesnelerin interneti (Internet of Things, IoT), farklı iletişim protokolleri ile birbirleriyle haberleşebilen, algılama ve veri işleme kabiliyetine sahip nesnelerden oluşan küresel bir ağdır [1]. IoT ağını oluşturan nesnelerin bellek, işlem gücü, batarya gibi sınırlı kaynaklara sahip olduğu göz önüne alındığında, IoT bileşenlerinin haberleşmesi için IoT sınırlamalarını gözönünde bulunduran, düşük maliyetli (lightweight) haberleşme protokolleri önem kazanmaktadır.

IoT nesneleri değişen aralıklarla ve/veya sürekli veri üretir. Zamana bağlı olarak sürekli üretilen ve biriken bu veriler; artan bant genişliği, depolama birimi, işlem kapasitesi gibi ihtiyaçlar doğurur. IoT cihazlarının haberleşmesinde oluşan bu ihtiyaçları en aza indirgemek üzere çeşitli iletişim modellerine sahip IoT haberleşme

protokolleri geliştirilmiştir. Bu protokollerden MQTT, MQTT-SN, CoAP, AMQP, WEBSOCKET, XMPP, SOAP, DDS, REST protokolleri günümüzde popüler ve standart halini almış olanlardan bazılarıdır. Bunlara ek olarak farklı ihtiyaç ve amaçları karşılamak üzere birçok protokolde geliştirilmiştir.

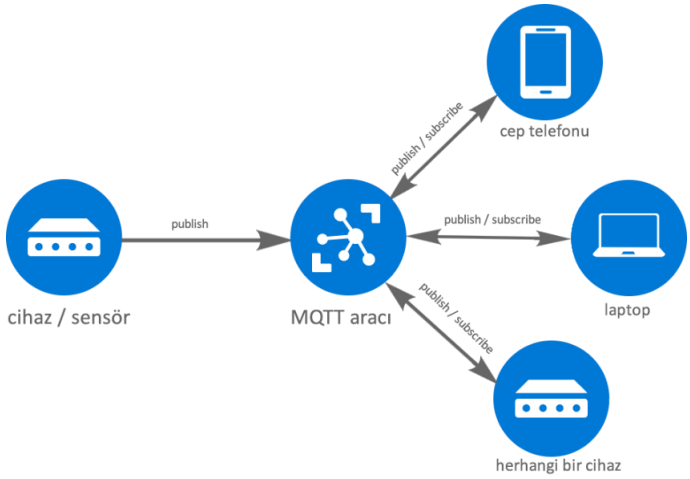
İlerleyen bölümlerde bu protokollerin genel tanıtımı, iletişim modeli, kullanım alanları vb. özelliklerinden bahsedilmektedir.

2.2. MQTT – Message Query Transport Telemetry

Mesaj Kuyruk Telemetri Ulaştırma (Message Queue Telemetry Transport, MQTT) protokolü 1999 yılında IBM ve Arcom (Eurotech) tarafından tanıtılmasına karşın 2013'te OASIS tarafından standartlaştırılmış ve 2016 yılında ISO tarafından ISO/IEC 20922:2016 olarak onaylanmış bir IoT haberleşme protokolüdür. MQTT protokolünün IoT uygulamaları için geliştirilen v5.0 ile kablosuz algılayıcı ağlar için geliştirilen MQTT-SN (MQTT for Sensor Network) olmak üzere iki versiyonu bulunmaktadır. Düşük maliyetli, açık kaynak kodlu, basit ve kolay uygulanabilirliği sayesinde IoT uygulamalarında yaygın olarak tercih edilmektedir [11].

2.2.1. MQTT Mimarisi

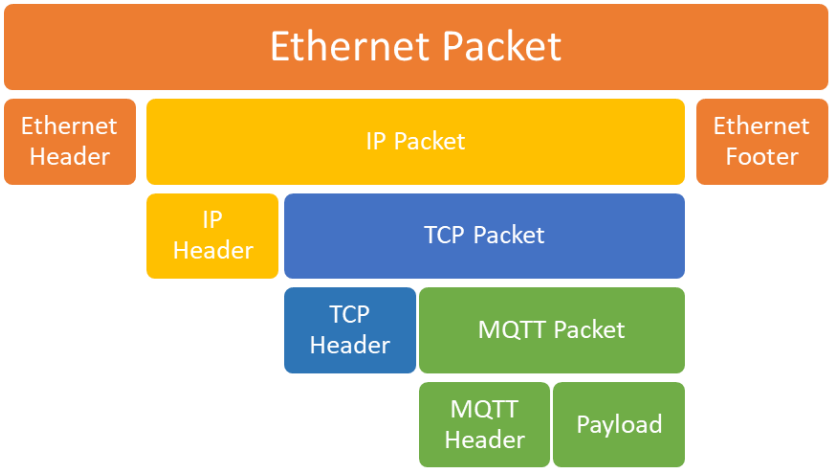
MQTT istemci-sunucu yayım/abone (Client-Server publish/subscribe) mesajlaşma protokolüdür. MQTT, yayımcı (publisher), abone (subscriber) ve sunucu (broker) olmak üzere üç temel bileşenden oluşur. Yayımcı, üretilen verinin kaynağıdır ve amacı üretilen veriyi aboneye göndermektir. Veri, Mesaj ve Konu (topic) olarak adlandırılan iki temel bileşenden oluşur. Aboneler yayılan veriyi analiz etmek ve işlemek için alan hedef kullanıcılardır. MQTT sunucu ise yayımcı ve aboneler arasında konuya göre verinin dağıtımını sağlar. Bir başka değişle, MQTT sunucu yayımcıdan abonelere ulaştırılmak istenen verileri depolar, filtreler ve iletir [11]. Şekil 2.1.'de MQTT haberleşme modeli görülmektedir.



Şekil 2.1. MQTT haberleşme modeli [11].

MQTT protokolü, IoT cihazların bulut (internet) ile bağlantılarını sağlamak üzere TCP/IP protokolü üzerine inşa edilmiştir. MQTT haberleşmesi TCP protokolünün üçlü el sıkışma yapısında çalışır. TCP üzerinden gönderilen herhangi bir mesaj TCP paketi olarak gönderilir.

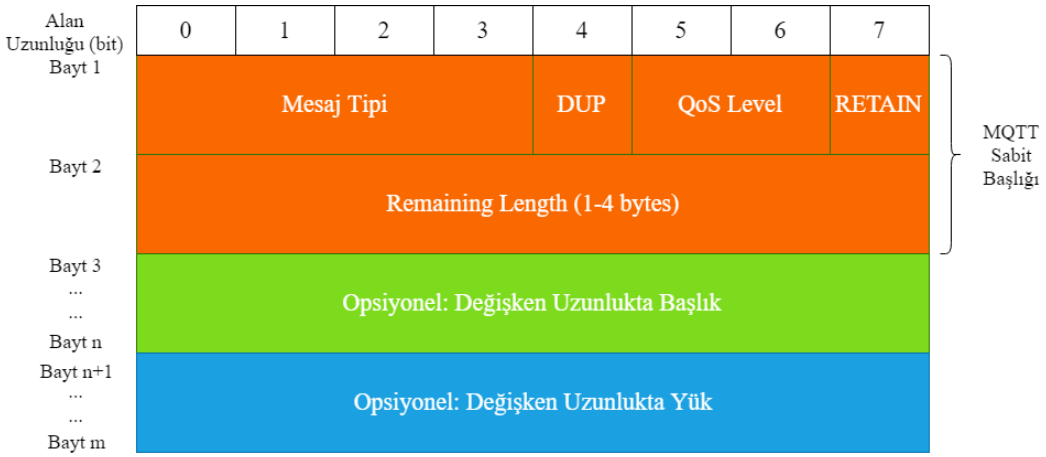
Gönderilecek olan mesaja önce TCP paket başlıkları eklenir ve mesaj TCP paketi özelliği kazanır. Daha sonra sırasıyla IP ve Ethernet başlıkları da eklenerek mesaj alıcıya gönderilir. MQTT ile gönderilen bir mesaj Şekil 2.2.'de görülen katmanlardan geçerek en son Ethernet Paketi olarak alıcı bilgisayara gönderilir. Burada MQTT'nin yönetebildiği alan en alttaki katmandır. Buradan sonrası ise işletim sisteminin kontrolündedir.



Şekil 2.2. Bir MQTT mesajının paketlenme hiyerarşisi [11].

2.2.2. Paket Yapısı

Şekil 2.3.'te MQTT paket formatı görülmektedir. Uygulama ihtiyacına göre paket boyutu istenen uzunluğa ayarlanabilmektedir. Ancak en az 2 bayt olacak şekilde sabit başlık (header) alanı bulunmaktadır. Message Type alanında, Tablo 2.1.'de verilen mesaj türleri tanımlanmaktadır. DUP, mesajın kopyalandığını gösteren bayraktır (duplicate). QoS Level, yayımlanan mesajın hangi QoS seviyesinde iletileceğini gösterir. Retain, son alınan yayımcı (publish) mesajı sunucuya bildirir ve yeni aboneye ilk mesaj olarak iletir. Remaining Length, mesajın kalan boyutunu gösterir. Header (Başlık) kısmı opsiyonel olup, protokol versiyonu vb. bilgileri içerir. Payload (Yük) kısmı ise konu (topic), mesaj, kullanıcı, şifre vb. bilgileri içerir.



Şekil 2.3. MQTT paket yapısı [11].

MQTT'de veri iletimi; bağlantı kurulması, kimlik doğrulama, iletim ve bağlantı sonlandırma olmak üzere dört aşamadan oluşur. Bu aşamalar MQTT paketleri tarafından yönetilir. Bir kere bağlantı kurulduktan sonra, bağlantı sonlandırılana kadar istenilen sayıda mesaj gönderilebilir. Tablo 2.1.'de MQTT paket türleri listelenmektedir.

Tablo 2.1. MQTT paket türleri [11]

Paket Adı	Değer	Akış Yönü	Tanım
Rezerve	0	Forbidden	Reserve edilmiş
CONNECT	1	İstemci → Sunucu	İstemcin Sunucuya bağlanma talebi
CONNACK	2	Sunucu →İstemci	Connect acknowledgment (onay)
PUBLISH	3	İstemci ↔ Sunucu	Publish mesajı
PUBACK	4	İstemci ↔ Sunucu	Publish geri bildirim
PUBREC	5	İstemci ↔ Sunucu	Publish alındı
PUBREL	6	İstemci ↔ Sunucu	Publish gönderildi
PUBCOMP	7	İstemci ↔ Sunucu	Publish tamamlandı
SUBSCRIBE	8	İstemci → Sunucu	İstemcinin abonelik talebi
SUBACK	9	Sunucu → İstemci	Subscribe geri bildirim
UNSUBSCRIBE	10	İstemci → Sunucu	Unsubscribe talebi
UNSUBACK	11	Sunucu → İstemci	Unsubscribe geri bildirim
PINGREQ	12	İstemci → Sunucu	PING talebi
PINGRESP	13	Sunucu → İstemci	PING yanıtı
DISCONNECT	14	İstemci → Sunucu	İstemci bağlantıyı sonlandırıyor
Rezerve	15	Forbidden	Reserve edilmiş

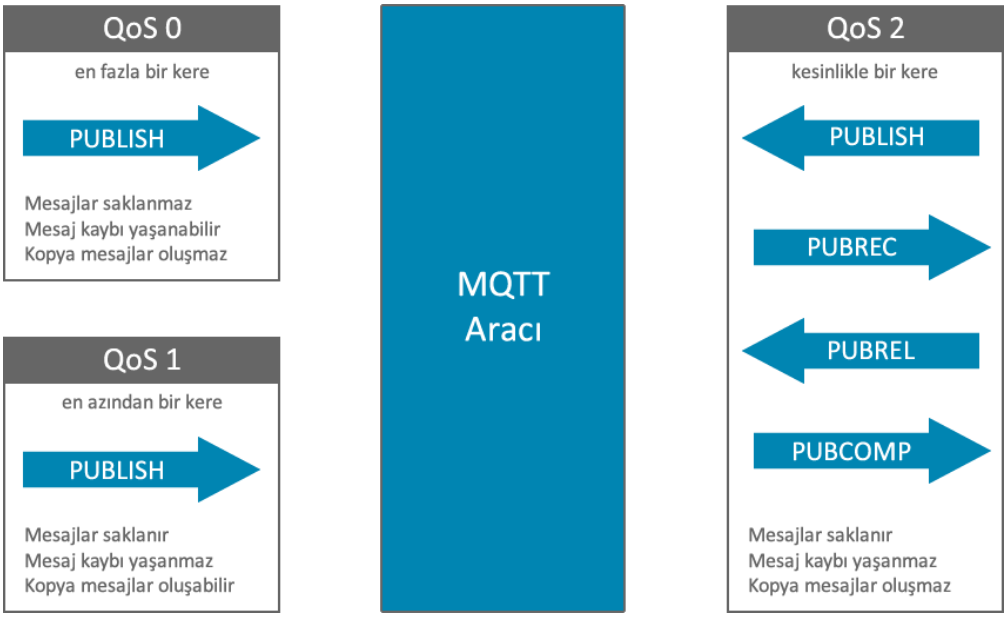
2.2.3. Servis Kalitesi Desteği

Şekil 2.4.'te MQTT mesaj yapısında desteklenen 3 farklı servis kalitesi seviyesi görülmektedir.

QoS 0: En fazla bir defa (Best Effort), yayıncı mesajı sunucuya en fazla bir kez yayınlar. Mesaj sunucuda ve diğer uçlarda saklanmaz ve mesajın ulaşması kontrol edilmez. Mesaj gönderilir fakat herhangi bir nedenden ötürü (bağlantı kopması vb.) mesaj adrese ulaşmayabilir/iletilmeyebilir. En düşük trafiğin olduğu QoS seviyesidir. QoS 0, Geri Bildirimsiz Hizmet (Unacknowledged Service) olarak ta bilinir. İletim aşamasında PUBLISH paket tipi kullanılır.

QoS 1: En azından bir defa (At Least Once), mesajın en az bir kere iletileceğini garanti eder. Ancak mesaj birden fazla iletilebilir. Yayıncı alıcıya mesajı bir kez gönderir. Alıcı bu mesajı aldığında yayıncıya bir PUBACK paketi ile alındığına dair bir geri bildirim gönderir. Yayıncı gönderdiği mesajı geri bildirim alana kadar saklar. Eğer yayıncı makul bir zaman aralığında geri bildirim almazsa aynı mesajı bu sefer "DUP (Duplicate)" bayrağını 1'e kurarak tekrar gönderir. Bu durumda alıcıda aynı mesajın birçok kopyası oluşabilir. QoS 1 seviyesi Geri Bildirimli Hizmet (Acknowledged Service) olarak da bilinir. İletim aşamasında PUBLISH ve PUBACK paketleri kullanılır.

QoS 2: Kesinlikle bir defa (Exactly Once), mesajın kesinlikle bir defa iletileceği garanti edilir. En güvenli ancak en yavaş servis kalitesi seviyesidir. Alıcı QoS 2 seviyeli bir PUBLISH paketi alırsa bu paketi uygun şekilde işler ve yayıncıya bir PUBREC (publish received – yayın alındı) paketi ile geri dönüş yapar. PUBREC paketi bir paket tanımlayıcı (Packet Identifier - packetId) içerir ve alıcı bu "packetId" değerini bir PUBCOMP paketi gönderene kadar saklar. Bu işlem aynı mesajın iki kere iletilmesini önlemek için gereklidir. Yayıncı alıcıdan PUBREC paketini aldığında karşı tarafın PUBLISH mesajını aldığını anlar ve herhangi bir hata durumuna karşı sakladığı bu PUBLISH mesajını siler. Yayıncı alıcıdan aldığı PUBREC paketini saklar ve alıcıya bir PUBREL (publish release) paketi gönderir. PUBREL paketi PUBREC paketi ile aynı "packetId" değerine sahip olur. Alıcı yayıncıdan PUBREL paketi aldığında ilgili PUBLISH için depoladığı tüm durum bilgilerini siler ve yayıncıya bir PUBCOMP (publish complete – yayın tamamlandı) paketi gönderir. Yayıncı alıcıdan PUBCOMP paketini aldığında ilgili PUBLISH için depoladığı tüm durum bilgilerini siler. Bu döngü tamamlandığında her iki tarafta mesajın iletildiğinden emin olur ve yayıncı taraf bunu bilir. Herhangi bir mesaj kaybı yaşandığında yayıncı taraf makul bir zaman aralığında hali hazırda sakladığı bu mesajı tekrar gönderir.

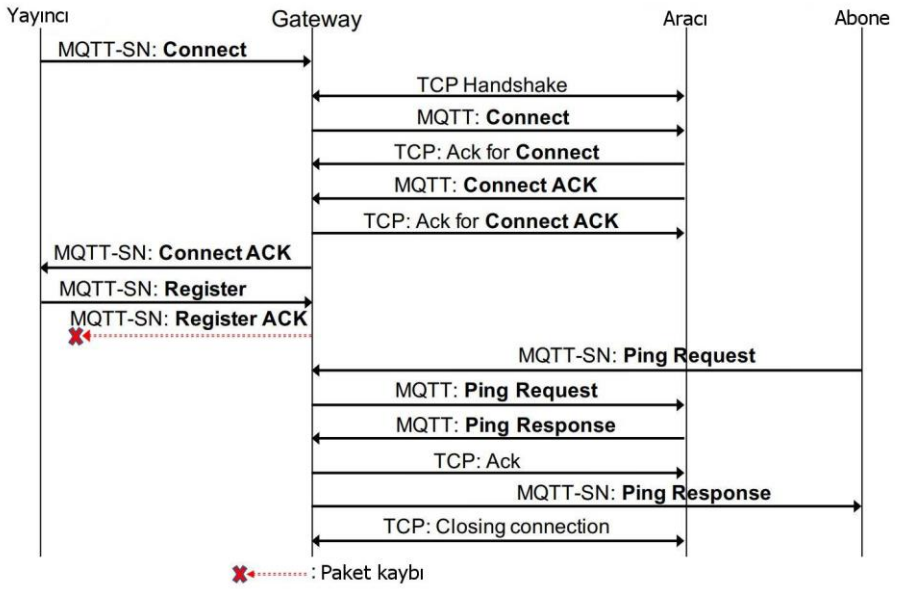


Şekil 2.4. MQTT servis kaliteleri

2.3. MQTT-SN

MQTT genel olarak düşük enerji tüketimi ve bant genişliği gereksinimi ile yüksek ölçeklenebilirlik ve oldukça düşük başlık boyutu sebebiyle IoT uygulamalarında sıkça tercih edilmektedir. Ancak MQTT sıralı ve kayıpsız bağlantı kapasitesi sunan bir TCP/IP bağlantısına ihtiyaç duyar. Ancak bu yapı basit, az yer kaplayan ve düşük maliyetli sensör cihazları için oldukça kompleks bir yapıdır. Bu meseleyi çözümlemek üzere kablosuz ağlar için MQTT-SN geliştirilmiştir. MQTT-SN kablosuz haberleşme ortamının özelliklerine bürünmüş bir MQTT versiyonu olarak düşünülebilir [12].

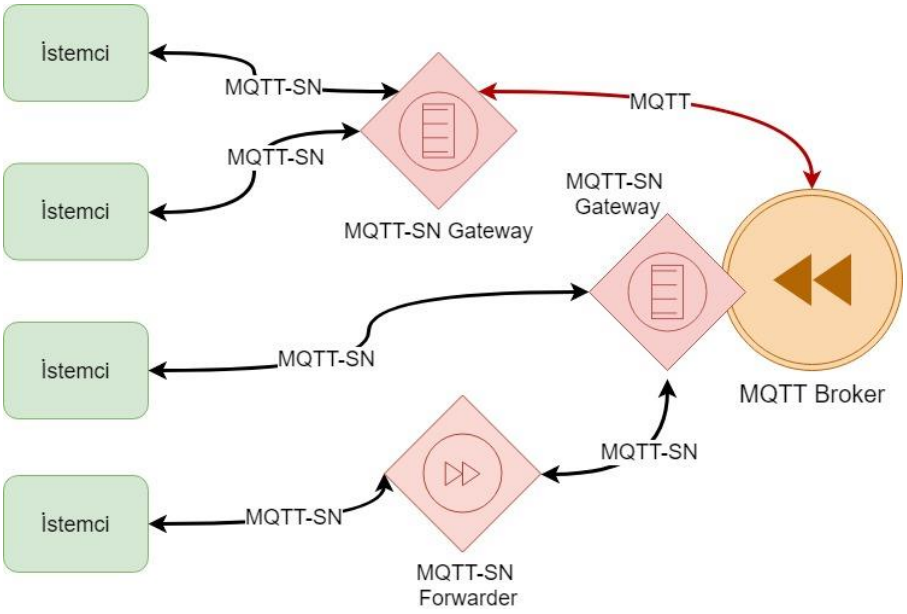
MQTT'nin aksine MQTT-SN sisteminde IoT nesneleri bir MQTT-SN protokolünü kullanarak bir geçide (gateway) / aracı cihaza kablosuz bağlantı üzerinden bağlanırlar. Bu geçit cihazı ise aracıya (broker) MQTT protokolünü kullanarak kablolu bağlantı üzerinden bağlanır. Diğer bir fark ise MQTT-SN protokolünün UDP üzerinden haberleşmesidir. UDP, kablosuz bağlantılarda TCP'den daha hızlı, daha basit ve daha hafif kod yükü getirdiğinden sensör uygulamalarına daha uygundur [13]. MQTT-SN mesaj parçalanmasını ve adrese ulaşınca yeniden birleştirilmesini desteklemez. Şekil 2.5.'te bir MQTT-SN paket iletimi şematize edilmektedir [14].



Şekil 2.5. MQTT-SN mesaj iletim örneği [14].

2.3.1. MQTT-SN Mimarisi

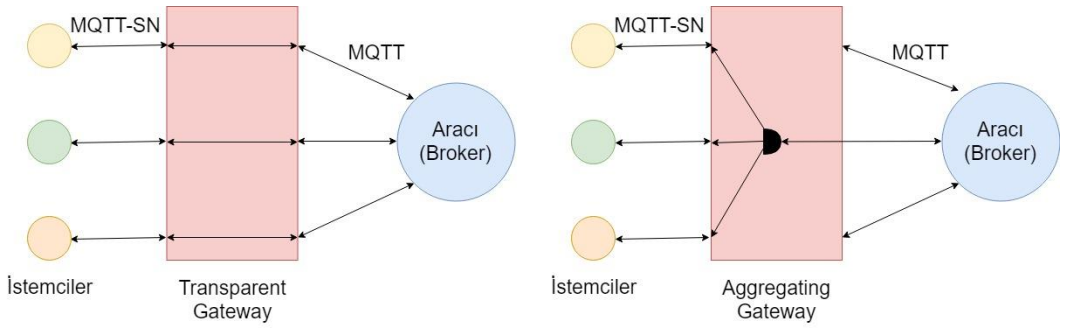
Şekil 2.6.'da MQTT-SN mimarisi görülmektedir. MQTT-SN mimarisi, istemciler, ağ geçitleri (gateway) ve ileticiler (forwarders) olmak üzere üç farklı mimariden oluşur. İstemciler MQTT-SN protokolünü kullanarak gateway'ler üzerinden bir MQTT Broker'a (Aracı) bağlanırlar. Gateway'in tek başına bir ağ oluşturduğu durumda MQTT Broker ile MQTT-SN Gateway arasındaki haberleşmede MQTT protokolü kullanılır. Buradaki Gateway'in amacı MQTT ile MQTT-SN arasında tercümanlık yapmaktır. İstemciler, ulaşmak istedikleri gateway'in kendi ağlarına doğrudan bağlı olmadığı durumlarda MQTT-SN Forwarder üzerinden bu gateway'e ulaşabilirler. Forwarder'lar basit olarak istemcilerden gelen paketleri kapsülleyip içeriğini değiştirmeden gateway'e gönderirler. Ters istikamette ise gateway'den gelen paketleri kapsülden çözüp yine içeriğini değiştirmeden istemcilere iletirler [15].



Şekil 2.6. MQTT-SN Mimarisi

MQTT-SN protokolü temel olarak Şekil 2.7.'de gösterilen iki farklı mimariye sahiptir. Şeffaf geçit (Transparent Gateway) mimarisinde bağlı olan her istemci için gateway MQTT sunucusuna bir bağlantı kurar ve bu bağlantının devamını sağlar. Bu bağlantı türü bir çeşit uçtan-uca bağlantı türüdür ve gatewayin yaptığı tek iş yönlendirmedir. Yani istemci ve sunucu arasında şeffaf bir iletim sağlanmış olur. Hibrit mimaride genel olarak gateway sayısı sensor sayısından az olur. Gateway ve sunucu arasında bağlı olan istemci sayısı kadar bağlantı oluşur. Transparent Gateway modelinin uygulanması Aggregating modeline göre daha kolaydır. Ancak MQTT sunucusunun her bir istemci için ayrı bir bağlantıyı desteklemesi gerekir [15].

Aggregated (Birleştirilmiş) gateway mimarisinde ise tüm istemcilerden gelen verileri toplayıp sunucuya tek bir bağlantı üzerinden aktaran tek bir gateway bulunur. MQTT-SN istemcisinden gelen tüm mesaj iletimleri Aggregating Gateway'de son bulur. Gateway daha sonra hangi bilginin sunucuya iletileceğine karar verir. Uygulanması daha zor olsada sunucu ve gateway arasında yalnızca bir bağlantının oluşması sunucu üzerindeki yükün azalmasını sağlar [15].



Şekil 2.7. MQTT-SN'de Transparent ve Aggregating Gateway mimarileri.

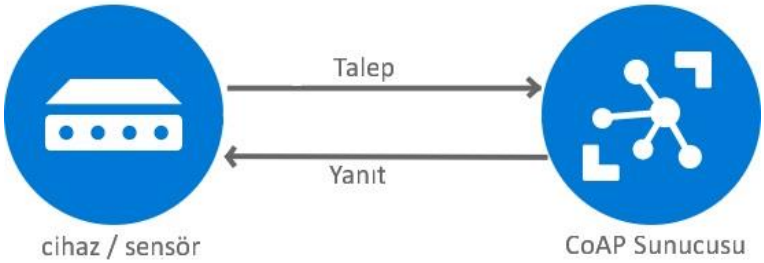
2.4. Kısıtlı Uygulama Protokolü (CoAP)

Kısıtlı Uygulama Protokolü (Constrained Application Protocol, CoAP) HTTP protokolü üzerinde çalışan REST (Representational State Transfer) temelli bir ağ transfer protokoldür. CoAP'ın, REST'in aksine UDP portları üzerinde çalışması onu IoT uygulamaları için daha uygun hale getirmiştir.

CoAP, IETF (Internet Engineering Task Force – İnternet Mühendisliği Görev Gücü) tarafından tasarlanmış bir uygulama katmanı protokolüdür. Kısıtlı cihazlar ve ağlarda çalışmak üzere tasarlanan bir web haberleşme protokolüdür. UDP üzerinde geliştirilmiştir ve REST modelinde çalışır. Bu yüzden HTTP ile benzer şekilde çalışır. CoAP, uygulama uç birimleri arasında istek/cevap modelinde çalışma imkânı sağlar; URI'ler üzerinden servisler ve kaynakların keşfi için yerleşik desteğe sahiptir.

2.4.1. CoAP Mimarisi

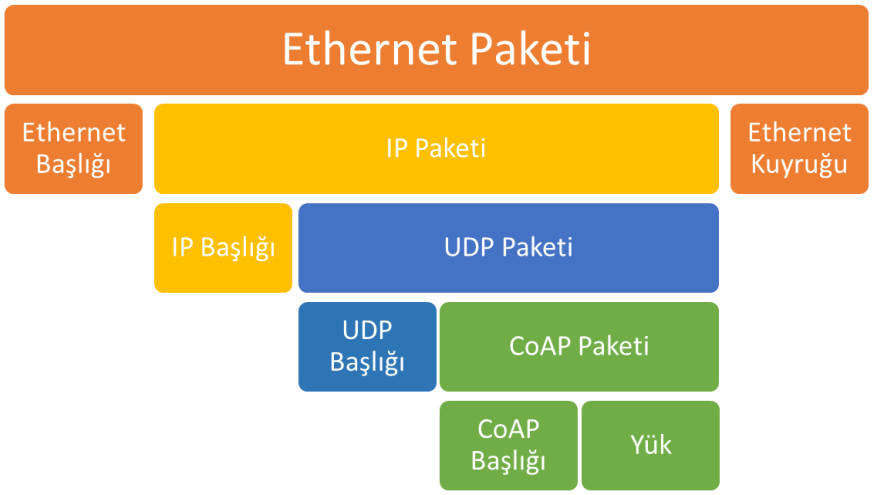
CoAP bir istemci-sunucu istek/yanıt (Client-Server request/response) mesajlaşma protokolüdür. Şekil 2.8.'de şematize edildiği üzere CoAP iletişimi istemci ve sunucu olmak üzere iki temel bileşenden oluşur. İstemci, üretilen verinin kaynağıdır ve amacı üretilen veriyi sunucuya göndermektir. İstemci bu verileri REST metotlarına benzeyen Get, Post, Put veya Delete CoAP metotları ile sunucuya gönderir. Ve sunucu da -eğer mesajlaşma tipi geri bildirimli ise- istemciye uygun bir cevap gönderir [16].



Şekil 2.8. CoAP Protokolü haberleşme düğümleri

CoAP protokolü, IoT cihazların bulut (internet) ile bağlantılarını sağlamak üzere UDP protokolü üzerine inşa edilmiştir. CoAP haberleşmesinde veriler UDP protokolü üzerinden datagramlar şeklinde gönderilir.

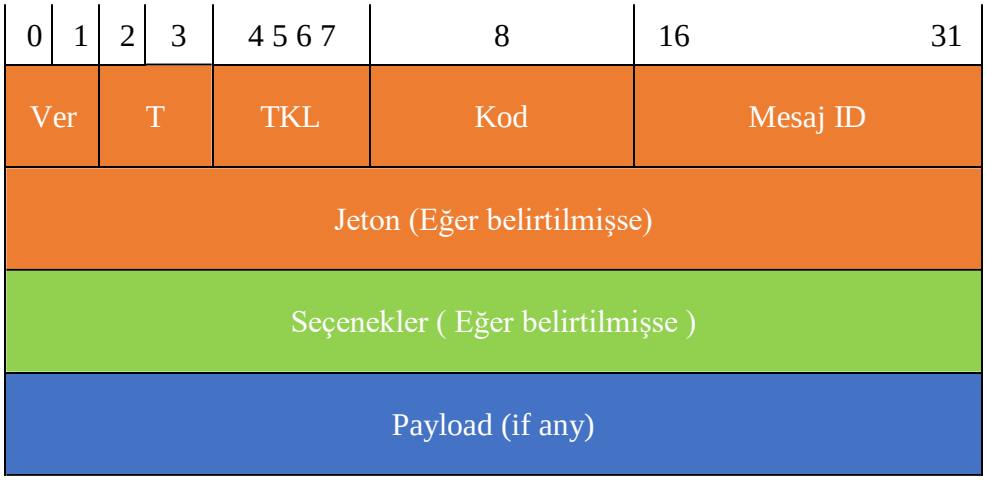
Gönderilecek olan mesaja önce UDP paket başlıkları eklenir ve mesaj UDP paketi özelliği kazanır. Daha sonra sırasıyla IP ve Ethernet başlıkları da eklenerek mesaj alıcıya gönderilir. CoAP ile gönderilen bir mesaj Şekil 2.9.'da görülen katmanlardan geçerek en son Ethernet Paketi olarak alıcı bilgisayara gönderilir. Burada CoAP'ın yönetebildiği alan en alttaki katmandır. Buradan sonrası ise işletim sisteminin kontrolündedir.



Şekil 2.9. Bir CoAP mesajının paketlenme hiyerarşisi

2.4.2. CoAP Paket Yapısı

Şekil 2.10.'da CoAP paket formatı görülmektedir. İlk kısım her mesaj için sabit olan başlık kısmıdır. Sonraki kısımlar ise tercihe bağlıdır. Bunlardan Jeton (Token) alanı istek ve yanıtları ilişkilendirmek için kullanılır ve 0 – 8 bit arası bir uzunlukta olabilir. Ver alanı CoAP'ın o an kullanılan versiyonunu belirtir. 2 bitlik T (Type) alanı mesajın tipini ifade eder. Bunlar onaylanabilir (confirmable-0), onaylanamayan (nonconfirmable-1), Onay mesajı (acknowledgement - 2) veya reset (3) tipleri olabilir. TKL alanı ise 4 bitlik token uzunluğunu ifade eden bir alandır. Kod (Code) alanı 8 bitlik yanıt (response) mesajlarının sonuç kodunu ifade eden bir alandır. Bu alan HTTP yanıt kodlarının kolay bir biçimde ifade edilmesi için tasarlanmıştır. Uygulama kısmında 3 bit yanıtın sınıfını ve 5 bit yanıtın detayını ifade edecek şekilde kullanılır. (Örn: 2.02) 16 bitlik Mesaj ID alanı ise duplication denetiminde ve yanıtların eşleşmesinde kullanılır.



Şekil 2.10. Bir CoAP mesajının paket yapısı.

2.4.3. CoAP Çalışma Modları

CoAP protokolünün iki temel iletim modu vardır. Bunlar non-cofirmable (onaylanamaz) ve confirmable (onaylanabilir) mesajlaşma modlarıdır. Non-confirmable modunda tek yönlü ve onaylanma (acknowledgment) beklenmeden iletim sağlanır. Bu yüzden başarılı iletim bu modda garanti edilmez. Confirmable modunda

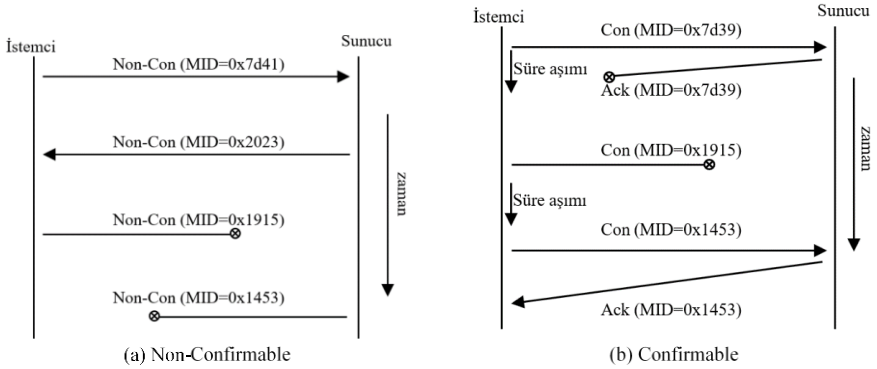
ise onaylanma alındığında mesajın en az bir kere iletildiği varsayılır.

1) Non-Confirmable İletim Modu

Non-confirmable veri iletimi Şekil 2.11. (a) daki gibi bir akış izler. Sunucu-istemci ya da istemci-sunucu arasında gerçekleşebilir. Mesaj bir kere gönderilir ve iletimin başarılı olduğu izlenmez.

2) Confirmable İletim Modu

Confirmable veri iletimi ise Şekil 2.11. (b) deki gibi bir akış izler. Mesaj gönderilir, önceden belirlenmiş bir timeout (süre aşımı) süresi içerisinde yanıt alınmazsa iletilmedi kabul edilir ve tekrar gönderilir. Aynı mesaj ID'sine sahip olan bir Ack (acknowledgement) paketi geldiğinde ise mesajın iletildiği doğrulanmış olur.



Şekil 2.11. CoAP mesaj iletim modları

2.4.4. CoAP Yanıt Tipleri

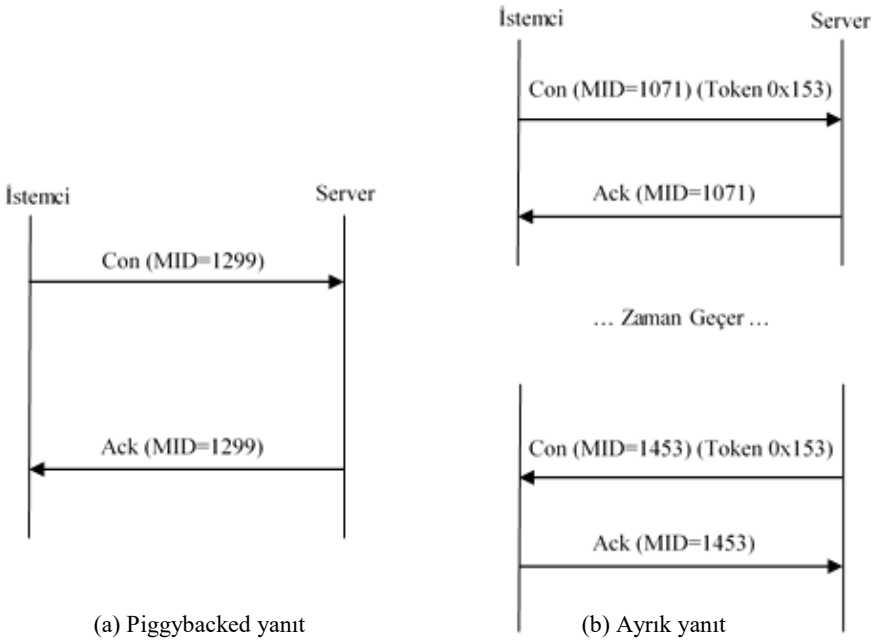
CoAP'ın gelen isteklere yanıt vermesi birkaç farklı şekilde olabilir. Senkron ve asenkron iletim mekanizmaları CoAP tarafından Piggybacked response ve Separate response (Ayrık Yanıt) isimleriyle tanımlanmıştır [17].

Çoğu durumda yanıt, doğrudan acknowledgement mesajı içerisinde taşınır (istek mesaj tipinin confirmable olmasıyla). Buna Piggybacked Response denir ve bu

doğrudan acknowledgement mesajında taşınan bir yanıttır.

Separate response iki CoAP mesajından oluşur. Bunlardan birincisi istemcinin mesajın iletilmediğine karar verip tekrar tekrar mesaj göndermesini önlemek için gönderilen bir ACK mesajıdır. İkincisi ise istemciye verilen cevabı içeren bir Confirmable (CON) mesajıdır [17].

Şekil 2.12. bir isteğe piggybacked (a) ve separate response (b) ile verilen yanıtları örneklemektedir.



(a) Piggybacked yanıt (b) Ayrık yanıt

Şekil 2.12. CoAP Yanıt tipleri

2.5. AMQP

Gelişmiş Mesaj Kuyruklama Protokolü (Advanced Message Queuing Protocol, AMQP), bir açık standart uygulama katmanı protokolüdür. İlk olarak 2006'da geliştirilmiş ve 2012 yılında bir OASIS[1] standardı haline gelmiştir. AMQP farklı platformlarda çalışabilirlik (interoperatibility), kararlılık (reliability) ve güvenlik (security) odaklı geliştirilmiştir. Farklı platformlarda çalışabilirlik özelliğine erişebilmek için kablo seviyesi (wire level) protokolü olarak geliştirilmiştir. Bu nedenle AMQP kullanılan bir haberleşmede, uygulamalar farklı yazılım dilleri ve

[1] http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-complete-v1.0-os.pdf

platformlarda yazılmış olsalar ve farklı teknolojik altyapılar kullanıyor olsalar dahi birbirlerine mesaj gönderebilirler.

AMQP, uygulamaların birer anlık mesajlaşma veya mail uygulamaları gibi birbirlerine mesaj göndermesi ya da almasına olanak sağlar. AMQP mesajların nasıl iletileceği ve bu iletim aşamasında kullanılan güvenlik, tutarlılık ve performans alanlarında çeşitli tercihler sunmasıyla diğer IoT protokollerinden farklılaşır [18].

Mesaj iletiminde kararlılığı sağlayabilmek adına AMQP'ye üç adet servis kalitesi (Quality of Servive - QoS) seçeneği eklenmiştir. Bunlar; en fazla bir kere (at most once), en az bir kere (at least once) ve kesinlikle bir kere (exactly once) seçenekleridir.

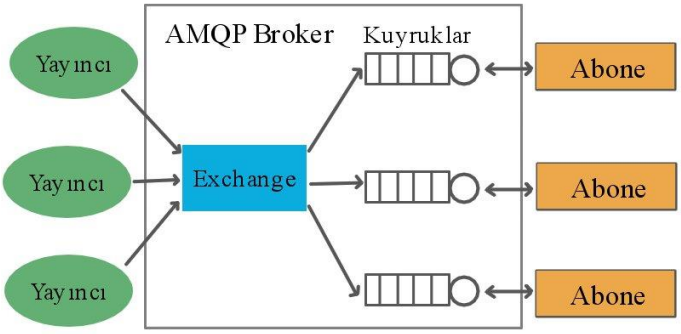
AMQP varsayılan haberleşme kanalı olarak TCP'yi, güvenlik için ise SSL/TLS ve SASL kullanır. SASL (Simple Authentication and Security Layer) basit yetkilendirme ve güvenlik katmanı anlamına gelir ve istemci veya sunucuların bir kullanıcı adı/parola ya da bir sertifika ile yetkilendirilmesine olanak sağlar. SSL/TLS ise mesaj yükünü şifrelemek için kullanılır [19].

2.5.1. AMQP Mimarisi

AMQP istek/yanıt ve yayıncı/abone mimarilerinden her ikisini de destekler. Güvenilir kuyruklama, başlık (topic) temelinde yayıncı/abone mimarisi, esnek yönlendirme (routing) ve toplu mesaj gönderimleri gibi birçok özellik sunar. AMQP haberleşmesi yayıncı (publisher) veya alıcının (consumer) verilen herhangi bir isimle bir takas (exchange) oluşturması ve bunu yayınlamasıyla başlar. Yayıncılar ve alıcılar bu exchange adını kullanarak birbirlerini keşfederler. Keşiften sonra alıcı bir kuyruk oluşturur ve exchange'e bağlanır. Exchange tarafından alınan mesajlar "binding" olarak isimlendirilen işlem aracılığıyla kuyruk ile eşleşmek zorundadır [8].

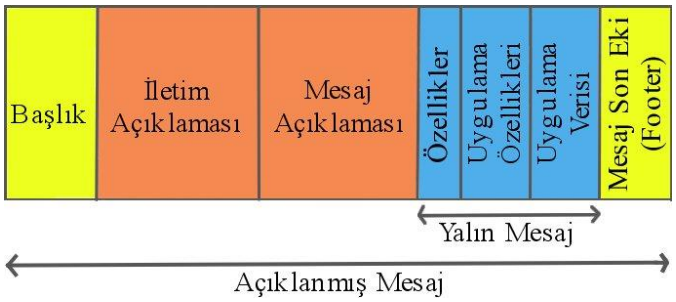
Şekil 2.13'te görüldüğü üzere AMQP haberleşmesi iki bileşenden oluşur. Bunlar Exchange'ler (takaslar) ve mesaj kuyruklarıdır. Exchange'ler mesajları uygun kuyruğa yönlendirmek için kullanılır. Mesajlar öncelikli olarak mesaj kuyruklarında depolanır

ve daha sonra alıcılara gönderilirler. Bu mekanizma uçtan uca çalıştığı gibi yayıncı-abone modelinde de çalışmaktadır [20].



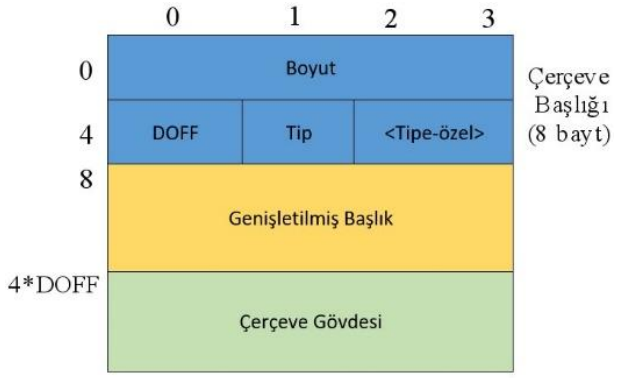
Şekil 2.13. AMQP Publish/Subscribe Mekanizması

AMQP iki farklı tip mesaj tanımlar: Bunlar yalın mesaj (bare message) ve açıklanmış mesajlardır (annotated message). Şekil 2.14'te bir AMQP mesajının genel formatı gösterilmektedir. Bu formattaki "başlık" alanı öncelik, mesajın yaşam süresi, ilk teslim alan, teslim sayısı gibi parametreleri içerir [20].



Şekil 2.14. AQMP Mesaj Formatı

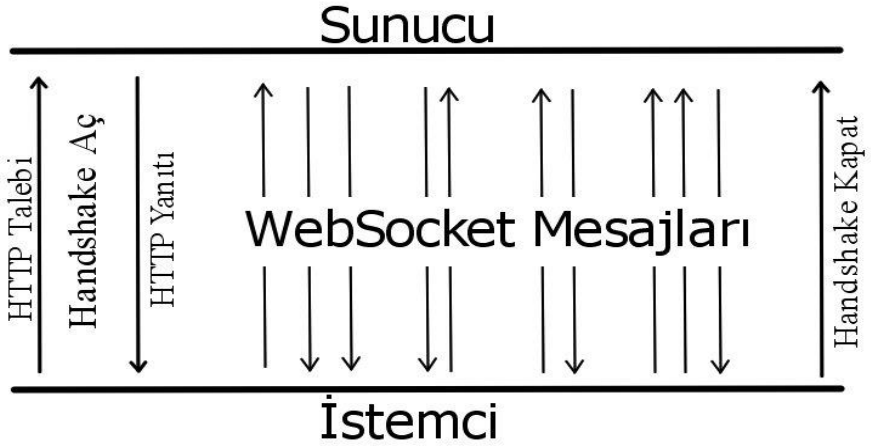
İletişim katmanında haberleşme çerçeve (frame) odaklıdır. Bir AMQP çerçevesinin yapısı Şekil 2.15.'te verilmiştir. İlk dört bayt çerçeve boyutunu gösterir. "DOFF" (Data Offset) çerçevede gövdenin konumunu belirtir. "Tip" alanı çerçevenin formatını ve amacını belirtir. Örneğin; 0x00 çerçevenin bir AMQP çerçevesi olduğunu ifade ederken 0x01 bir SASL çerçevesi olduğunu ifade eder.



Şekil 2.15. AMQP çerçeve formatı

2.6. WebSocket

WebSocket protokolü, sunucu ve istemciler arasında gerçek zamanlı ve çift yönlü bağlantı sağlamak için geliştirilmiştir. TCP üzerinden gerçekleşen bu bağlantıda taraflar birbirlerine aynı bağlantı üzerinden gerçek zamanlı veri aktarımı yapabilirler. WebSocket Protokolü, mevcut altyapılardan (proxy'ler, filtreleme, kimlik doğrulama) faydalanmak üzere HTTP'yi bir taşıma katmanı olarak kullanan mevcut çift yönlü iletişim teknolojilerinin yerini alacak şekilde tasarlanmıştır [21].WebSocket protokolü temel olarak web browser ve web sunucuları arasında çift yönlü haberleşmeye etkili ve standartlaşmış bir çözüm getirmek amacıyla geliştirilmiştir. Web browserlarda kullanılmak üzere tasarlanmıştır ancak IoT dahil farklı amaçlarla da kullanılmaktadır [5].



Şekil 2.16. WebSocket çalışma prensibi

2.6.1. WebSocket Mimarisi ve Çalışma Prensibi

WebSocket protokolünün çalışması temel olarak iki aşamadan oluşur. Bunlar mutabakat (handshake) ve veri transferi aşamalarıdır.

Bir WebSocket bağlantısı HTTP talepleri üzerinden başlatılarak handshake sağlanır. Daha sonra iletilen mesajlar TCP üzerinden iletilir [5]. Bu çalışma prensibi Şekil 2.16.'da görülmektedir.

İlk olarak istemci sunucuya http üzerinden bir bağlanma talebi iletir. Sunucu WebSocket'i destekliyorsa yine http üzerinden olumlu bir yanıt iletir ve handshake sağlanarak bağlantı sağlanmış olur. Bağlantı açıldıktan sonra istemci ve sunucu iletişimde WebSocket protokolünü kullanmaya başlar. Handshaka sonlandırmadıkça bu iki uç arasındaki bağlantı açık kalır. Bağlantı açık olduğu sürece her iki tarafta birbirlerine herhangi bir zamanda hatta aynı anda mesaj gönderebilirler. Bu aşamada mesajların talep veya yanıt paketleri üzerinden gönderilmesi gerekli değildir.

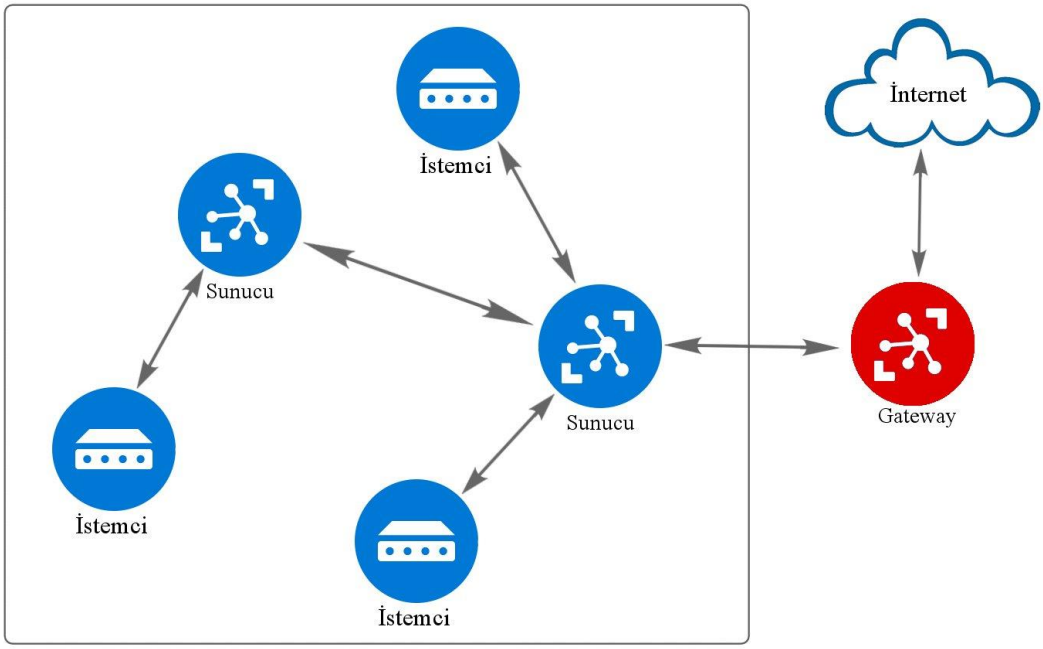
Mesajlar metin veya binary (ikili) veri içerebilirler. Mesaj başlıkları mümkün olan en küçük boyuta indirgenmiştir. Bu da bant genişliği tüketimini en aza indirmektedir. Mesajların birden fazla çerçeveye (frame) bölünerek sırayla iletilmesi mümkündür. Bu ise WebSocket protokolünün streaming olarak adlandırılan büyük boyutlu dosyaların iletilmesi işleminde kullanışlı bir protokol olmasını sağlamaktadır.

WebSocket bağlantıları HTTP'de olduğu gibi şifresiz veya TLS kullanılarak şifreli olarak kurulabilir. Varsayılan olarak HTTP ve HTTPS ile aynı portları (80 ve 443) kullandığından güvenlik duvarları ve proxy'ler tarafından tanınır. Bu sayede var olan HTTP ayarlarında herhangi bir değişiklik yapılmasına gerek olmaksızın çalışabilir.

2.7. XMPP

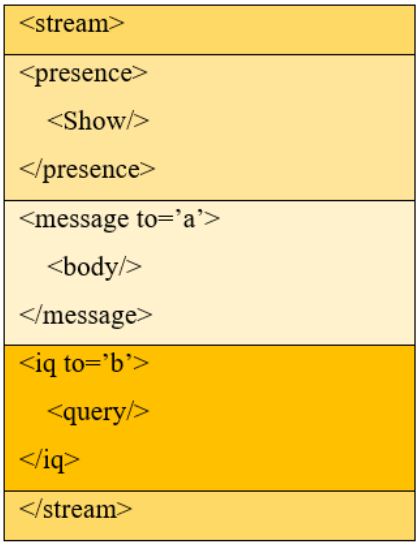
XMPP (Extensible Messaging and Presence Protocol), çoklu oturumlarda, sesli ve görüntülü aramalarda ve telekonferanslarda kullanılan bir IETF anlık mesajlaşma (instant messaging) standardıdır [20].

XMPP, Jabber açık kaynak topluluğu tarafından açık kaynaklı, güvenli, spamdan korunan ve bağımsız bir mesajlaşma protokolünü desteklemek için geliştirilmiştir. XMPP, kullandıkları işletim sistemi fark etmeksizin kullanıcıların internet üzerinden birbirleriyle anlık mesajlar göndererek haberleşmesine olanak sağlar. Bununla beraber anlık mesajlaşma uygulamalarının yetkilendirme, erişim kontrolü, mahremiyet önlemleri ve diğer protokollerle uyumluluk hedeflerine ulaşmalarına olanak sağlar. Şekil 2.17.'de gatewaylerin farklı mesajlaşma ağlarına köprü olabileceği genel bir XMPP haberleşmesi şematize edilmiştir [20].



Şekil 2.17. XMPP haberleşmesi

XMPP farklı özellikleriyle, IoT kapsamında değerlendirilen birçok anlık mesajlaşma uygulaması tarafından tercih edilmiştir. Bağımsız yapısıyla çeşitli internet tabanlı platformlarda çalışır. Güvenlidir ve çekirdek yapısının üzerine ek uygulamalar geliştirilmesine olanak sağlar. XMPP istemci ile sunucuyu bir XML stanza (dörtlük) stream'i (akışıyla) ile birbirine bağlar. XML stanza üç parçaya bölünmüş bir kod parçasını temsil eder. Bu parçalar; mesaj, varlık (presence) ve iq (info/query – bilgi/sorgu)'dur. Şekil 2.18.'deXMPP stanza yapısı görülmektedir [20].



Şekil 2.18. XMPP stanza yapısı

Mesaj stanzaları kaynak ve hedef adreslerini, tipleri ve veriyi almak için push metodu icra eden XMPP varlıklarının ID'lerini belirler. Mesaj stanzası konu (subject) ve gövde (body) alanlarını mesajın başlığı ve içeriğiyle doldurur. Varlık (presence) stanzası yetkili olan müşterilerin durum değişimlerini gösterir. *İq* stanzası göndericiler ve alıcıları eşleştirir [20].

Metinsel iletişimde XMPP, XML kullandığından ağa göreceli ek yük getirir. Bunun önüne geçmek için EXI gibi XML veri akışı sıkıştırıcıları geliştirilmiştir [20].

2.8. DDS

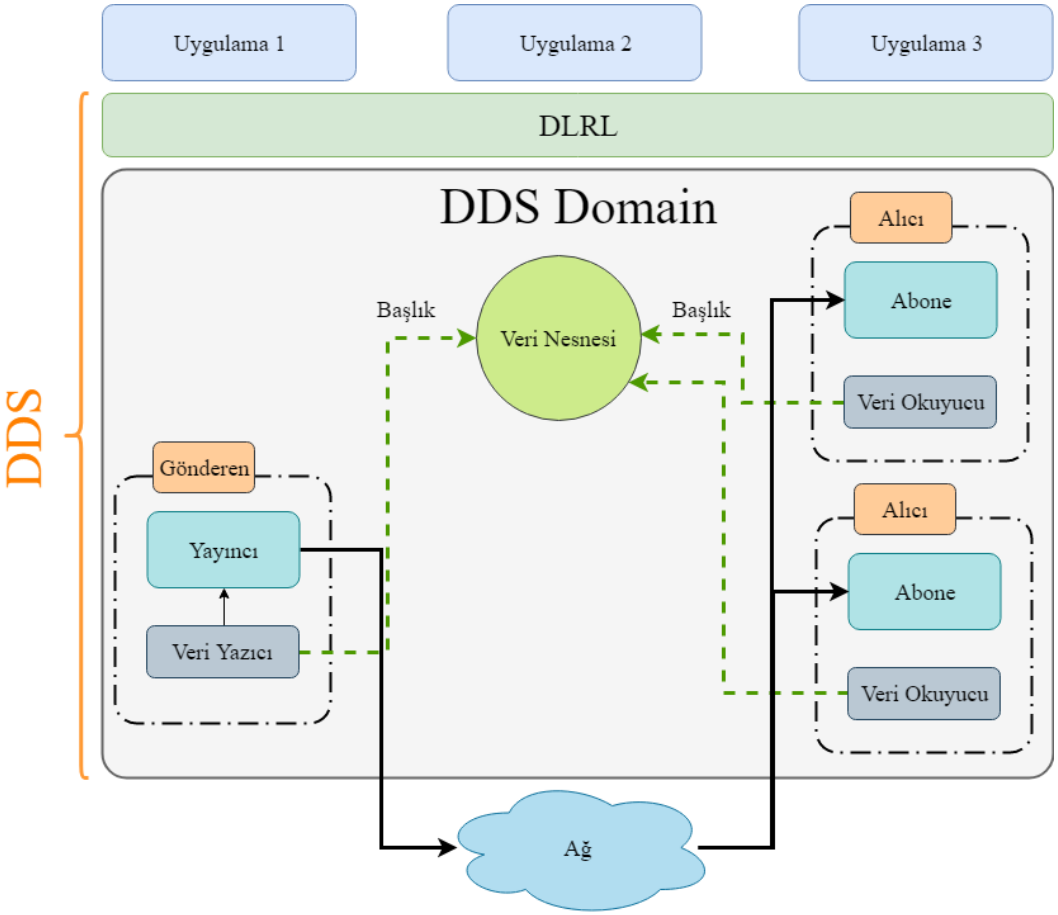
Data Distribution Service (DDS), Object Management Group (OMG) tarafından geliştirilen gerçek zamanlı makineden makineye (M2M) haberleşme için kullanılan publish/subscribe iletişim modelinde çalışan bir protokoldür [20][22].

MQTT ve AMQP gibi diğer publish/subscribe haberleşme protokollerinin aksine DDS broker'sız (aracısız) bir mimaride geliştirilmiştir. Bununla beraber uygulamalarında en iyi QoS yapısı ve yüksek kararlılığı sağlamak için multicasting (çoklu gönderme)

kullanır. DDS'in aracısız publish/subscribe mimarisi kısıtlı gerçek zamanlı M2M ve IoT haberleşmesiyle uyumludur. DDS güvenlik, önem sırası, öncelik, yaşam süresi, kararlılık vb. Kararlar için farklılık gösteren 23 adet QoS politikasını benimser.

DDS mimarisi Data Centric Publish-Subscribe (Veri Merkezli Publish/Subscribe - DCPS) ve Data-Local Reconstruction Layer (Yerel Veri Tekrar Yapılanma Katmanı - DLRL) olmak üzere iki adet katman tanımlar. DCPS bilgiyi abonelere dağıtma görevinin yerine getirir. DLRL ise DCPS fonksiyonlarına bir arayüz gibi çalışır ve tercihe bağlı bir kullanımı vardır. Yayımlanan verilerin dağınık nesneler arasında paylaşılmasını kolaylaştırır [22].

DCPS'deki veri akışında beş farklı varlıktan söz edilebilir. Bunlar: (1) Veriyi dağıtan yayıncı (Publisher), (2) verilen tipe göre veri içeriği ve değişiklerle ilgili yayıncıyla etkileşime geçmek için uygulama tarafından kullanılan veri yazmacı (DataWriter). DataWriter ve Publisher arasındaki ilişki, uygulamanın ilgili veriyi sağlanan içerikle yayınlayacağını gösterir; (3) yayınlanan veriyi alan abone, (4) alınan veriye erişim için aboneler tarafından kullanılan veri okuyucusu (DataReader), (5) bir veri tipi ve isimle tanımlanan ve DataWriter ile DataReader arasındaki ilişkiyi kuran başlık (Topic). Veri aktarımına, birbirine bağlı yayıncı ve abone uygulamalarından oluşan sanal bir ortamdan oluşan DDS alanı (domain) içerisinde izin verilir. Şekil 2.19.'da DDS'in kavramsal modeli verilmiş ve bu anlatılanlar örneklenmiştir.



Şekil 2.19. DDS'in kavramsal Modeli

2.9. SOAP

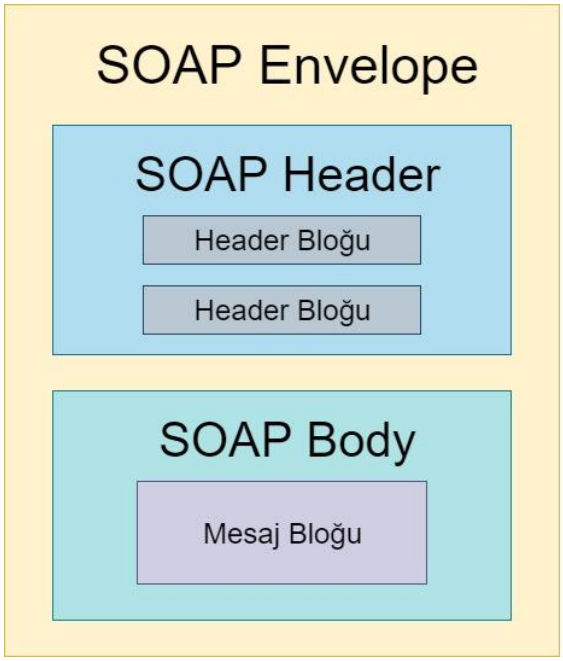
SOAP (Simple Object Access Protocol) küçük boyutlu (light-weight) ve genişletilebilir bir mesaj iletim protokolüdür [23]. SOAP, mesaj iletiminde Extensible Markup Language (XML) kullanımını gerekli kılar. XML birçok platform tarafından desteklenmesi sebebiyle SOAP protokolünün uygulama alanı da oldukça genişlik kazanmıştır. HTTP internet üzerinde en çok kullanılan transfer protokolüdür. SOAP temel bir transfer protokolünü zorunlu kılmaz. Ancak HTTP, SOAP için en sık kullanılan transfer protokolü haline gelmiştir. SOAP, HTTP üzerinden XML formatlı mesajlar gönderebildiğinden oldukça popüler hale gelmiştir.

2.9.1. SOAP Gönderimi

SOAP mesajları, verinin ASCII metin ifadesi olarak gönderilmesini gerektiren XML kullanılarak kodlanır. Verinin XML ile kodlanabilmesi içinde öncelikle metinsel bir ifade olması gerekir. Özellikle resim, video gibi medya verilerinin metinsel ifadeye dönüştürülmesi veri boyutunun büyük oranda artması anlamına gelir. Daha sonra metinsel veriye dönüştürülmüş olan bu mesaj açılış ve kapanış etiketleriyle sarmalanır. Bu işlemde bazen mesaj boyutunun iki katı veya daha fazlası artmasına sebep olur. XML formatına çevrilen mesaj ASCII formatına dönüştürüldükten sonra iletilir. Bu işlem ise özellikle akademik ve bilimsel ifadelerin veya ondalıklı sayıların gönderilmesinde mesaj boyutunun artmasına sebep olur.

2.9.2. SOAP Mesaj Yapısı

Bir SOAP mesajı bir XML dokümanı olarak tanımlanır ve zorunlu bir SOAP zarf (envelope), tercihe bağlı bir başlık (header) ve zorunlu bir gövde (body) elemanlarından oluşur. SOAP zarfı bu XML dokümanının en üstteki elemanıdır ve dokümanın bir SOAP mesajı olduğunu ifade eder. Header elemanı SOAP mesajlarına çeşitli özellikler eklemek için genel bir mekanizmadır. Bu özellikler yetkilendirme, iletim yönetimi, ödeme seçeneği vb. farklı eklentiler olabilir. Header elemanı, eğer kullanılacaksa SOAP Envelope elemanının ilk çocuk (child) elemanı olmalıdır. Alıcıya iletilmek istenen mesajın yer aldığı bölüm gövde elemanıdır. Aynı zamanda hata bildirimleri içinde kullanılır. Eğer başlık elemanı belirtilmişse gövde, başlıktan hemen sonra gelir. Aksi halde gövde elemanı zarf elemanının ilk çocuk elemanı olmalıdır [23]. Şekil 2.20. genel bir SOAP mesaj yapısını örneklemektedir.



Şekil 2.20. SOAP mesaj yapısı

Aşağıda Header kullanan bir SOAP mesajı örneklenmiştir.

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "Some-URI"

<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  <SOAP-ENV:Header>
     <t:Transaction
        xmlns:t="some-URI"
        SOAP-ENV:mustUnderstand="1">
           5
     </t:Transaction>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
     <m:GetLastTradePrice xmlns:m="Some-URI">
        <symbol>DEF</symbol>
     </m:GetLastTradePrice>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Burada ise Header kullanmayan bir SOAP mesajı örneklenmiştir.

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "Some-URI"

<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  <SOAP-ENV:Body>
     <m:GetLastTradePriceDetailed
      xmlns:m="Some-URI">
        <Symbol>DEF</Symbol>
        <Company>DEF Corp</Company>
        <Price>34.1</Price>
     </m:GetLastTradePriceDetailed>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

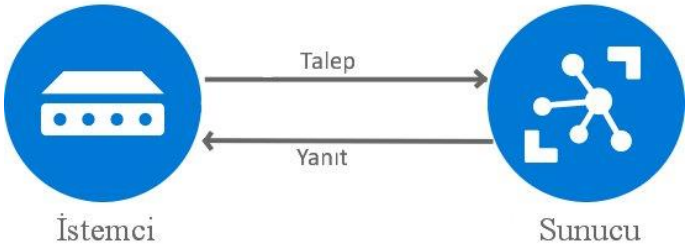
2.10. REST

REST (Representational State Transfer) mimarisi, HTTP protokolü üzerinden istemci-sunucu modelinde veri iletimini sağlayan web odaklı bir mimaridir. Basit yapıdadır; veri içeriği formatı, göstermesi istenen ek özellikler ve amaca göre şekillenebilmekte oldukça esnek yapıdadır. Bu mimariyi kullanan servisler RESTful servis olarak adlandırılır.

REST mimarisinde her bir bileşen birer kaynak (resource) olarak tanımlanır. Bu resource'lar birer metin dosyası, HTML sayfası, resim, video veya herhangi bir tipte veri olabilir. REST sunucusu her bir resource'a birer URI (Uniform Resource Identifier) atayarak erişime açar. İstemciler bunları, URI'ler ile oluşturulan URL'ler üzerinden erişerek yeniden formatlayabilir veya doğrudan kullanabilirler [24].

REST mimarisinde resource formatı hakkında herhangi bir kısıtlama söz konusu değildir. İletilecek veri uygulamaya göre çok farklı formatlarda iletilebilir. Bu formatlar web ortamında sıkça kullanılan yalın metin, JSON formatı, XML vb. olabilir.

REST mimarisinde mesajlar HTTP protokolü üzerinden talep-yanıt sisteminde iletilir. İstemci mesajı birer HTTP talebi olarak gönderir ve sunucuda bu mesaja bir HTTP yanıtı ile cevap verir. Bu model Şekil 2.21.'de örneklenmiştir. REST mimarisi HTTP taleplerinde Get, Post, Put ve Delete metotları kullanımını önerir. Bu metotlar uygulama kararlılığı ve güvenlik açısından önemli olabilmektedir.



Şekil 2.21. REST mimarisinde mesaj iletim modeli

REST mimarisi, veriye kolay ve anlamlı bir şekilde ulaşılmasını sağlaması, oldukça esnek yapısı ve küçük boyutlu çerçeve boyutu gibi özellikleriyle IoT'de sıkça kullanılan protokollerden biridir.

2.11. Protokol Karşılaştırmaları

Önceki bölümde IoT'de sıkça kullanılan birkaç haberleşme protokolünün genel tanımından, mimari yapılarından ve mesajları iletim modelleri ele alınmıştır. Bu bölümde ise adı geçen protokoller farklı özellikler üzerinden karşılaştırılmaktadır.

IoT'de uygulama geliştirenler kullandıkları cihazların sınırlılıkları, bant genişliği kısıtlamaları, enerji tüketimi vb. olumsuz birçok durumu göz önüne alırlar. Ele alınması gereken önemli bir değişken verinin iletilmesinde veya alınmasında kullanılacak haberleşme protokolünün belirlenmesidir. Günümüze kadar geliştirilen tüm haberleşme protokolleri farklı ihtiyaçların karşılanmasına dönük olarak ortaya çıkmış ve geliştirilmiştir. İhtiyaçların ve bu ihtiyaçları karşılayacak yöntemlerin farklı olmasıyla iletişim protokolleri de birbirlerinden az veya çok farklılaşmaktadırlar. Tablo 2.2.'de MQTT, MQTT-SN, CoAP, AMQP, WebSocket, XMPP, DDS, SOAP ve REST protokollerinin birbirine benzer ve farklı özellikleri özetlenmiştir.

Tablo 2.2. IoT protokolleri ve özellikleri

Özellikler	MQTT	MQTT-SN	CoAP	AMQP	WebSocket	XMPP	DDS	SOAP	REST
İletim Protokolü	TCP	TCP veya UDP	UDP	TCP	TCP	TCP	UDP veya TCP	TCP	TCP
Mimari	İstemci → Aracı → Abone	İstemci → Aracı	İstemci → Sunucu	İstemci → Aracı → Abone, İstemci → Sunucu	İstemci → Sunucu	İstemci → Sunucu → Abone	İstemci → Abone	İstemci → Sunucu	İstemci → Sunucu
İletim Modeli	Yayıncı → Abone	Yayıncı → Abone	İstek → Yanıt	İstemci → Aracı, Talep → Yanıt	İstek → Yanıt ve sonra Stream	Yayıncı →Abone	Yayıncı → Abone	İstek → Yanıt	İstek → Yanıt
Minimum Başlık Boyutu	2 bayt	2 veya 4 bayt	4 bayt	8 bayt	16 bayt	-	-	-	-
QoS – Servis Kaliteleri	QoS 0, QoS 1, QoS 2	QoS -1, QoS 0, QoS 1, QoS 2, QoS 3	Confirmable / Non-confirmable	Settle Format / Unsettle Format	TCP Standart	-	23 Farklı QoS Politikası	-	TCP Standart
Mesaj Boyutları	Azami 256 Mb	Azami 64 kilobayt	Genelde 1024 Kb veya daha az	Ön tanımsız	Ön tanımsız	Ön tanımsız	Ön tanımsız	Ön tanımsız	Büyük, Ön tanımsız
Güvenlik	TLS / SSL	TLS / SSL	DTLS, SSL	TLS / SSL, IPSec, SASL	TLS / SSL	TLS	SSL, DTLS	-	TLS / SSL
Varsayılan Port	1883, 8883 (TLS / SSL)	20000 (UDP), 1883 (TCP)	5683 (UDP) / 5684 (DLTS)	5671 (TLS / SSL), 5672	80/ 443 (TLS / SSL)	80/ 443 (TLS/SSL)	7400, 7410, 7411 54461, 59992, 59668	80/ 443 (TLS / SSL)	80/ 443 (TLS / SSL)

BÖLÜM 3. DENEYSEL ÇALIŞMA VE BAŞARIM DEĞERLENDİRMESİ

3.1. Deney Düzeneği

Bu bölümde, önceki bölümlerde ele alınan IoT protokollerinden MQTT, CoAP, WebSocket protokollerinin başarım değerlendirilmesi sunulmaktadır.

Gerçek dünya şartları deney düzeneğinin odak noktası olmaktadır ve bu durum önceki bölümlerde dile getirilmiştir. Bu amaçla yapılan deneylerde gerçek dünya şartlarındaki performans değerlendirmesinin ölçülmesi hedeflendi ve bunun için gerçek bir IoT kullanım amacına yakın bir deney ortamı hazırlanmaya çalışıldı.

IoT'de kullanılan sınırlı cihazların genel anlamda görevi sensör verisi veya başka bir cihazdan elde ederek ihtiyaca göre anlamlandırıp ağ üzerinden sunucuya (veya aracıya) iletmek veya sunucudan gelen komutları işleyip planlanan işlemi gerçekleştirmek olarak özetlenebilir. Bu işlem bir makineyi çalıştırmak, garajı kilitlemek, klimayı açmak, hastanın yatağını düzeltmek ve dahası birçok şey olabilir.

Deneylerde "yalnızca IoT haberleşme protokolünün performansını" ölçmek adına bu genellemede birkaç şeyden vazgeçilmek zorunda kalındı. Deneylerde herhangi bir sensör veya cihazdan gelen veri okumadı veya işlenmedi. Değişik amaçlara hizmet eden sensörlerden ve cihazlardan farklı markalarda ve standartlarda geliştirilmiş birçok farklı model vardır. Eğer bunlardan herhangi biri seçilerek deney yapılsaydı IoT protokolünün performansını ölçmede yapacağı etkinin genellenebilirliği olmayacaktı. Sonuçlara büyük bir etkisi olabilecek bu durumunun IoT protokolünün gerçek performansını ölçmemize engel olacağı düşünülmektedir. Aynı durum sunucudan gelen veriyi cihazlara aktarmada da geçerli olduğundan herhangi bir makine deneylere dahil edilmemiştir.

Deneylerde kullanılan sunucu kısmı için herhangi bir online IoT aracı, sunucu vb. özel firmalara ait yapılar kullanılmadı. Çeşitli özelliklerde, farklı konumlarda, farklı protokollere hizmet veren bu firmalar, yapılan deneylerde IoT protokolünün performansına doğrudan etki edecekti. Bu etki ise IoT protokolünün performansı üzerinde yapılacak yorumlarda genellenebilirlik açısında olumsuz etki yapacaktı. Bununla beraber Mosquitto, RabbitMQ gibi birçok farklı IoT protokolünü destekleyen, büyük boyutlu ve güçlü bilgisayar yazılımları yine belirtilen sebeplerden ötürü kullanılmadı. Bunun yerine normal bir dizüstü bilgisayar üzerinde sıfırdan bir yazılım geliştirilerek deneyler gerçekleştirildi.

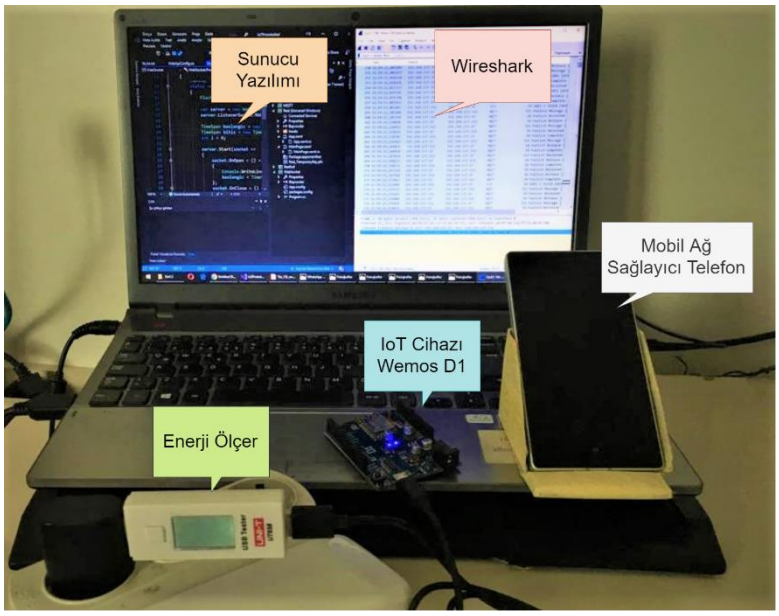
Tüm bunlara ek olarak çalıştırılması için online bir sunucu gerektiren protokoller deneylere dahil edilmedi. REST, SOAP, XMPP gibi HTTP protokolü üzerinde çalışan protokolleri çalıştırmak için Name Server (NS) sağlayan bir cihaz üzerinden internete açmak gerekmektedir. Bu ise ya Windows Server gibi sunucu işlevi gören bir işletim sistemine sahip olarak yapılabilir ya da deney yazılımını bir Hosting firması üzerinde çalıştırarak gerçekleştirilebilir. Bu iki seçenek hem maliyetli olması hem de oluşan paket trafiğinin izlenemediğinden veya deney yazılımının çalıştırılamadığından dolayı gerçekleştirilemedi. Ayrıca yerel olarak gerçekleştirilen deneylerin yanında harici bir kaynağa dayanan deneyler; yapılan karşılaştırmaların geçerliliğini olumsuz yönde etkileyecekti. Deneyler MQTT'de QoS 0, QoS 1 ve QoS 2 için, CoAP'ta Get, Post ve Put metotları için , WebSocket'te ise istemci-sunucu yönlü gerçekleştirildi ve grafiksel karşılaştırmalar yapıldı.

Sonuç olarak deneylerde kullanılan düzenek şu şekildedir: Sınırlı bir cihaz rastgele oluşan 4 basamaklı sayıyı çoklayarak (basitçe yan yana ekleyerek) oluşturulan 8, 16, 32, 64, 128, 256, 512 ve 1024 baytlık verileri her bir deney için 10 bin kere mobil 4G altyapısını kullanan kablosuz bir ağ üzerinden bir Laptop bilgisayara gönderir. Enerji ölçer harici bir cihaz, sınırlı cihazın harcadığı enerjiyi ölçer. Dizüstü bilgisayar üzerinden gelen ve giden paket trafiğini ise Wireshark programı kullanılarak izlenmektedir. Dizüstü bilgisayarda kullanılan yazılım ise her bir paketin geliş süresi arasında istenmeyen gecikmeleri ölçerek paketler arasındaki ortalama gecikme sürelerini hesaplar. Tablo 3.1.'de deney platformuna ait özellikler verilmiştir.

Tablo 3.1. Deney platformu özellikleri

METRİKLER	AÇIKLAMA
Mesaj Sayısı	10.000
Mesajlar arası bekleme zamanı	10 ms (ağa bağlanma hatası gibi istenmeyen hataların önüne geçmek için)
Mesaj Boyutu	8, 16 ,32, 64, 128, 256, 512, 1024 bayt
IoT cihazı	WeMOS D1
Sunucu	I7, 2.4 GHz, 8 GB RAM
Ağ Analiz Yazılımı	Wireshark
Enerji Ölçüm Cihazı	UNI-T UT658 USB

Şekil 3.1.'de oluşturulan deney düzeneği görülmektedir. Deney düzeneğinde IoT cihazı olarak, 80/160 MHz çalışma frekansı, 4 Mbayt bellek ve ESP8266EX WiFi modülüne sahip Wemos D1 cihazı kullanılmıştır. Sunucu olarak ise i7 işlemci, 2.4 GHz işlemci hızı, 8 GB belleğe sahip ve Windows 10 işletim sitemi kurulu bir dizüstü bilgisayar kullanılmıştır. Harcanan enerjiyi ölçmek için ise UNI-T UT658 USB marka, mAh formatında ölçüm yapan bir cihaz kullanılmıştır.



Şekil 3.1. Deney düzeneği

3.2. Başarım Değerlendirmeleri

3.2.1. Başarım Değerlendirmesi

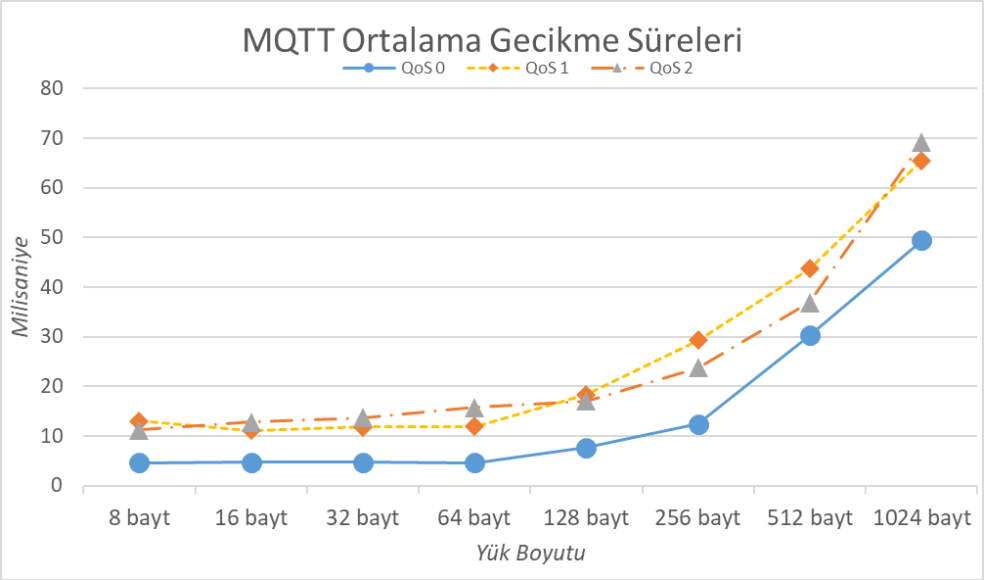
MQTT, CoAP ve WebSocket protokollerinde yapılan deneyler ile bu protokollerin iş çıkarım oranları (throughput), mesaj paketleri arasındaki ortalama gecikme süreleri ve enerji tüketimi değerleri elde edildi. Bu bölümde, önce her bir protokol için elde edilen sonuçlar ayrı ayrı incelenmektedir. Daha sonra bu sonuçlar grafik üzerinde karşılaştırarak performans analizi yapılmaktadır.

3.2.2. MQTT Performans Değerlendirmeleri

MQTT protokolünün başarım analizini yapmak için bu protokolün çalışma mimarisine uygun olarak birer istemci, aracı (broker) ve aboneden oluşan üç elemanlı bir deney düzeneği kuruldu. Bu düzenekte istemci tarafında rastgele oluşan 8, 16, 32, 64, 128, 256, 512 ve 1024 yük boyutuna (payload length) sahip mesajların her birinden 10 bin kere MQTT aracısına yayınlatıldı. Abone ise ilgili konuya (topic) abone olarak bu mesajları aldı. Bu işlemlerin sonunda MQTT protokolü için mesajların ortalama gecikme süresi, iş çıkarım oranı (throughput) ve enerji tüketimini değerleri elde edildi.

3.2.2.1. MQTT Protokolünün Mesaj Gecikme Değerleri

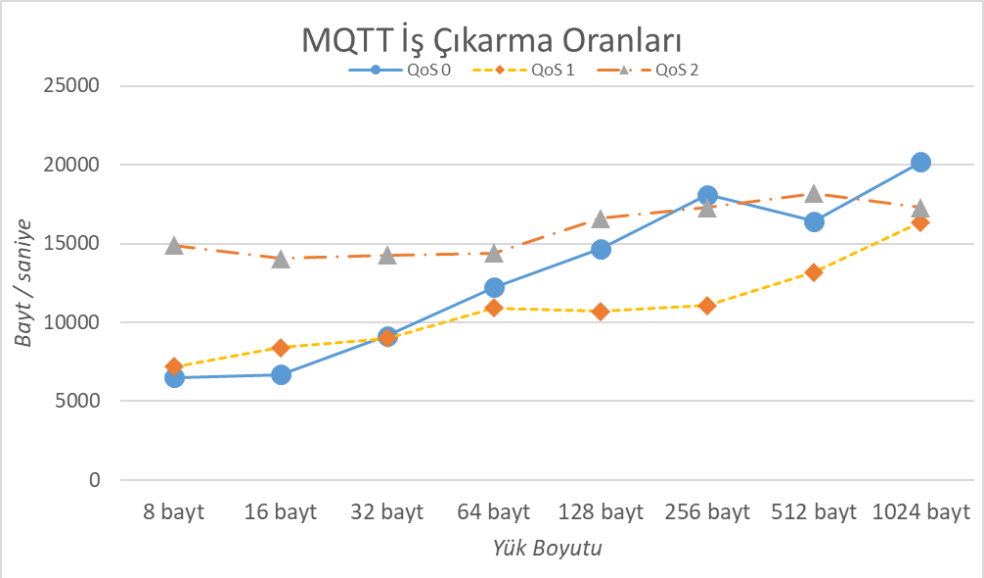
Ortaya çıkan Şekil 3.2.'deki grafiğe bakıldığında QoS seviyeleri ve yük boyutu artarken MQTT'nin mesaj gecikme sürelerinde doğrusal bir artış gözlenmektedir. Gecikme sürelerinin yüksek olması istenmeyen bir durumdur. QoS 0 modunda çalışırken herhangi bir geri bildirim paketi alınmadığından mesaj yükünün gecikme süresine etkisi en fazladır. QoS 1 ve QoS 2 modunda çalışırken yer yer farklılıklar olsa da yine aynı şekilde iki mod içinde doğrusal bir artış gözlenmekte. Aralardaki dalgalanmaların sebebi mobil bağlantıdan, cihazın o anki işlem kapasitesinden veya sunucunun yoğunluk durumundan kaynaklanmış olabilir.



Şekil 3.2. MQTT QoS seviyelerine göre ortalama mesaj gecikme süreleri.

3.2.2.2. MQTT Protokolünün İş Çıkarım Değerleri

İş çıkarım oranı (throughput) MQTT protokolünün saniyede iletebildiği veri boyutunu ifade eder (bayt/saniye). Yüksek boyutlu iş çıkarım oranı istenen bir durumdur ve protokolün performansını gösterir. Şekil 3.3.'de MQTT'nin QoS seviyelerine göre iş çıkarım performansları gösterilmektedir. Genel olarak yük boyutu arttıkça throughput değerinin arttığı gözlemlenebilmektedir. Ancak QoS seviyelerine göre farklı karakteristiklerde bir başarım söz konusudur. Örneğin; QoS 0 seviyesinde throughput değeri yük boyutuna göre keskin bir şekilde değişiklik gösterirken QoS 2 seviyesinde bu değişikliğin daha yumuşak olduğu görülmektedir. QoS 2 seviyesinde düşük yük boyutlarında bile yüksek throughput değerinin alınmasının sebebini mesaj iletiminde "kesinlikle bir kere" amacının sağlanması için oluşan bildirim paketi trafiğinin iletilen bayt boyutuna etkisi olarak yorumlanabilir.



Şekil 3.3. MQTT QoS seviyelerine göre iş çıkarma oranları.

3.2.2.3. MQTT Protokolünün Enerji Tüketimi Değerleri

Enerji ölçer cihaz, IoT cihazının çalıştırmak için gereken, üzerinden geçen elektrik akımını mili amper (mAh) cinsinden ölçmektedir. Bu akım değerini aşağıdaki formül ile Joule birimine dönüştürülür.

1. Akım (A) x Zaman (saniye) = Yük (C)
2. Yük (C) x Gerilim (V) = Enerji (J)

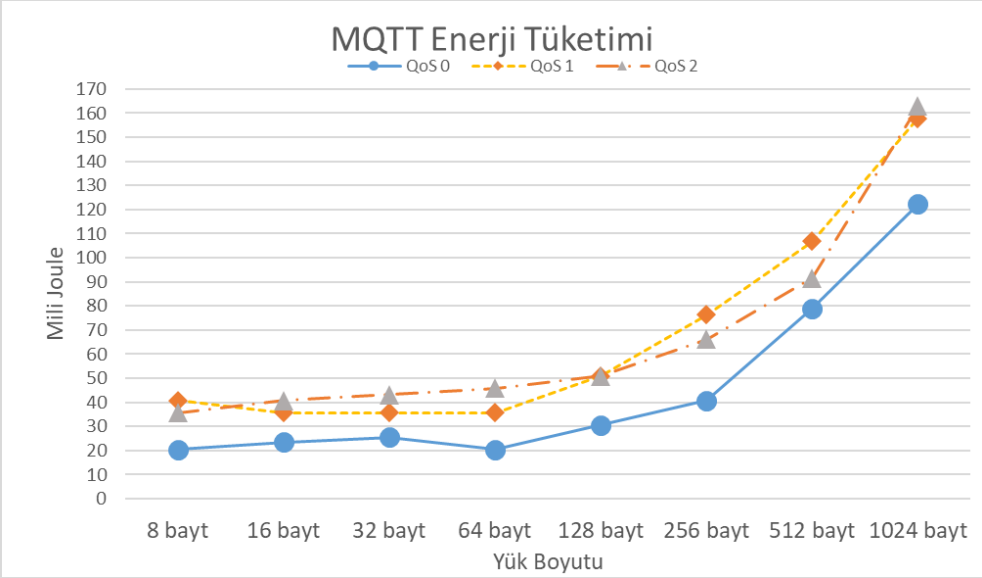
Birinci formülde (1) belirli bir zamanda harcanan Amper değerini bularak harcanan toplam elektriksel yükü bulunur. Akım (A) ifadesi zamana göre değişmeyen bir Amper değeri alır. Bu değeri zamana çarparak toplam elektriksel yük bulunmuş olur. Enerji ölçer cihaz çalıştığı süre boyunca değişen aralıklardaki tüm akım değerlerini saydığı için doğrudan ilk formülün sonucunu verir.

İkinci formülde (2) ise ilk formülde bulunan elektriksel yük ile voltaj değeri çarpılır. Bulunan sonuç Joule biriminden harcanan enerjiyi verir. Bu sonuç düşük seviyelerde olduğu için Joule biriminden MiliJoule birimine çevrim yapıldı.

Sonuç olarak aşağıdaki formül ile harcanan enerjiyi hesaplanabilir.

3. Enerji (J) = Enerji Ölçer Cihaz(mAh) X Voltage(V)

Şekil 3.4.'te elde edilen enerji tüketimleri MiliJoule biriminde izlenebilir. Harcanan enerjinin yük boyutuyla doğru orantılı olduğu gözlenmekte ve QoS seviyelerinde beklenen şekilde değişim gösterdiği görülebilmektedir.



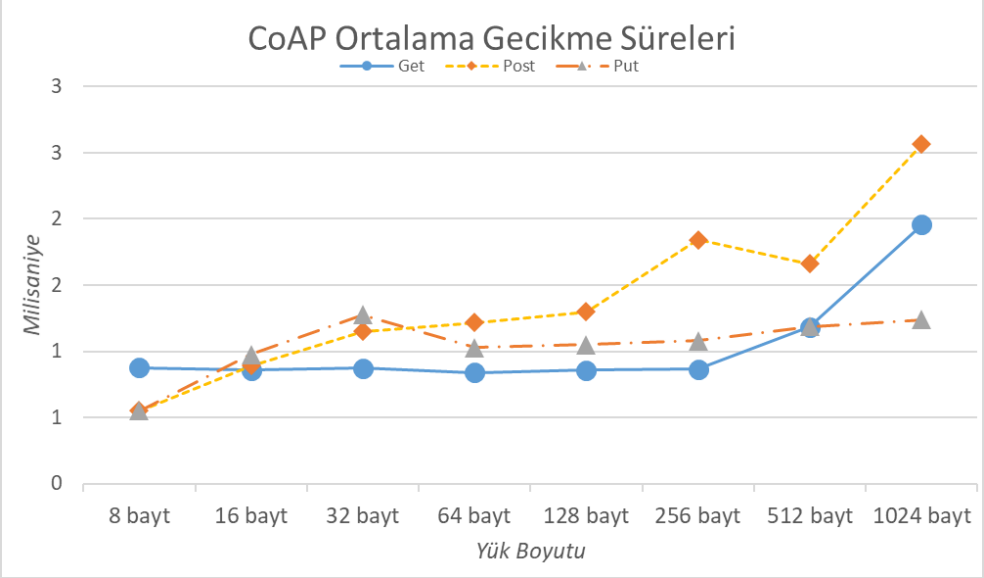
Şekil 3.4. MQTT QoS seviyelerine göre enerji tüketimleri.

3.2.3. CoAP Performans Değerlendirmeleri

CoAP protokolü için yapılan deneylerde istemci-sunucu arası haberleşme modelini sağlamak üzere bir adet istemci cihaz ve birde sunucudan oluşmak üzere iki elemanlı bir deney düzeneği kuruldu. Bu düzenekte istemci tarafında rastgele oluşan 8, 16, 32, 64, 128, 256, 512 ve 1024 bayt yük boyutlarına (payload length) sahip mesajların her birinden 10 bin kere POST ve PUT metotları üzerinden CoAP sunucusuna gönderim yapıldı. GET metodunun gereğini sağlamak için ise mesaj içeriklerini sunucunun oluşturması ve GET talebine yanıt olarak dönmesi sağlandı.

3.2.3.1. CoAP Protokolünün Mesaj Gecikme Değerlendirmesi

Şekil 3.5.'te CoAP protokolünün farklı metotlara göre farklı mesaj boyutlarındaki mesaj gecikme değerlendirme sonuçları görülmektedir. Mesaj gecikme değeri her bir başarılı mesaj iletiminde yaşanan gecikme süresini ifade etmektedir. CoAP protokolünde mesaj boyutu arttıkça genel olarak gecikme süresinin de arttığı görülmektedir. GET metodunun yüklü olmayan bir CON paketi göndermesine yanıt olarak sunucudan düzenli bir şekilde artan yüklü ACK paketi alması ve sınırlı istemci cihazın üzerine düşen iş yükünün az olması sebebiyle grafikte GET metodunda 512 bayt yük boyutuna kadar kararlı bir çizgi gözlenmektedir. 512 bayt yük boyutu ve sonrası için gecikme süreleri artmaktadır. POST ve PUT metotlarında ise istemci tarafındaki sınırlı cihaz üzerine iş yükü düştüğünden gecikme sürelerinde doğrusal olmayan farklılıklar oluşmuştur.

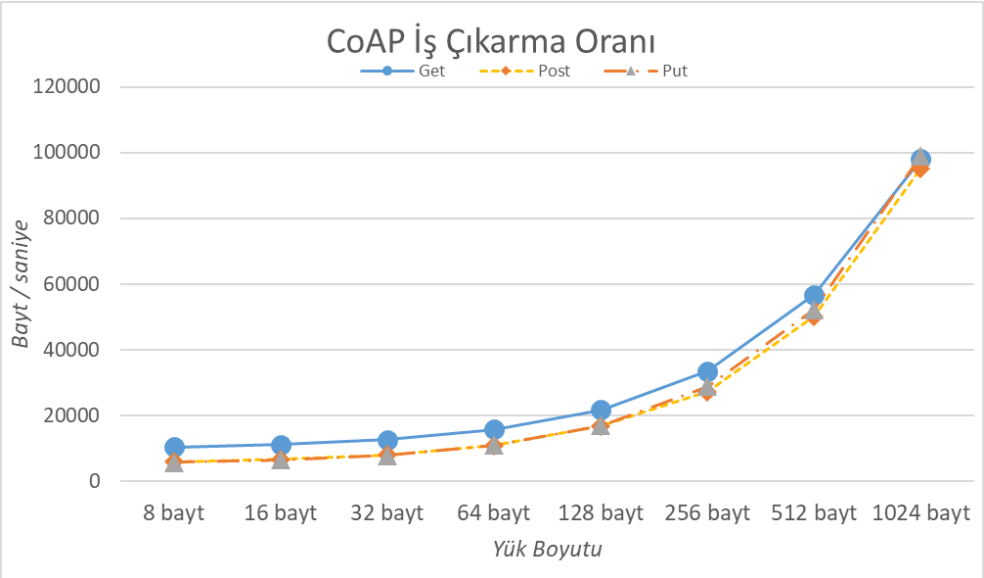


Şekil 3.5. CoAP protokolünün farklı metotlarda mesaj gecikme değerleri

3.2.3.2. CoAP Protokolünün İş Çıkarım Değerleri

Şekil 3.6.'da CoAP protokolünün farklı metotlara göre iş çıkarma oranları görülmektedir. Bu grafikte CoAP protokolü metotlarının yük boyutlarına göre yapılan testlerde saniyede iletilen ortalama bayt boyutları karşılaştırılmış ve yük boyutlarına göre doğrusal oranda artan sonuçlar elde edilmiştir. Bu sonuçlara en büyük etkinin yük

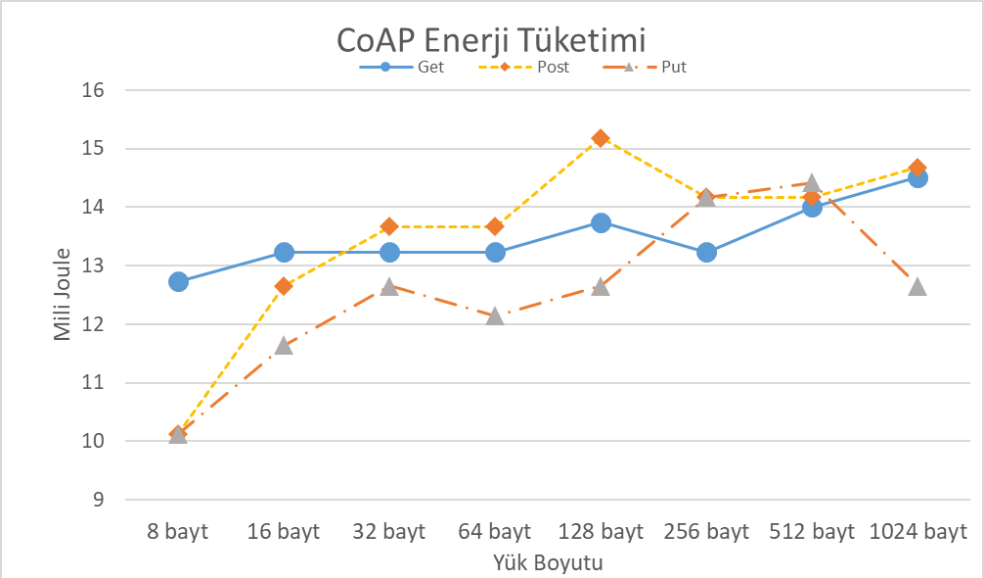
boyutu olduğu söylenebilir. Çünkü CoAP, yükünü çok düşük boyutlara sahip çerçeve boyutlarıyla sarmaladığından iş yapma oranı, yüküyle doğru orantılı olarak artmaktadır.



Şekil 3.6. CoAP protokolünün farklı metotlarda iş çıkarma oranları

3.2.3.3. CoAP Protokolünün Enerji Tüketimi Değerleri

Bölüm 3.2.2.3. 'de harcanan enerjiyi hesaplamak için kullanılan 3. formül CoAP protokolü içinde kullanıldı. Deneylerin sonunda elde edilen verilerden Şekil 3.7.'deki gibi bir grafik oluştu. Bu grafikte enerji tüketimlerinin genel olarak mesaj yüküne bağlı olarak arttığı söylenebilir. Enerji tüketiminde görünen bu farklar aslında oldukça düşüktür. Oluşan farkları istemciye düşen iş yükünden çok mobil bağlantı üzerinde oluşan anlık daralmalara ve mesaj kayıplarına dayandırabiliriz. Bunlarla beraber ortalama gecikme süreleri ile enerji tüketimi arasında özellikle GET metodu üzerinde görülebilen bir ilişki söz konusu. İlerleyen bölümlerde tüm protokollerin enerji tüketimi tek bir grafik üzerinde gösteridiğinden farkın ne kadar az olduğu daha iyi görülebilir.



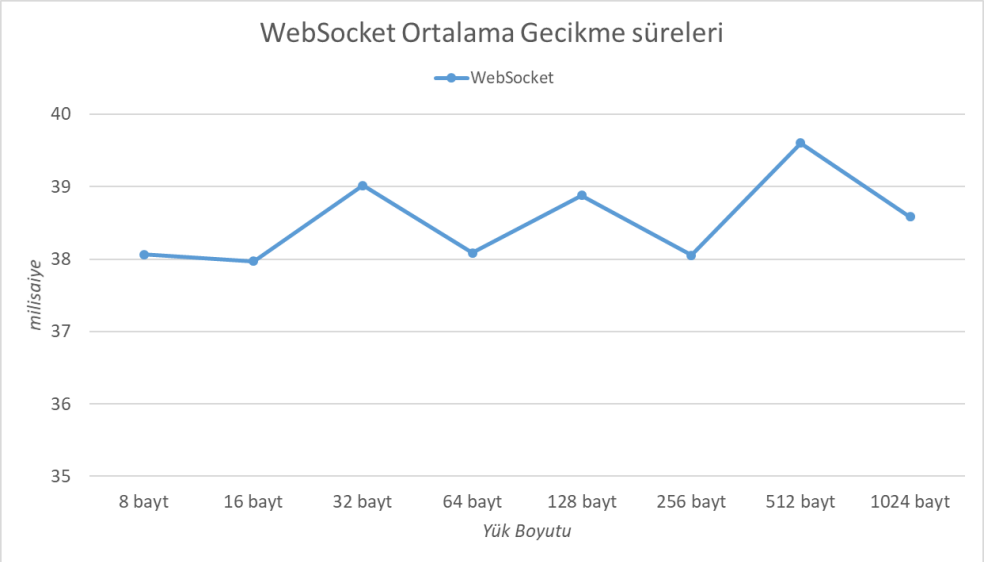
Şekil 3.7. CoAP protokolünün farklı metotlarda enerji tüketim değerleri

3.2.4. WebSocket Performans Değerlendirmeleri

WebSocket protokolü için yapılan deneylerde tıpkı CoAP'ta olduğu gibi istemci-sunucu arası haberleşme modelini sağlamak üzere bir adet istemci cihaz ve birde sunucudan oluşmak üzere iki elemanlı bir deney düzeneği kuruldu. WebSocket istemci-sunucu arasında sonlanana kadar sürekli açık bir bağlantı üzerinden gerçekleşir. Deneylerde istemci tarafından 8, 16, 32, 64, 128, 256, 512 ve 1024 yük boyutuna (payload length) sahip mesajların her birinden 10 bin kere, sonlanana kadar açık olan bu bağlantı üzerinden WebSocket sunucusuna veri iletimi yapıldı.

3.2.4.1. WebSocket Protokolünün Mesaj Gecikme Değerleri

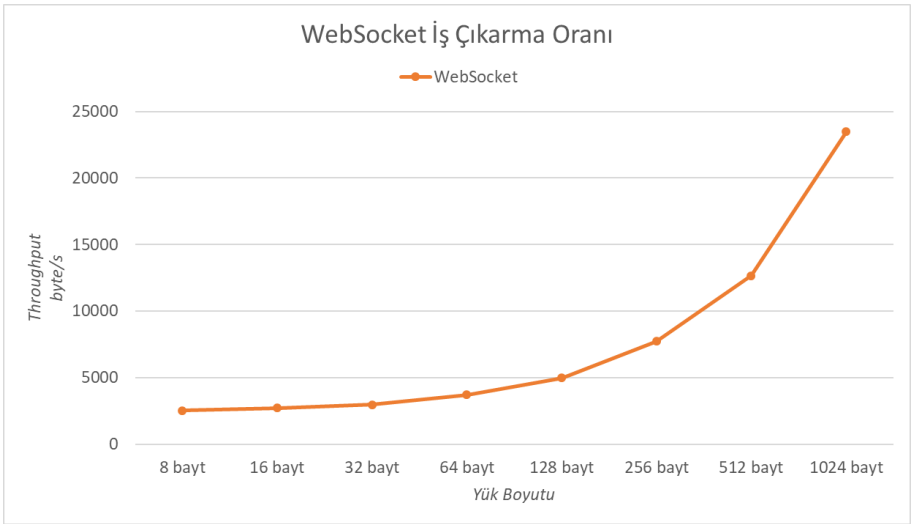
Yapılan deneylerde WebSocket protokolünde gönderilen verilerin gecikme sürelerinin yük boyutlarına göre ±2 milisaniye (ms) gibi çok küçük farklarla değiştiği gözlendi. Şekil 3.8.'de bu değerler görülebilir. WebSocket TCP'nin bütün imkanlarını kullanır ve bu sayede aynı anda çok büyük yüke sahip paketleri "veri akışı" olarak iletebilir. Bu nedenle yük boyutları arasında fark olsada ortalama mesaj gecikmeleri arasında çok büyük farklar oluşmamaktadır.



Şekil 3.8. WebSocket protokolü ortalama gecikme süreleri

3.2.4.2. WebScoket Protokolünün İş Çıkarım Değerleri

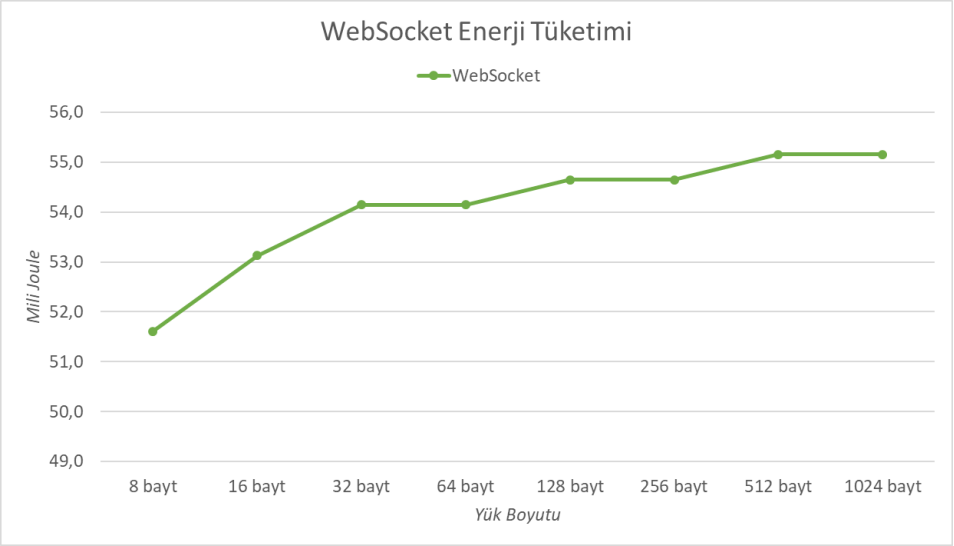
Bir önceki başlıkta WebSocket protokolünün TCP'nin tüm imkanlarını kullandığından bahsedilmişti. Şekil 3.9.'da bu durumu kanıtlar nitelikte olan sonuçlar görülmektedir. Grafik incelendiğinde WebSocket protokolünün yük boyutuna göre doğrusal artan iş çıkarma oranı sağladığı görülebilir. Bu ise sürekli açık olan bir bağlantı üzerinden TCP ile gönderilen paketlerin çerçeve boyutunun oldukça küçük olmasıyla açıklanabilir. Her paket için yeni bir bağlantı açılmasına ihtiyaç duyulmadığından iletilmesi istenen paket hedefine daha az ek çerçeve boyutlarıyla ulaşmış oluyor.



Şekil 3.9. WebSocket protokolünün iş çıkarma oranları

3.2.4.3. WebSocket Protokolünün Enerji Tüketimi Değerleri

WebSocket protokolünde mesaj iletiminin sonlandırılana kadar sürekli açık olan bir bağlantı üzerinden gerçekleştiğinden daha önce bahsedilmişti. Bu özellik yüksek iş çıkarma oranı avantajı sağlar ancak enerji tüketiminin artması gibi önemli bir bedelle elde edilebilir. Şekil 3.10.'da görüldüğü üzere WebSocket'in enerji tüketim değerleri önceki protokollerden oldukça fazla ve yük boyutu arttıkça bu tüketim değerleri daha da artıyor.



Şekil 3.10. WebSocket protokolünün enerji tüketim değerleri

3.3. Başarımların Karşılaştırılması

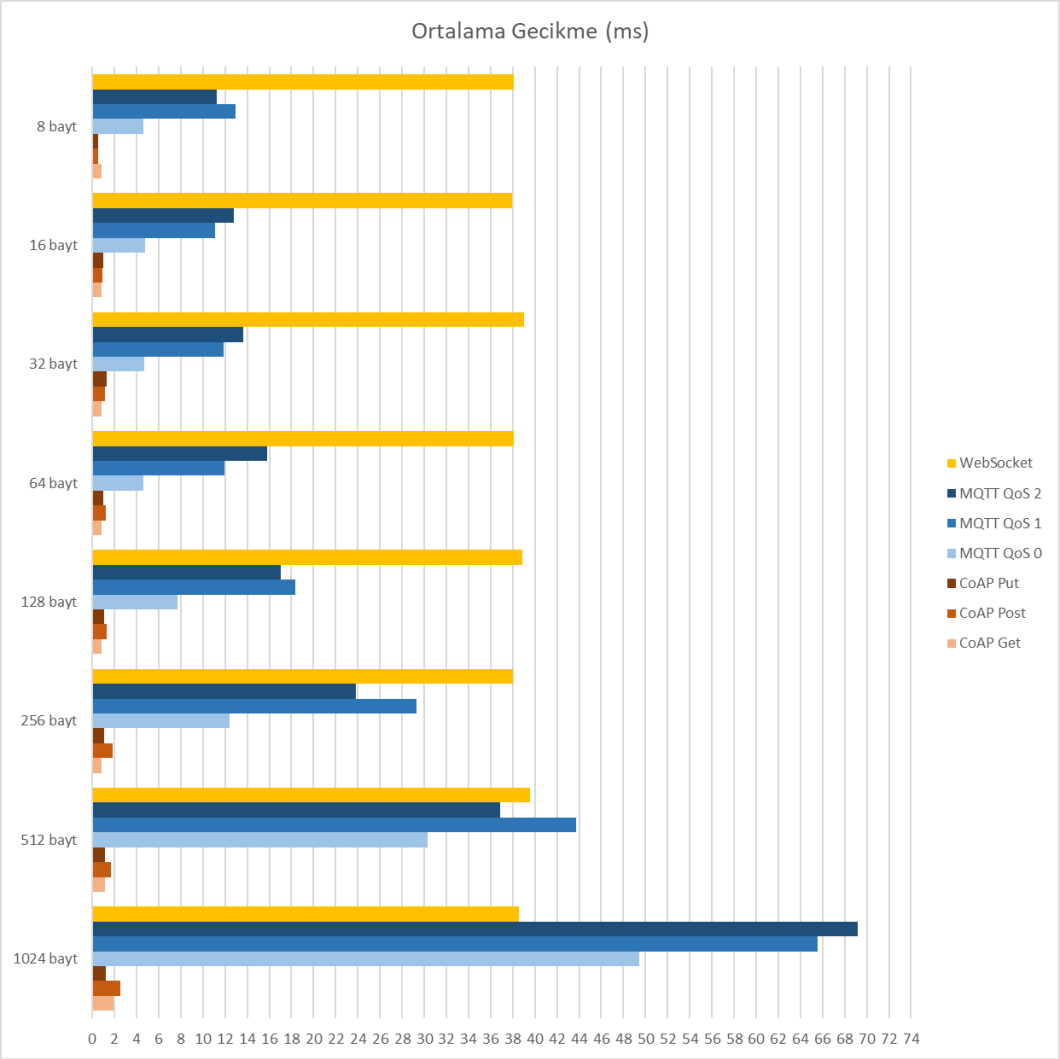
Önceki bölümde MQTT, CoAP ve WebSocket protokolleriyle gerçekleştirilen deneylerde elde edilen ortalama mesaj gecikmeleri, iş çıkarma oranları ve enerji tüketimleri değerleri incelendi ve bu değerler üzerinde analizler yapıldı. Bu bölümde ise bu değerler kümülatif bir şekilde grafik üzerinde karşılaştırılacak ve genel anlamda bu protokollerin performansları hakkında değerlendirmeler yapılacaktır.

Burada MQTT protokolü hakkında bir noktadan bahsedilmesi gerekmektedir. Deneylerde MQTT'nin yayıncı/abone mimarisi kullanıldı. Bu mimaride iletilecek her bir mesaj önce aracıya gönderilir. Aboneler de bu paketi aracıdan çekerler. CoAP ve

WebSocket varsayılan istemci-sunucu mimarisinde mesajlar istemciden sunucuya tek seferde gönderilirken MQTT'de iki defa işlem yapılır. Bu yüzden MQTT'yi diğer protokoller ile kıyaslarken bu durumu gözönüne almak gerekir. MQTT'yi istemci/sunucu mimarisine uyarlayacağımız bir yapı kurarak deneyler gerçekleştirilseydi elde edilen sonuçlarda kabaca ortalama paket gecikme sürelerini ve enerji tüketimlerini iki kat daha az, iş çıkarım oranlarının ise iki kat daha fazla olacağını düşünmek yanlış olmazdı.

3.3.1. Gecikme Süreleri Analizi

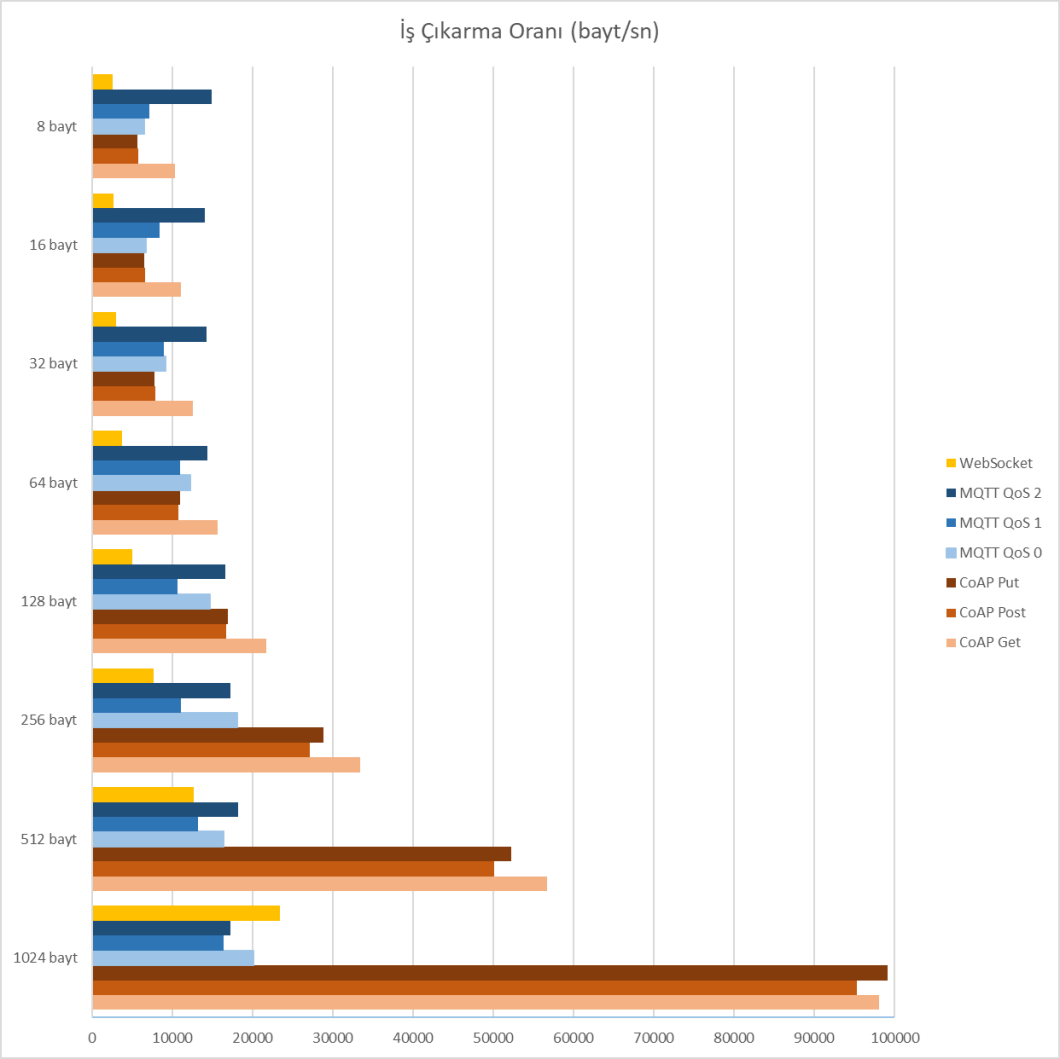
Şekil 3.11.'de üzerinde deneyler yapılan protokollerin ortalama gecikme süreleri değerleri ve bu değerlerin birbirinden farkları görülebilir. UDP üzerinden iletişim sağlaması ve performans odaklı yapısıyla paket başına ortalama en az gecikme süresini sağlayan protokolün CoAP olduğu görülmektedir. En çok gecikme süresi ise WebSocket protokolü ile yapılan deneylerde gözlemlendi. MQTT ise WebSocket'e göre daha performanslı olsada TCP üzerinden çalışması ve paketleri yayıncı/abone mimarisinde iletmesi sebebiyle CoAP'ın oldukça gerisindedir.



Şekil 3.11. Protokollerin ortalama gecikme sürelerinin karşılaştırılması

3.3.2. İş Çıkarma Oranı Analizi

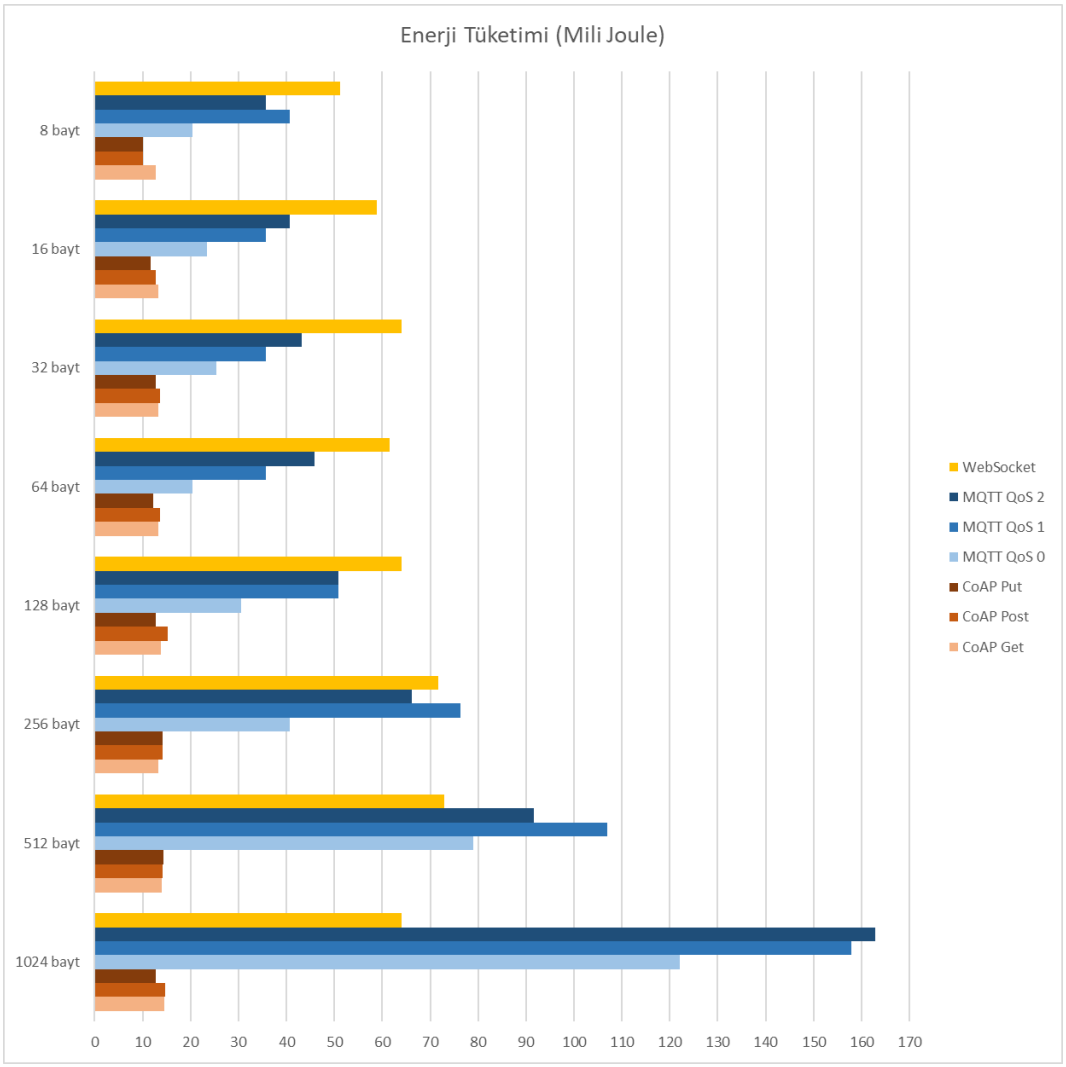
Şekil 3.12.'de deneylerden elde edilen, protokollerin iş çıkarma değerleri görülmektedir. İş çıkarma değeri saniyede iletilen bayt miktarını ifade eder ve mümkün olduğunca yüksek olması istenir. Grafik incelendiğinde MQTT'nin iş çıkarma oranı yük boyutlarına göre çok fazla değişmezken CoAP yük boyutlarına göre giderek artan bir ivme göstermektedir. Bununla beraber küçük yük boyutlu denemelerde CoAP ile MQTT başa baş bir performans gösterirken yük boyutu arttıkça CoAP'ın iş çıkarma performansında MQTT'nin önüne geçtiği söylenebilir. WebSocket ise yük boyutu arttıkça gösterdiği performans artsada MQTT ve CoAP'ın oldukça gerisindedir.



Şekil 3.12. Protokollerin iş çıkarma oranlarının karşılaştırılması

3.3.3. Protokollerin Enerji Tüketim Analizi

Şekil 3.13.'te deneylerden elde edilen, protokollerin yük boyutlarına göre enerji tüketimi değerleri görülebilir. Enerji tüketiminin mümkün olduğunca az olması istenen bir davranıştır. CoAP gösterdiği yüksek iş çıkarma oranına kıyasla en az enerji tüketen protokol olmuştur. MQTT protokolünün yüksek yük boyutlarında WebSocket'ten daha fazla enerji tükettiği grafik incelendiğinde görülebilir. Bunun sebebi ise yine ise yayıncı/abone mimarisinde veri iletilmesine bağlanabilir. Eğer istemci-sunucu mimarisinde yapılacak olsaydı kabaca yarı yarıya daha az enerji tüketimi gözlemlenebilirdi.



Şekil 3.13. Protokollerin enerji tüketimlerinin karşılaştırılması

BÖLÜM 4. SONUÇ VE ÖNERİLER

Bu tez çalışmasında IoT'de sıklıkla kullanılan protokoller, çalışma mantığı ve mimari yapıları gözönünde bulundurularak ele alınmıştır. Bu protokollerden MQTT, CoAP ve WebSocket protokolleri üzerinde oluşturulan düzenekle üzerlerinde başarım karşılaştırmaları yapıldı. Bu deneylerin sonucunda MQTT protokolünün QoS 0, QoS 1 ve QoS 2 seviyelerinde, CoAP protokolünün Get, Post ve Put metotlarında ve WebSocket protokolünde istemci sunucu arasında her bir 8, 16, 32, 64, 128, 256, 512, 1024 baytlık yük boyutu aralığında gösterdikleri ortalama paket gecikme süreleri, iş çıkarma oranları ve enerji tüketimi değerleri karşılaştırıldı.

Deney sonuçları incelediğimizde hemen hemen tüm durumlarda CoAP'ın en iyi performansı sergilediğini görülmüştür. CoAP, UDP üzerinde çalışır ve UDP üzerinden yapılan haberleşmelerde handshake (üçlü el sıkışma) aşamaları desteklemez. Bu durum CoAP'ı performans olarak diğer protokollerden üstün kılmaktadır. Bu açıdan bakıldığında CoAP'ın az enerji harcanarak çok haberleşme gerektiren –sensörlerden okunan verilerin milisaniyeler bazında istemciden sunulara iletilmesi gibi– ve handshake gerektirmeyen IoT uygulamalarında kullanımının uygun olduğu söylenebilir.

MQTT protokolü ise TCP üzerinde çalışır. TCP üzerinde çalışan uygulamalar handshake prosedürlerini takip eder ve aralarında bir bağlantı oluştuktan sonra bu bağlantı üzerinden haberleşirler. UDP'ye göre daha güvenli bir haberleşme sunsada performans olarak geri kalmaktadır. Yayıncı/abone modelinde çalışan MQTT'nin güçlü yanı; aynı mesajın birden fazla aboneye maksimum performansla iletilmesidir. Yayıncı modunda çalışan sınırlı cihaz sadece bir kere aracı cihaza mesaj gönderir ve aboneler bu mesajı bu aracı cihazdan çekerler. Bu metotla sınırlı cihaz üzerine düşen

iş yükü minimize edilmiş olur. Bu nedenlerle ve deney sonuçlarına göre TCP üzerinden handshake sağlanarak güvenli haberleşme ortamında çalışması istenen ve bir mesajın birden fazla alıcıya aynı anda gönderilmesini gerektiren IoT uygulamalarında MQTT protokolü tercih edilebilir.

WebSocket protokolü MQTT gibi TCP üzerinde çalışır ve handshake prosedürlerini gerçekleştirir. İlk bağlanma mesajları HTPP üzerinden iletilir ve bağlantı sağlandığında bu bağlantı sonlanana kadar mesajlar TCP protokolü üzerinden çift yönlü iletilir. Bu bağlantının haberleşme boyunca açık olması deney sonuçlarında görüldüğü gibi WebSocket'in performansına olumsuz etki yapar. Ancak WebSocket'in veri akışlarıyla gerçekleştirdiği haberleşme modelinde çok büyük verilerin hemen hemen aynı işlem yüküyle iletilmesi bu alanda kullanılmasına olanak sağlar. Deney sonuçları incelendiğinde mesaj yükü arttığında CoAP ve özellikle MQTT'nin enerji tüketim değeri ve ortalama mesaj gecikme süreleri artarken WebSocket protokolünde bu değerler hemen hemen aynı kalmıştır. Buradan yola çıkarak büyük boyutlu verilerin kısa sürede teslimini gerektiren IoT uygulamalarında WebSocket'in iyi bir tercih olacağını söyleyebiliriz.

Bu çalışma alanında, gelecekteki çalışmalarda bu protokollere ek olarak HTTP üzerinde çalışan (REST, SOAP, XMPP gibi) başka protokollerin performans analizleri yapılabilir. Ölçtüğümüz parametrelerin yanında CPU kullanımı gibi başka verilerde analiz edilebilir. Deney düzeneği olarak sensörlerden veri okuyan bir yapı kullanılabilir. Bunlara ek olarak bu protokollerin performansları üçüncü parti yazılımları veya online araçlar kullanarak test edilerek bu ortamlardaki performans analizleri de çıkarılabilir.

KAYNAKLAR

[1] Hung, M. (2017). Leading the iot, gartner insights on how to lead in a connected world. Gartner Research, 1-29. https://www.gartner.com/imagesrv/books/iot/iotEbook_digital.pdf, Erişim Tarihi: 18.04.2019.

[2] Vermesan, O., & Friess, P. (Eds.). (2013). Internet of things: converging technologies for smart environments and integrated ecosystems. River Publishers.

[3] Mazhelis, O., Warma, H., Leminen, S., Ahokangas, P., Pussinen, P., Rajahonka, M., & Myllykoski, J. (2013). Internet-of-things market, value networks, and business models: State of the art report. University of Jyvaskyla, Department of Computer Science and Information systems, Technical Reports TR-39.

[4] Chaudhary, A., Peddoju, S. K., & Kadarla, K. (2017, October). Study of internet-of-things messaging protocols used for exchanging data with external sources. In 2017 IEEE 14th International Conference on Mobile Ad Hoc and Sensor Systems (MASS) (pp. 666-671). IEEE.

[5] Srinivasan, L., Scharnagl, J., & Schilling, K. (2013). Analysis of websockets as the new age protocol for remote robot tele-operation. IFAC Proceedings Volumes, 46(29), 83-88.

[6] Amaran, M. H., Noh, N. A. M., Rohmad, M. S., & Hashim, H. (2015). A comparison of lightweight communication protocols in robotic applications. Procedia Computer Science, 76, 400-405.3.

[7] Sarafov, V. (2017). Comparison of IoT Data Protocol Overhead.

[8] Naik, N. (2017, October). Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP. In 2017 IEEE international systems engineering symposium (ISSE) (pp. 1-7). IEEE.

[9] Gültunca, C. (2018). Nesnelerin internetinde uygulama katmanı üzerindeki haberleşme protokollerinin incelenmesi ve deneysel karşılaştırılması (Master's thesis, İstanbul Ticaret Üniversitesi Fen Bilimleri Enstitüsü).

[10] Foster, A. (2015). Messaging technologies for the industrial internet and the internet of things. PrismTech Whitepaper.

[11] Ebleme, M. A., Bayilmis, C., & Cavusoglu, U. Examination and Performance Evaluation of MQTT, 3. Uluslararası Bilgisayar Bilimleri ve Mühendisliği Konferansı (UBMK2018), 246-250, Saraybosna, Bosna Hersek, 20-23 Eylül 2018.

[12] Govindan, K., & Azad, A. P. (2015, January). End-to-end service assurance in IoT MQTT-SN. In 2015 12th Annual IEEE Consumer Communications and Networking Conference (CCNC) (pp. 290-296). IEEE.

[13] Hunkeler, U., Truong, H. L., & Stanford-Clark, A. (2008, January). MQTT-S—A publish/subscribe protocol for Wireless Sensor Networks. In 2008 3rd International Conference on Communication Systems Software and Middleware and Workshops (COMSWARE'08) (pp. 791-798). IEEE.

[14] Liri, E., Singh, P. K., Rabiah, A. B., Kar, K., Makhijani, K., & Ramakrishnan, K. K. (2018, June). Robustness of IoT Application Protocols to Network Impairments. In 2018 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN) (pp. 97-103). IEEE.

[15] Stanford-Clark, A., & Truong, H. L. (2013). Mqtt for sensor networks (mqtt-sn) protocol specification. International business machines (IBM) Corporation version, 1.

[16] http://coap.technology, Erişim tarihi: 11.03.2019.

[17] Herrero, R. (2018). Analytical model of IoT CoAP traffic. Digital Communications and Networks.

[18] Fernandes, J. L., Lopes, I. C., Rodrigues, J. J., & Ullah, S. (2013, July). Performance evaluation of RESTful web services and AMQP protocol. In 2013 Fifth International Conference on Ubiquitous and Future Networks (ICUFN) (pp. 810-815). IEEE.

[19] Pohl, M., Kubela, J., Bosse, S., & Turowski, K. (2018, October). Performance Evaluation of Application Layer Protocols for the Internet-of-Things. In 2018 Sixth International Conference on Enterprise Systems (ES) (pp. 180-187). IEEE.

[20] Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., & Ayyash, M. (2015). Internet of things: A survey on enabling technologies, protocols, and applications. IEEE communications surveys & tutorials, 17(4), 2347-2376.

[21] Fette, I., & Melnikov, A. (2011). The websocket protocol (No. RFC 6455).

[22] Data distribution services specification, V1.4, Object Manage. Group (OMG), Needham, MA, USA, Apr. 20, 2019. https://www.omg.org/spec/DDS/1.4/, Erişim Tarihi: 19.04.2019.

[23] Simple Object Access Protocol (SOAP) 1.1, W3C Note 08 May 2000, https://www.w3.org/TR/soap11/, Erişim Tarihi: 20.04.2019.

[24] Garg, S., & Ansari, M. S. (2017, August). Implementation of REST Architecture in ARDUINO Based Home Automation System. In 2017 International Conference on Innovations in Control, Communication and Information Systems (ICICCI) (pp. 1-3). IEEE.

ÖZGEÇMİŞ

Mehmet Ali Ebleme 08.06.1991'de Karabük'te doğdu. İlk, orta ve lise eğitimini İstanbul'da tamamladı. 2009 yılında Avcılar Teknik Lisesi'nden mezun oldu. 2009 yılında başladığı Kocaeli Üniversitesi Bilgisayar Öğretmenliği Bölümü'nü 2014 yılında bitirdi. 2016 yılında Sakarya Üniversitesi Bilgisayar Mühendisliği Bölümünde yüksek lisans eğitimine başladı. Halen IoT üzerine çalışmalar yürütüyor.