T.C.

İSTANBUL OKAN ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

BİLGİSAYAR MÜHENDİSLİĞİ YÜKSEK LİSANS PROGRAMI

JSON SÖZDİZİMLİ GERÇEK ZAMANLI İLETİŞİM PROTOKOLÜ

YÜKSEK LİSANS TEZİ

SERKAN AYAZ

tarafından

YÜKSEK LİSANS

derecesi şartını sağlamak için hazırlanmıştır.

Haziran 2019

Program: Bilgisayar Mühendisliği

JSON SÖZDİZİMLİ GERÇEK ZAMANLI İLETİŞİM PROTOKOLÜ

YÜKSEK LİSANS TEZİ

SERKAN AYAZ

tarafından

İSTANBUL OKAN ÜNİVERSİTESİ

Bilgisayar Mühendisliği Programı

Yüksek Lisans

derecesi şartını sağlamak için sunulmuştur.

Onaylayan:

Prof. Dr. Semih Bilgen
Danışman

Doç. Dr. Sanem Sarıel Uzer
(İstanbul Teknik Üniversitesi)
Üye

Prof. Dr. B. Tevfik Akgün
Üye

Haziran 2019

Program: Bilgisayar Mühendisliği

ÖZET

Nesnelerin İnterneti (IoT), kritik görev yazılımları, kurumsal uygulamalar, mikroservisler vb. alanlarda gerçek zamanlı ve platformdan bağımsız iletişim altyapılarına ihtiyaç duyulmaktadır. Bu konuda TCP protokolünün Unicast, Multicast ve Broadcast tipi iletişim tipleri yetersiz kalmaktadır. Aracı sunucu üzerinden yapılan veri aktarımlarında ağ trafiği azaldığından veri iletim performansı artmaktadır ve aktarılan veriler daha optimum seviyede kullanılmaktadır. Bu çalışmada platformdan bağımsız ağ istemcileri arasında gerçek zamanlı iletişim kurmak için aracı sunucu üzerinden verilerin aktarıldığı metin tabanlı ve JSON sözdizimli JTP (*JSON Transmission Protocol*) adında yeni bir protokol önerilmektedir. Yapılan çalışmada STOMP, XMPP, RESP ve NATS protokolleri incelenerek protokol boyutları ve algoritma performansı, çerçeve boyutları ve algoritma performansı ve insanlar tarafından okunup kodlanabilme seviyeleri değerlendirilmiştir. İncelenen protokoller ile JTP protokolü, karşılaştırma kriterleri ve işlevsel özellikler bakımından testleri yapılarak karşılaştırılmıştır. Yapılan değerlendirme sonucunda JTP'nin diğer alternatiflerine göre daha fazla işlevsel özelliğe sahip olduğu gösterilmiştir.

Anahtar Kelimeler: Protokol, JSON, Aracı Sunucu, TCP, STOMP, XMPP, RESP, NATS, Gerçek Zamanlı İletişim

ABSTRACT

Real-time and platform-independent communication infrastructures are needed in the Internet of Things (IoT), critical task software, enterprise applications, microservices etc areas. In this regard, Unicast, Multicast and Broadcast communication types of the TCP protocol are insufficient. Data transmission performance increases through data transfer from the broker server, as network traffic is reduced and the transferred data are used at the optimum level. In this study, we propose a new protocol based on text-based and JSON syntax JTP (*JSON Transmission Protocol*) in which data is transmitted via the broker server to communicate in real time between the platform-independent network clients. In this study, STOMP, XMPP, RESP and NATS protocols were examined and protocol dimensions and algorithm performance, frame sizes and algorithm performance and human readability and coding levels were evaluated. The examined protocols and JTP protocol were compared by doing tests in terms of comparison criteria and functional properties. It has been shown that JTP features a higher number of functional properties in comparison to alternatives.

Keywords: Protocol, JSON, Broker Server, TCP, STOMP, XMPP, RESP, NATS, Real-Time Communication

TEŞEKKÜR

Bu çalışmanın gerçekleştirilmesinde, değerli bilgilerini benimle paylaşan, kullandığı her kelimenin hayatıma kattığı önemini asla unutmayacağım saygıdeğer danışman hocam; Prof. Dr. Semih BİLGEN'e, çalışma süresince tüm zorlukları benimle göğüsleyen ve hayatımın her evresinde bana destek olan sevgili eşim Fatma AYAZ'a sonsuz teşekkürlerimi sunarım.

İÇİNDEKİLER

I. TABLO LİSTESİ

II. ŞEKİL LİSTESİ

III. KISALTMALAR

CRLF: Carriage Return Line Feed

FTP: File Transfer Protocol

HTTP: Hyper Text Transfer Protocol

IoT: Internet of Things

JSON: JavaScript Object Notation

JTP: JSON Transmission Protocol

QoS: Quality of Service

RESP: Redis Serialization Protocol

SIP: Session Initiation Protocol

SMTP: Simple Mail Transfer Protocol

STOMP: Simple Text Orientated Messaging Protocol

TCP/IP: Transmission Control Protocol / Internet Protocol

UDP: User Datagram Protocol

UTC: Universal Time Coordinate

XML: Extensible Markup Language

XMPP: Extensible Messaging and Presence Protocol

I. GİRİŞ

İletişim protokolleri bir mesajı veya veriyi diğer bir uç noktaya güvenli ve tutarlı bir şekilde ulaştırmak amacıyla kullanılır. Ancak pratikte bir verinin sadece diğer noktaya iletilmesi yeterli değildir. Veri iletiminin yanı sıra iletim kontrollerini de üstlenecek bir yapıya ihtiyaç duyulmaktadır. Bu tip iletişim sistemleri, kurumsal uygulamalarda veri kaybına bağlı hata toleransı sıfıra yakın, kritik görevler üstlenen sunucular arasında iletişimin sağlanması, çok sayıdaki istemciden (IoT cihazları, loglayıcı vb.) verinin toplaması, aynı anda birden fazla hedefe gerçek zamanlı veri aktarımının sağlanmasında, mikroservis [9] tabanlı altyapılardaki servisler arası iletişimin sağlanması gibi amaçlarla kullanılmaktadır. İletişim protokolleri ikili (*binary*) ve metin tabanlı olmak üzere ikiye ayrılır. Metin tabanlı protokollerin programlama dillerinden ve işletim sistemlerinden bağımsız olmaları tüm platformlar arası iletişim durumlarında kullanılmalarına olanak tanımaktadır.

Aracı sunucu (*broker server*), kendisine bağlanan istemciler arasında veri yönlendirmesi yaparak istemcilerin birbiriyle haberleşmesini sağlayan sunucu yazılımıdır. Bir verinin birçok uç noktaya iletilmesinde Bölüm 2.1.2'de belirtilen TCP/IP protokolündeki Unicast, Multicast ve Broadcast'de karşılaşılan problemler aracı sunucu ile çözülmektedir. Ayrıca aracı sunucu ile TCP/IP'nin bir özelliği olan gerçek zamanlı iletişim de desteklenmektedir.

Bu çalışmada, önce, piyasada kullanılan aracı sunucusuna sahip metin tabanlı protokollerin özellikleri incelenerek avantaj ve dezavantajları araştırılmıştır. Araştırma sonucunda genel olarak protokollerin iletişim için yüksek veri boyutu gerektirmesi, kompleks iletişim tiplerini desteklememesi, protokol sözdizimlerinin insanlar tarafından okunup yazılamaması gibi dezavantajlarının olduğu görülmüştür.

Çalışmada daha sonra bu temel dezavantajları ortadan kaldıran veya en aza indiren ve bir aracı sunucuyla çalışarak aşağıdaki özellikleri içeren gerçek zamanlı iletişim için kullanılmak üzere JSON (JavaScript Object Notation) [1] sözdizimli yeni bir protokol önerilecektir. Bu protokol, metnin devamında JTP (JSON Transmission Protocol) olarak anılacaktır. Önerilen JTP protokolünün işlevsel özellikleri şunlardır:

- Gerçek zamanlı çok hedefli iletişim desteği
- Platform bağımsız veri iletimi
- Yayımcı-abone tipi iletişim desteği
- Arayan-aranan tipi iletişim desteği
- Yük dengeleyicili kuyruk desteği
- Süre limitli veri iletimi desteği
- QoS (Quality of Service) iletim denetimi desteği
- Geriye dönük mesajlara erişim desteği
- Protokolün genişletilebilir olması
- Verilerin gerçek zamanlı sıkıştırılarak iletimi
- Kimlik doğrulama desteği
- İletişim tipi ve konu bazlı yetkilendirme

- İzleme desteği

JTP protokolünü kullanarak çalışan aracı sunucu ve bu aracı sunucuyla haberleşmeyi sağlayan istemci yazılımı da bu çalışma kapsamında gerçekleştirilecektir.

Ayrıca aracı sunucunun fonksiyonel testleri, performans testleri ve diğer alternatifleri ile olan performans karşılaştırmaları da çalışma kapsamında sunulacaktır.

II. LİTERATÜR ARAŞTIRMASI

2.1.Araştırma Kriterlerinin Belirlenmesi

Araştırma kapsamının belirlenmesinde aşağıdaki kriterler göz önünde bulundurulmuştur:

- Metin tabanlı veri iletimi
- Aracı sunucu üzerinden haberleşmesi
- Gerçek zamanlı iletişim

2.1.1. Metin tabanlı protokoller

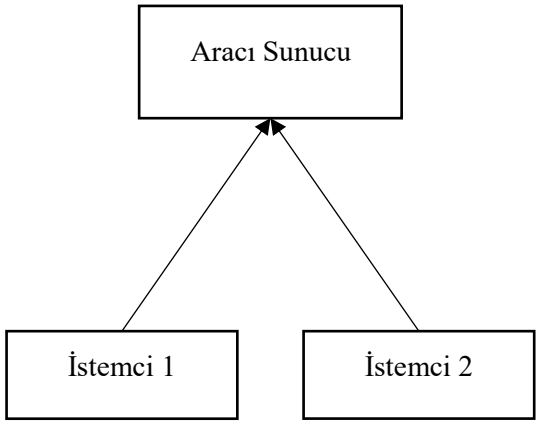
Alfabedeki harfler, rakamlar ve noktalama işaretlerinden oluşan karakter grubuna metin denir. İki nokta arasında haberleşme yapılabilmesi için gelen ve giden verilerin anlamlandırılabilmesi ve iki tarafın da bir kontrat ile anlaşmasına protokol denir. Metin tabanlı protokol ise sadece metin karakterleri ile oluşturulan protokole denir [16]. Bu protokol verilerinin diğer uca iletilebilmesi için çerçevelenmesi gerekiyorsa çerçevenin de metin tabanlı bir protokol ile iletilmesi gerekmektedir.

2.1.2. Aracı sunucu üzerinden haberleşme

TCP/IP [15] protokolü RFC 1180 koduyla standart bir protokoldür. TCP/IP protokolü Unicast, Multicast ve Broadcast olmak üzere 3 çeşit haberleşme metodunu destekler. Unicast metodu ile bir istemci sadece bir sunucuya bağlanarak sadece birbirleriyle veri alışverişi yapmaktadır. Bir istemci birden fazla uç noktaya veri transfer etmek istediğinde Multicast ve Broadcast metodları kullanılmaktadır. Broadcast metodunda gönderilen veri tüm uç noktalara iletilir. Eğer iletilen bu verilere uç noktaların ihtiyacı yoksa ağ trafiği gereksiz yere meşgul edilmektedir ve uç noktaların kaynakları gereksiz yere tüketilmektedir. Multicast metodunda,

Multicast metodunun kullanımı için belirlenmiş IP blokundan bir IP seçilir ve bu IP adresi ağ elemanlarına bildirilerek grup kurulmaktadır. Gönderici, bu IP adresine veri gönderdiğinde grup IP adresine üye olan tüm uç noktalara ağ elemanları vasıtasıyla veri aktarımı yapılmaktadır. Göndericinin gönderdiği tüm veriler üye olan uç noktalara iletilir. Multicast bu yönüyle Broadcast metoduna göre daha az ağ trafiğini meşgul etmektedir. Ancak uç noktalar, üye oldukları IP adresi grubundan belirli bir port numarası ile kendilerine ulaşan veriler üzerinde ihtiyacı olmayan verileri filtreleme işlemi yapamadığından ağ trafiği gereksiz yere meşgul edilmektedir. Ayrıca Multicast ve Broadcast metodları UDP kullanması sebebiyle gönderilen verilerin uç noktalara iletilme denetimi bulunmamaktadır.

Aracı sunucu [17] üzerinden temel haberleşme mimarisinde bir adet aracı sunucu bulunur ve tüm istemciler bu aracı sunucuya TCP ile bağlanır. Bir istemciden başka bir istemciye veri transferi yapabilmek için veri önce aracı sunucuya gönderilir. Ardından aracı sunucu gelen verinin hangi istemci veya istemcilere gönderileceğini, gönderen istemcinin alıcı istemciye gönderebilme yetkilerini, alıcı istemcinin o an aracı sunucuya bağlı olup olmadığı gibi bazı analizleri yaparak mesajı belirli istemcilere dağıtmaktadır. Böylelikle bir verinin birden fazla uç noktaya iletiminde Unicast, Multicast ve Broadcast'de karşılaşılan veri dağıtımındaki problemleri çözmektedir. Aracı sunucunun bağlantı mimarisinin diyagramı Resim 1'de gösterilmiştir. Aşırı yüke sahip aracı sunucularının yükünü azaltmak veya başka bir ağ lokasyonundaki istemcilerin aracı sunucuyla daha hızlı iletişim kurabilmesi için birden fazla aracı sunucu küme yapısında çalışabilmektedir. Küme yapısındaki aracı sunucuları kendi aralarında iletişim kurabilmektedir.



Şekil 1. Aracı sunucunun bağlantı mimarisi

2.1.3. Gerçek zamanlı iletişim

İletişimin gerçek zamanlı olabilmesi için gönderici istemciden gönderilen verilerin, alıcı istemciye ulaşması eşzamanlı veya neredeyse eşzamanlı olarak gerçekleşmelidir [18]. Birçok kaynakta gerçek zamanlı işlem tanımında "uygulamanın gerektirdiği ölçüde gecikmesiz olarak" [41] gibi ifadeler kullanılmaktadır. Bu kimi uygulamalarda mikrosaniyeler ya da daha kısa süreler, kimilerinde ise milisaniyeler ya da daha uzun süreler düzeyinde olabilmektedir. Ancak genellikle kabul edilen ölçüt, gerçek zamanlı iletişim olabilmesi için gönderilen verinin alıcıya iletilene kadar hiçbir yerde geçici olarak saklanmaması şeklinde belirtilmektedir [39]. Bunun yapılabilmesi için tüm istemcilerin TCP protokolüyle aracı sunucuya bağlandıktan sonra bağlantılarını sonlandırmamaları gerekmektedir. Gerçek zamanlı iletişime örnek olarak mobil telefonlar üzerinden sesli iletişim, anlık mesajlaşma uygulamaları (WhatsApp, Telegram vb.), video konferans yazılımları (Skype, Hangout vb.) gösterilebilir. Çünkü bir uç birimden aracı sunucuya gönderilen veri, diğer istemciye aracı sunucu üzerinden hiçbir yere kaydedilmeden direk olarak gönderilerek gerçek zamanlı iletişim sağlanır. Bu bağlamda önerilen aracı sunucunun gerçek zamanlı iletişimi sağlayabilmesi için verinin veriyi gönderen uç birimden ağ katmanına çıktıktan sonra aracı sunucuya iletildiğinde iletilen veri hiçbir yere kaydedilmeden

gönderilecek diğer uç birimlerin soket tamponuna asenkron olarak yazılıp gönderilmektedir.

Gerçek zamanlı iletişim, yarım çift yönlü ve tam çift yönlü olmak üzere iki çeşittir [39]. Yarım çift yönlü gerçek zamanlı iletişimde bir seferde bir yönde iletişim sağlanmaktadır. Bu iletişimde gönderici uç birimleri verileri aynı anda gönderebilirler ancak aynı anda diğer uç birim tarafından veriler alınamamaktadır. Tam çift yönlü gerçek zamanlı iletişim tipinde ise gönderen uç birim ile diğer alıcı uç birimler aynı anda veri gönderip alabilmektedir. Bu çalışmada önerilen aracı sunucu bağlamında tam çift yönlü iletişime destek verilmektedir.

2.2.Protokolleri Karşılaştırma Kriterleri

Her protokolün çeşitli avantaj ve dezavantajları bulunmaktadır. Mesajlar ağ üzerinde bir uçtan diğer bir uca aktarılırken protokol ve çerçeve gibi ek veriler eklenerek aktarıldığından protokol ve çerçeve verilerinin boyutu iletişim performansını etkilemektedir [20]. Ayrıca protokol ve çerçeve verilerinin kodlama ve çözümleme işleminde kullanılan algoritmanın karmaşıklığı işlemci kaynaklarını tüketmesi nedeniyle iletişim performansını etkilemektedir [21]. Protokol ve çerçevenin kodlama ve çözme işleminin insanlar tarafından yapıldığında protokol ve çerçevede kullanılan harf ve sayıların dışındaki tüm karakter tiplerinin toplam sayı değeri insanlar tarafından anlaşılabilirliğin ölçütü olarak belirlenmiştir [19]. Bu çalışmanın kapsamına yönelik aşağıdaki karşılaştırma kriterleri belirlenmiştir.

- Protokol boyutu
- Protokol kodlama ve çözümlemede algoritma performansı
- Çerçeve boyutu
- Çerçeve kodlama ve çözümlemede algoritma performansı
- İnsanlar tarafından anlaşılabilmesi ve yazılabilmesi

2.2.1. Protokol boyutu

Transfer edilecek mesaj veya mesajların belirli bir sözdizimi ile kodlanmasıyla oluşan protokolün veri boyutu bayt cinsinden karşılaştırılmıştır.

2.2.2. Protokol kodlama ve çözümlemede algoritma performansı

Protokol verisi belirli bir sözdizimine dayanan metin kümesinden meydana gelmektedir ve bu metin kümesinden protokol verisinin içerdiği mesaj veya mesajları alabilmek veya yeniden protokol sözdiziminde kodlamak için kullanılan algoritmanın Big-O analizi yapılarak karşılaştırılmıştır [21].

2.2.3. Çerçeve boyutu

Ağ üzerinden veriler bir akış şeklinde gönderilmektedir. Birden fazla protokol verisini bir uçtan diğer bir uca aktarırken birbirinden ayırabilmek için her bir protokol verisi çerçevelenerek gönderilmektedir. Her protokolün kendine ait bir çerçeveleme tipi bulunmaktadır. Her çerçeveleme tipi için farklı uzunluklarda karakterler kullanılmaktadır. Çerçeve boyutuna göre protokoller bayt cinsinden karşılaştırılmıştır.

2.2.4. Çerçeve kodlama ve çözümlemede algoritma performansı

Araştırılan protokollerin kendilerine ait bir çerçeve tipi bulunmaktadır. Her çerçeve tipinin ayrı bir kodlama ve çözümleme algoritması mevcuttur ve her çerçeve kodlama-çözümleme algoritması farklı performans değerlerine sahiptir. Her mesajın istemciden gönderilirken ve istemciye gelen mesajın çerçevesi çözülürken bu algoritma çalışmaktadır. Algoritmanın hızlı veya yavaş olması iletişim hızını etkilemektedir. Bu yüzden protokollerin, çerçeve kodlama ve çözümlemedeki performans değerleri Big-O karmaşıklık analizi yapılarak karşılaştırılmıştır [21].

2.2.5. İnsanlar tarafından anlaşılabilmesi ve yazılabilmesi

Protokolün insanlar tarafından okunup yazılabilmesi için öncelikle metin tabanlı bir protokol olması gerekmektedir. Çünkü insanların bilgisayar ve terminallere veri girebilmek için standart bir klavye kullanıldığı düşünüldüğünde sadece alfabe karakterleri, rakamlar ve noktalama işaretleri girebilmektedirler yani sadece metin tabanlı giriş yapabilmektedirler. Bazı klavyelerde özel tuş kombinasyonları veya yazılım desteğiyle standart veri girişi dışında veriler de girilebilmektedir ancak böyle bir kullanım bu çalışmanın kapsamı dışındadır.

Metin tabanlı veri girişi yapılabilmesi protokolün insanlar tarafından kolayca okunup yazılabilmesi anlamına gelmemektedir. Protokol kodlamak için kodlanacak mesajın dışında yazılan her karakter tipi protokolün öğrenilmesini, hafızada tutmayı ve kullanım kolaylığını düşürmektedir. Bu çalışmada gönderilecek mesaj dışında protokol ve çerçeveyi oluşturan harf ve rakamların dışındaki veri tiplerinden kaç adet girildiği bilgisi baz alınarak insanlar tarafından anlaşılabilme ve yazılabilme kriteri analiz edilecektir. Analiz sonucunda bulunan değer ne kadar az ise o oranda insanlar tarafından kolayca kodlanabildiği kabul edilecektir [19].

2.3.Araştırılan Protokoller

Bölüm 2.1'de belirtilen araştırma kriterlerine göre bu çalışmada aşağıdaki protokoller araştırılmıştır. Bu kriterlere uymayan diğer protokoller bu çalışmanın kapsamı dışındadır. Araştırma kriterlerinin bir kısmını karşılayıp bir kısmını karşılamayan protokoller de mevcuttur ve Bölüm 2.4'de belirtildiği gibi bunlar da kapsam dışı bırakılmıştır. Bölüm 2.4'de verilen protokoller örnek vermek amacıyla bu çalışmada bahsedilmiştir ve Bölüm 2.4'te belirtilen özelliklere sahip olan diğer protokollere yer verilmemiştir. Araştırılan protokolleri birbirleri arasında adil bir şekilde karşılaştırmak için hepsinin desteklediği iletişim tipi olan yayımcı-abone tipi

iletişimin en az parametrelerle “Merhaba Dünya” verisini “konular/konu1” isimli konuya gönderme örneği kullanılmıştır.

- STOMP
- XMPP
- RESP
- NATS

Aşağıda önce her bir protokol ayrı ayrı ele alınacak, ardından, karşılaştırma sonuçları özet çizelge şeklinde sunulacaktır.

2.3.1. STOMP (Simple Text Orientated Messaging Protocol)

STOMP, aracı sunucu ile istemciler arasında eşzamansız mesaj transfer etmek için kullanılan basit protokol tasarımıdır [2]. Basit komutsal iletişim yapabilmek için tasarlanmıştır. Günümüzde ihtiyaç duyulan birtakım karmaşık iletişim işlemlerini desteklememektedir. STOMP protokolünün 1.2 versiyonuna göre kullanılan komutlar "SEND", "SUBSCRIBE", "UNSUBSCRIBE", "BEGIN", "COMMIT", "ABORT", "ACK", "NACK", "DISCONNECT", "CONNECT", "STOMP" olmak üzere 11 komut ile işlemler yapmaktadır. Protokol 4 ana kısım olmak üzere komut, parametreler, gövde ve bitiş karakteri kısımlarından oluşmaktadır. Buna binaen protokolün genişletilebilme özelliği bulunmamaktadır. Örnek bir STOMP protokolüne uygun bir paket yapısı aşağıdaki gibidir:

```
COMMAND
header1:value1
header2:value2

Body^@
```

Karşılaştırma testleri için yayımcı-abone tipi iletişime ait veri gönderme örneği aşağıdaki gibidir:

```
SEND\r\n
destination:konular/konu1\r\n
content-type:text/plain\r\n
\r\n
Merhaba Dünya^@
```

Veri örneğinin çerçeve ve protokol verilerinin ayrıştırılması gerekmektedir. Çerçevenin sonunu belirtmek için 2 bayttan oluşan "^@" karakterleri kullanılmaktadır. Dolayısıyla çerçeve çözümlenirken gelen her bir bayt incelenerek "^@" karakterleri aranacaktır. Bu yüzden çerçeve çözümlemedeki algoritma karmaşıklığı Big-O notasyonuna göre O(n)'dir. Çerçeve oluşturulurken verilerin sonuna sadece "^@" karakterleri ekleneceğinden algoritma karmaşıklığı O(1)'dir. Çerçeve karakterleri olan "^@" karakterleri dışındaki tüm veriler protokol kısmını oluşturmaktadır. Tüm veri 13 bayt mesaj, 60 bayt protokol ve 2 bayt çerçeve olmak üzere 75 bayttan oluşmuştur. Gönderilen veri ve veri hakkında bilgiler protokol verileriyle gönderilmektedir. Gönderilen mesaja ait her bir bilginin sonuna "\r\n" karakterleri bulunmaktadır ve bu bilgi listesi bittikten sonra yeniden "\r\n" karakterleri eklenerek mesaj hakkındaki bilgilerin bittiğini ve mesajın gönderilmeye başlanacağını bildirmektedir. Protokolün performansı çözümlenirken gelen her baytta "\r\n" karakterleri aranacağından algoritma karmaşıklık değeri O(n) olarak belirlenir. Protokol verisini oluştururken ise verilerin sonuna sadece "\r\n" eklendiğinden O(1) olarak belirlenir. İnsanlar tarafından protokolün anlaşılabilmesi ve kodlanabilmesi için gönderilen mesaj dışındaki protokol ve çerçeveyi oluşturan tüm veriler incelenmelidir. Bu durumda protokol ve çerçeve verisinde "\r", "\n", ":", "/", "^" ve "@" olmak üzere 6 farklı tipte karakter bulunmaktadır.

2.3.2. XMPP (Extensible Messaging and Presence Protocol)

XMPP, XML tabanlı bir protokoldür [3]. Protokol, açık kaynak olarak yayınlanmıştır ve XEP-0001 kodlarıyla standartlaştırılmıştır [4] [11]. 1998 yılında geliştirilmesinden beri başta Google olmak üzere birçok alanda kullanıldığı için kendini kanıtlamıştır. SASL (Simple Authentication and Security Layer) ve TLS

(Transport Layer Security) kullanılarak veri güvenliği sağlanmıştır. Genişletilebilir bir yapıya sahip olması sebebiyle protokolün çekirdek özelliklerinin üzerine özel işlemler oluşturulabilir. Metin kodlama olarak UTF-8 kullanması sebebiyle tüm dillere destek vermektedir. Çift yönlü iletişim, Yayıncı-abone iletişimi [12] ve çok kullanıcılı grup iletişim olmak üzere 3 çeşit iletişim tipine sahiptir.

XML sözdizimi ile iletişim amaçlı kullanıldığında yapısı gereği kodlanacak mesaj boyutunun üstünde bir boyuta sahip olmaktadır. XML sözdiziminin yapısı gereği CSV [5], JSON vb. sözdizimleriyle karşılaştırıldığında veri boyutunun yüksek olması bir dezavantaj olarak görülmüştür. Bu sebeple veri aktarımı yapılırken yüksek bant genişliğine ihtiyaç duymaktadır ve bu da ağ performansını düşürmektedir. Genellikle mesajlaşma yazılımlarının haberleşme protokolü olarak tasarlanmıştır. Bu yüzden bir istemci oturum açarken bir kimlik bilgisi (genellikle e-posta) girer ve bu kimliğe gönderilen mesajlar anlık olarak oturum açan istemciye iletilebildiği gibi başka bir istemciye de mesaj gönderilebilir. XMPP protokolü ile yayımcı-abone tipi iletişim için mesaj gönderme örneği [12] aşağıdaki gibi konu bilgisi ve mesaj içeriği düzenlenerek yeni bir XMPP protokol verisi oluşturulmuştur.

```
<message from='pubsub.shakespeare.lit'
to='denmark.lit/konular_konu1' id='test'>
  <event
xmlns='http://jabber.org/protocol/pubsub#event'>
    <items node='princely_musings'>
      <item id='ae890ac52d0df67ed7cfdf51b644e901'>
        <entry xmlns='http://www.w3.org/2005/Atom'>
          <title>Test</title>
          <summary>
            Merhaba Dünya
          </summary>
          <link rel='alternate' type='text/html'
href='http://denmark.lit/2003/12/13/atom03'/>
          <id>tag:denmark.lit,2003:entry-32397</id>
          <published>2003-12-13T18:30:02Z</published>
          <updated>2003-12-13T18:30:02Z</updated>
        </entry>
      </item>
    </items>
  </event>
</message>
```

Protokol incelendiğinde mesajı diğer mesajlardan ayırmaya yarayan çerçeve verileri için XML yapısı kullanıldığı görülmektedir. Çerçevenin başlangıcı için

“<message>” karakterleri kullanılırken çerçeve bitişini ifade etmek için “</message>” karakterleri kullanılmıştır. Bu yüzden çerçeve boyutu 19 bayttan oluşmaktadır. Çerçeve verileri aynı zamanda protokolün de verileridir. Mesaj boyutu 13 bayt olduğundan geriye kalan 520 baytlık veri protokol verilerini oluşturmaktadır. Çerçeve çözümleme için “<message>” karakterlerini okunduktan sonra “</message>” karakterleri gelene kadar gelen her bayt için karşılaştırma yapıldığından çerçeve çözümlemenin algoritma karmaşıklığı O(n) olarak belirlenir. Çerçeve kodlamak için ise sadece veri birleştirme yapıldığı için O(1) olarak belirlenir. Çerçeve ve protokol XML yapısında olduğundan çerçevedeki algoritma karmaşıklık değerleri protokol çözümleme ve kodlamada da aynı karmaşıklık değerleri içermektedir. İnsanlar tarafında anlaşılabilme ve kodlanabilmesi incelendiğinde mesaj dışındaki protokol ve çerçevede kullanılan özel karakter tipleri “<”, “=”, “””, “>”, “#”, “_”, “:”, “.” ve “/” olmak üzere 9 adet karakter tipi bulunmaktadır.

2.3.3. RESP (Redis Serialization Protocol)

RESP (Redis Serialization Protocol), Redis sunucusuyla istemci arasındaki iletişim amacıyla tasarlanmıştır [6]. Ancak diğer istemci-sunucu yazılımlarında da bu protokol kullanılabilmektedir. Metin tabanlı ve belirli bir standarta sahip olmayan bir protokoldür. Ayrıca protokolün yapısı gereği verilerin hangi alanı temsil ettiği bilgisi olmaması sebebiyle kompleks veri yapıları bu protokol ile tanımlanamamaktadır [13]. Ancak birçok komut seti ile her desteklenen işlem için farklı komut girilerek işlemler gerçekleştirilir [13]. Uygulanmasının basit olması ve ayrıştırma işleminin hızlı yapılması protokolün ana amaçları olarak belirlenmiştir. Ana çalışma prensibi bir sorgu iletisi gönderildiğinde geriye basit bir cevap mesajı gönderme şeklindedir. İstisna olarak birden fazla sorgu yapıldığında sorgular işlendikçe cevap mesajı gönderilir veya yayıncı-abone olarak bir konu dinlendiğinde mesajlar geldikçe istemciye gönderilir. Sunucu istemciye mesaj gönderirken protokol verisinin içerisinde ne tür bilgi taşıdığı ilk karakter ile belirlenir ve “+” (metin), “-“ (hata mesajı), “:” (tamsayı), “$” (çoklu metin), “dir *” (dizi) ifade etmek için kullanılır. Sözdiziminin sonraki karakterleri gönderilecek veriyi içerir ve mesajı sonlandırmak

için mesajın sonuna “\r\n” (CRLF) eklenmektedir. Sunucudan cevap olarak ise mesaj uygun bir şekilde işlenmişse “+OK\r\n” mesajı dönerken hatalı bir işlem yapıldığında “-Hata Mesajı\r\n” şeklinde mesaj gönderilmektedir. RESP protokolü ile yayıncı-abone iletişim tipi ile mesaj yayınlama örneği aşağıdaki gibidir:

```
PUBLISH konular.konu1 Merhaba Dünya\r\n
```

Protokol karşılaştırma için incelendiğinde verinin sonunu belirtmek için 2 bayttan oluşan “\r\n” karakterleri kullanıldığından bu karakterler çerçeve verileri olduğu görülmektedir. Verilerin çerçeve çözümlemesi için gelen her bayt analiz edilerek içerisinde “\r\n” karakterleri arandığından çerçeve çözümleme algoritma karmaşıklığı O(n) olarak belirlenir. Çerçeve kodlanırken de protokol verilerinin sonuna sadece “\r\n” karakterleri eklendiğinden kodlamanın algoritma karmaşıklığı O(1) olarak belirlenir. Mesaj boyutu 13 bayttan oluşmaktadır ve çerçeve dışında kalan veri protokol verisi olduğundan protokol verisinin boyutu 22 bayttan oluşmaktadır. Protokol verisi incelendiğinde komut ve konuta ait parametreler arasında birer boşluk karakteri bulunmaktadır. Son boşluk karakteri ile çerçeve arasında kalan kısımda mesaj verisi bulunmaktadır. Dolayısıyla protokol verisi çözümlenirken her karakter kontrol edilerek boşluk karakteri aranacağından algoritma karmaşıklığı O(n) olarak belirlenir. Aynı şekilde protokol kodlanırken komut ve parametrelerin sonuna sadece boşluk karakteri ekleneceğinde algoritma karmaşıklığı O(1) olarak belirlenir. İnsanlar tarafından anlaşılması ve kodlanması incelendiğinde “ “, “.”, “\r” ve “\n” olmak üzere 4 adet karakter tipi bulunmaktadır.

2.3.4. NATS protokolü

NATS protokolü metin tabanlı bir protokoldür [14]. Basit bir protokol tasarımına sahiptir. Ana kullanım amacı yayıncı-abone iletişim tipi ile diğer istemcilerle iletişim kurmaktır. Herhangi bir standart ile tanımlanmamıştır. Protokol yapısı RESP protokolüne benzemektedir ancak bazı sözdizimi ve işlevsellik özellikleri bakımından farklılıkları bulunmaktadır. Kompleks veri yapıları bu protokol ile

tanımlanamamaktadır [14]. “INFO, CONNECT, PUB, SUB, UNSUB, MSG, PING, PONG, OK, ERR” olmak üzere 10 adet komut ile haberleşme yapılmaktadır. Her komut sonu “\r\n” karakterleri ile bitmelidir. Gönderilen komut sonrası sunucudan komut gönderen istemciye olumlu veya olumsuz cevap dönmektedir. Eğer komut sonucu olumluysa “+OK\r\n” cevabı dönülürken eğer olumsuzsa “-ERR ‘Hata Mesajı’\r\n” cevabı dönülmektedir. Yayıncı-abone tipi iletişim ile mesaj göndermek için örnek veriler aşağıdadır:

```
PUB konular.konu1 13\r\nMerhaba Dünya\r\n
```

NATS protokolü incelendiğinde verinin çerçeve bilgisi sondaki 2 bayttan oluşan “\r\n” karakterleridir. Mesaj bilgisi 13 bayttan oluşmaktadır ve geri kalan 22 bayt protokol verilerini içermektedir. Çerçeve çözümleme için gelen her baytta “\r\n” karakterleri aranacağından algoritma karmaşıklığı O(n) olarak belirlenir. Çerçeve kodlamada ise protokol verisinin sonuna sadece “\r\n” ekleneceğinden O(1) olarak belirlenir. Protokol çözümlemede de aynı şekilde gelen her baytta “\r\n” ve “ “ karakterleri aranacağından algoritma karmaşıklığı O(n) olarak belirlenir. Protokol kodlamada ise komutlar, parametre ve mesaj aralarına sadece “ “ ve “\r\n” karakterleri eklendiğinden algoritma karmaşıklığı O(1) olarak belirlenir. İnsanlar tarafından anlaşılma ve kodlanabilmesi incelendiğinde “ “, “.”, \r” ve “\n” olmak üzere 4 veri tipine sahiptir.

2.3.5. Araştırılan protokollerin karşılaştırılması

Araştırılan STOMP, XMPP, RESP ve NATS protokollerinin bu çalışmada belirlenen karşılaştırma kriterlerine göre değerlendirmesini özetleyen tablo aşağıda verilmiştir.

Tablo 1: Araştırılan protokollerin karşılaştırılması

	STOMP	**XMPP**	**RESP**	**NATS**
Çerçeve Boyutu	2 bayt	19 bayt	2 bayt	2 bayt
Çerçeve Kodlama	O(1)	O(1)	O(1)	O(1)
Çerçeve Çözümleme	O(n)	O(n)	O(n)	O(n)
Protokol Boyutu	60 bayt	520 bayt	22 bayt	22 bayt
Protokol Kodlama	O(1)	O(1)	O(1)	O(1)
Protokol Çözümleme	O(n)	O(n)	O(n)	O(n)
İnsanlar Tarafından Kodlanabilme Değeri	6 veri tipi	9 veri tipi	4 veri tipi	3 veri tipi

2.4.Kapsam Dışı Bırakılan Bazı Protokoller ve Özellikleri

Araştırılan protokoller arasında araştırma kriterlerinden bir veya birkaçını sağlayan ancak tüm kriterleri tam olarak sağlamayan protokoller aşağıdadır.

- HTTP
- WebSocket
- SMTP
- SIP
- NSQ

2.4.1. HTTP (Hyper-Text Transfer Protocol)

Bu protokol metin tabanlı bir protokoldür [22]. Ancak herhangi bir aracı sunucu üzerinden haberleşme yapılmamaktadır. Ayrıca istemci sunucuya sorgu yapıp istemciye sunucudan cevap döndükten sonra TCP bağlantısı kapatıldığından gerçek zamanlı iletişim özelliği bulunmamaktadır.

2.4.2. WebSocket

Bu protokol metin tabanlı bir protokoldür. İnternet tarayıcılarında gerçek zamanlı iletişim amacıyla kullanmak için standartlaştırılmıştır [23]. Ancak bir aracı sunucu üzerinden haberleşmek yerine direk istemci ile sunucu arasında bir haberleşme yapılmaktadır. Ancak sunucu ve istemci yazılımı üzerinde ek yazılımlar yapılarak ve websocket protokolünün mesaj içeriğinde kendi özel protokollerini ekleyerek sunucu aracı sunucu olarak çalışabilmektedir. SocketCluster [10] protokolü gibi protokoller websocket protokolünü bu şekilde kullanarak çalışmaktadır. Ancak bu ve bunun gibi bir kullanım websocket protokolünün bir özelliği olmadığından bu çalışmanın kapsamı dışındadır.

2.4.3. SMTP (Simple Mail Transfer Protocol)

Metin tabanlı bir protokoldür ve E-Posta transferlerinde kullanılmaktadır [24]. Bir E-Posta, gönderici istemciden aracı sunucuya gönderilir ve ardından alıcı istemci de aracı sunucuya bağlanarak kendisine gelen E-Posta verisini kendine indirmektedir. Ancak SMTP protokolünü kullanan bu istemciler aracı sunucuya sürekli bağlı olmadıklarından gerçek zamanlı bir iletişim mevcut değildir. Aynı şekilde SMTP protokolüyle aynı özelliklere sahip FTP [25] gibi protokoller de mevcuttur. Bunun gibi benzer özelliklere sahip protokoller irdelenmemiştir.

2.4.4. SIP (Session Initiation Protocol)

Bu protokol de metin tabanlı bir protokoldür. Birebir veya grup olarak istemcilerin birbirleri arasında metinsel, sesli veya görsel olarak iletişim kurmaları için oturum başlatıcı protokoldür [26]. Tüm SIP protokolünü kullanan istemciler aracı sunucuya sürekli bağlıdırlar ve birbirleriyle gerçek zamanda iletişim halindelerdir. Ancak oturum başlatıldıktan sonra iki veya daha fazla istemci aracı sunucu üzerinden haberleşmezler ve birbirleriyle direk ağ üzerinden bağlantı kurarak SIP veya farklı bir protokolde haberleşme yapılmaktadır. İstemciler arası metinsel bir iletişim yapılacaksa SIP protokolü bildirim, yayımcı-abone ve mesaj olarak üç tipte mesaj tipine destek vermektedir. Ancak sesli, görüntülü mesajlaşma için RTSP protokolü kullanılabildiği gibi başka bir metinsel mesajlaşma protokolü veya istemcilerin kendi aralarında oluşturdukları özel bir protokol de kullanılabilir. SIP protokolüyle hangi protokol ile nasıl iletişim kuracaklarını istemciler oturum açarken bildirmektedirler.

2.4.5. NSQ

Bu protokol metin tabanlı olmasına karşın veriler ikili tabanlı olarak çerçevelenmektedir [27]. Sadece metin tabanlı bir protokol yapısından oluşmadığı ve hibrit bir protokole sahip olduğu için bu çalışmanın kapsamı dışındadır. Diğer taraftan bir aracı sunucu üzerinden gerçek zamanlı olarak iletişim sağlanmaktadır. Ayrıca bu protokol standart bir protokol değildir. Bu standart olmayan protokol gibi birçok protokol bulunmaktadır ancak örnek vermek amacıyla bu çalışmada sadece NSQ protokolüne yer verilmiş olup diğer protokollere yer verilmemiştir.

III. YENİ PROTOKOL ÖNERİSİ

Temel olarak literatürde araştırılan protokollerin avantajlarının tek bir yerde birleştiği bir protokol önerilecektir. Önerilen protokol JTP (JSON Transmission Protocol) olarak adlandırılmıştır. Bu bölümde JTP işlevsel özellikleri, protokol şartnamesi ve sunucu mimarisi bakımından anlatılacaktır.

3.1.JSON Sözdizimi

JSON (JavaScript Object Notation), metin tabanlı veri transferi için kullanılan bir sözdizimidir [1]. ECMA-404 koduyla standartlaştırılmıştır [8]. İnsanlar tarafından okunabilir ve yazılabilir bir formata sahiptir. Alternatifi olabilecek kompleks verileri formatlayabilme özelliği olan XML'e göre çıktısının boyutu daha az boyutludur. Çoğu programlama dili için kütüphaneleri mevcuttur ve kendini kanıtlamış bir sözdizimidir.

İsim-değer çifti (nesne) ve dizi olarak iki ana yapıdan oluşmaktadır. Bir nesne "{" karakteri ile başlayıp "}" karakteri ile sonlanırken, bir dizi "[" karakteri ile başlayıp "]" karakteri ile sonlanmaktadır. Metin ifadelerinin başında ve sonunda """ karakteri bulunmaktadır. Bir nesne, nesne adı ve değeri olmak üzere iki kısımdan oluşmaktadır. İki eleman arasına her zaman "," karakteri konularak ayrıştırılır. Nesne adı metin tipinde olmalıdır ve değer kısmı ise metin, sayı, nesne, dizi, true, false ve null değerlerini alabilmektedir. Örnek bir JSON sözdizimi aşağıdaki gibidir:

```
[
 "metin",
 100,
 {
  "nesne": "deger"
 },
 [
  "eleman1",
  "eleman2"
 ],
 true,
 false,
 null
]

["metin",100,{"nesne":"deger"},["eleman1","eleman2"],tr
ue,false,null]
```

3.2.JTP Protokolünün İşlevsel Özellikleri

Önerilen JTP protokolünün işlevsel özellikleri Bölüm 1'de özetlenmişti. Bu bölümde işlevsel özelliklerin detayları anlatılmıştır.

3.2.1. Gerçek zamanlı çok hedefli iletişim desteği

İletişimin, yukarıda Bölüm 2.1.3'te verilen tanım çerçevesinde gerçek zamanlı olabilmesi için gönderici ile alıcının aynı anda birbirine veri göndermek ve almak için hazır durumda olması gerekmektedir. Ancak ağ iletişiminin yavaş altyapıya sahip olması, sunucu veya istemcinin düşük performanslı bir donanım barındırması vb. gibi durumlarda neredeyse gerçek zamanlı bir iletişim kurulabilmektedir. Gerçek zamanlı iletişim için TCP ile bağlantı kurulduktan sonra bağlantının koparılmadan iletişim halinde olunması gerekmektedir. Ancak TCP bağlantısı ile sadece 2 uç birim bağlantı kurabilmektedir. Çok hedefli veri aktarımı için öncelikle tüm istemciler aracı sunucuya TCP bağlantısı ile bağlanmaktadır. Bir istemci, aracı sunucuya bağlı başka bir istemci veya istemcilere veri gönderebilmek için kanal yöntemi kullanılmaktadır. İstemciler istedikleri kanallara abone olup herhangi bir istemci bu kanal ismine veri

gönderdiğinde aracı sunucu abone olan tüm istemcilere bu mesajı dağıtarak çok hedefli veri iletimini gerçekleştirmektedir. Gerçek zamanlı iletişim yapılabilmesi için veriyi dağıtma işlemi asenkron olarak dağıtılacak uç birimlerin gönderilmek üzere soket akışına yazılmalıdır ve verinin ilk gönderiminden son iletilen uç birime kadar veri hiçbir yere kaydedilmeden gönderilmektedir. Veriyi gönderen uç birim, gönderdiği verinin diğer uç birim veya birimlere iletilme durumu hakkında Bölüm 3.2.7'de anlatılan QoS iletim denetimi ile bilgi alınabilmektedir.

Yayımcı-abone tipi iletişim gerçek zamanlı çalışan iletişim tipidir. Ancak tüm iletişim tipleri yapıları gereği gerçek zamanlı olarak çalışmamaktadır. Arayan-aranan, yük dengeleyicili kuyruk, süre limitli veri iletimi tiplerinin işlevlerini yerine getirebilmeleri için aracı sunucu tarafından bazı verilerin saklanması gerektiğinden gerçek zamanlı iletişimi tam olarak desteklememektedir.

3.2.2. Platform bağımsız veri iletimi

Gönderilen ve gelen mesajlar insanlar tarafından da rahatça okunabilmeli, komut yapısı oluşturulabilmeli ve kullanıcının gereksinimleri doğrultusunda protokol genişletilebilmelidir. Protokol verilerinin tüm platformlar ve tüm yazılım dilleri tarafından kolayca ayrıştırılabilir olması gerekmektedir. Aynı zamanda büyük miktarlarda veri transfer etme gereksinimi duyan uygulamalar için protokol ve çerçevesi de dahil olmak üzere en az veri ile mesaj iletilmelidir. Bu bilgiler doğrultusunda ECMA-404 standardına sahip olan JSON (JavaScript Object Notation) [1] sözdizimi kullanılacaktır.

3.2.3. Yayımcı-Abone tipi iletişim desteği

Bir mesajın sadece belirli istemciye veya istemcilere çoklu olarak gönderilmesidir. Veri aktarımının yapılabilmesi için öncesinde mesaj almak isteyen istemcilerin almak istedikleri konuya abone olmaları gerekmektedir. Daha sonra bu konuya gönderilen tüm mesajlar abone olan tüm istemcilere gönderilmektedir [38].

3.2.4. Arayan-Aranan tipi iletişim desteği

Bir sunucunun başka bir sunucudan bilgi talep etmesinde kullanılmaktadır. Ayrıca uzaktaki sunucuda bir fonksiyonun çalıştırılmasında da kullanılmaktadır. Sunucu sistemlerinde ve mikro hizmet mimarilerinde [7] çok kullanışlı bir özelliktir.

3.2.5. Yük dengeleyicili kuyruk desteği

Aşırı sunucu yükünü dengelemek için aynı işi yapan birden fazla sunucu çalıştırılarak yük dengelenmeye çalışılmaktadır. HTTP protokollerinde istemciye sorgu cevabı gönderildiğinde bağlantı kapatılmaktadır. Her yeni sorgu isteğinde yük dengeleyici cihazlar o an yükü en az olan sunucuya sorguyu yönlendirmektedir [14]. JTP'nin gerçek zamanlı çalışabilmesi için bağlantının kapatılmadan sürekli TCP bağlantısının açık durumda olması gerekmektedir. Bu probleme çözüm olarak HTTP protokollerinde yük dengeleyici cihazlar kullanılmaktadır. Ancak önerilecek protokolde bu tip protokollerin mimari yapısı gereği sürekli bağlı olan bir bağlantı söz konusu olduğundan yük dengeleyici cihazları işlevsiz kalmaktadır. Bu desteği tekrar sağlamak için çoklu mesaj gönderimlerinde yükü en az olan sadece 1 sunucuya mesaj iletilebilir olmalıdır.

3.2.6. Süre limitli veri iletimi desteği

Mesajların belirli bir zamana kadar geçerlilik süresi olabilecek durumlarla karşılaşılmaktadır. Süre bitiminde mesaj, kuyruktan silinecektir [6].

3.2.7. QoS (Quality of Service) iletim denetimi desteği

Mesajların uçtan uca güvenli bir şekilde ulaşmasını sağlamaktadır. Ayrıca mesajı gönderen istemciye mesajın iletilip iletilemediği hakkında bilgi verilmelidir [30].

3.2.8. Kalp atışı desteği

Soket iletişiminde bağlantının açık olup olmadığı ile ilgili kontroller TCP protokolü kullanılarak yapılmaktadır. Ancak bağlantı kablolarının kopması, iki uç arasındaki ağ cihazlarının arızalanması vb. gibi nedenlerle olağan dışı bağlantı kopmaları yaşanabilmektedir. Eğer kurulmuş bir bağlantı varsa olağan dışı durumlar yaşandığında TCP protokolüyle veri akışı olmadığı için bazı uç noktalar bağlantı kopmuş olsa bile hala diğer uç noktaya yanlış bir bilgi olarak bağlı olduğu bilgisini verebilmektedir. Aynı şekilde sunucu yazılımları da yazılım veya donanım kaynaklı hatalar oluştuğunda hizmet verememektedirler.

Yazılım veya donanım kaynaklı arızalar olduğunda oluşan hizmet kesintisinin tespiti amacıyla kalp atışı yöntemi kullanılan yöntemler arasındadır [14]. Bu yöntem ile aracı sunucuya bağlı tüm istemciler ile aracı sunucu arasında belirli periyotlarla "Aktif misin?" anlamında "PING" komutu gönderilir ve bu komutu alan diğer uç birim "Evet aktifim" anlamında "PONG" komutunu göndermektedir. Eğer PING komutu gönderildiğinde belirli bir süre içinde PONG komutu gelmezse iletişim sona ermiş veya sunucu hizmet vermeyi bırakmış anlamı taşımaktadır.

3.2.9. Protokolün genişletilebilme desteği

Farklı amaçlar için önerilecek protokolün kullanılabilmesi ve gelişen teknolojiye ayak uydurabilmesi için protokol, genişletilebilir bir yapıda olmalıdır. Bunun için JSON sözdiziminde istemci verilerini barındıran kısım metin tabanlı olduğundan genişletilebilir bir yapıda olacaktır.

3.2.10. Verilerin gerçek zamanlı sıkıştırılarak iletimi

Gerçek zamanlı olarak büyük boyutta verilerin ağ üzerinden iletimi yapılırken ağ gecikmeleri yaşanmaktadır. Bu gecikmeyi en aza indirmek için istemciden sunucuya ve sunucudan istemciye veriler sıkıştırılarak gönderilmelidir [31].

3.2.11. Kimlik doğrulama desteği

Sadece yetkili istemciler aracı sunucuya bağlanabilmelidir. Varsayılan olarak bir adet yönetici hesabı ile kurulmaktadır ve diğer kullanıcılar ve bu kullanıcıların yetkilendirme işlemleri yönetici hesabı ile yönetilebilmelidir [32].

3.2.12. İzleme desteği

Yetkisi bulunan istemcilere anlık olarak aracı sunucunun çeşitli istatistiklerini gönderebilmelidir [14].

3.3.Protokol Şartnamesi

JTP protokolü kompakt olarak boşluk karakterleri silinmiş tek satırdan oluşan ve UTF-8 ile formatlanmış JSON sözdizimi ile oluşturulmaktadır. JTP protokolünü çerçevelemek için protokol verilerinin sonuna “\r\n” (CRLF) karakterleri eklenmektedir. Protokolün model yapısı sabittir ve JTP protokolü ile haberleşmek isteyen tüm uç birimler bu modele uymak zorundadır. Ancak istisna olarak protokolün genişletilebilme özelliği kullanılmak istendiğinde modeldeki Bölüm 3.3.7’de anlatılan ve bu işlem için ayrılan alanlar kullanılmalıdır. JTP protokolünün .NET C# dili ile yazılmış modeli EK-A’da verilmiştir. Modelde yer alan alanların açıklamaları aşağıda verilmiştir.

- Topics: Gönderilen verinin hangi kanallar üzerinde işlem yapılacağını belirtmektedir. Liste yapısında olması sebebiyle tek seferde gönderilen komut ile birçok kanal üzerinde işlem yapılabilmesini sağlamaktadır. Varsayılan değeri boş listedir.
- IsCompressed: "Payload" alanına girilen verinin sıkıştırılmış bir veri olup olmadığını diğer uç birime bildirmek için kullanılır. Eğer bu alanın girişi yapılmazsa varsayılan değeri "false" olmaktadır.
- IsReset: Varsayılan olarak "false" değerini alır ve komut gönderirken "Topics", "IsCompressed" ve "Commands" alanlarından her biri için Null olarak gönderilmişse bu değerlerden son gönderilen veriler kabul edilerek otomatik olarak son gönderilen veri olarak tanımlanmaktadır. Böylelikle kendini tekrar eden komutlarda ağ trafiğini düşürülerek iletişim hızlandırılmaktadır. Eğer "true" olarak belirlenirse eskiden gönderilen verileri yok sayarak sadece gönderilen paketteki verileri işleme almaktadır. İlk defa karşı uç birime komut paketi gönderiliyorsa veya hem bu parametre "true" girilmiş hem de herhangi bir komut bilgisi girilmemişse komut parametrelerinin varsayılan değerleri kabul edilmektedir.
- Commands.QoS: İletim denetimini ayarlamaktadır. Bu alanın detayları Bölüm 3.3.5'te anlatılmıştır. Varsayınlan değeri "AtMostOnce" olarak belirlenmektedir.
- Commands.CommandType: Komut tipini belirlemektedir. Varsayılan değeri "Publish" olarak belirlenmektedir.
- Commands.CommandParameters: Commands.CommandType alanına girilen veriye göre uygun olan "BaseParameterModel" sınıfından türemiş PublishParameterModel, SubscribeParameterModel, UnsubscribeParameterModel, CallerParameterModel, CalleeParameterModel, PushParameterModel, PopParameterModel, SignInParameterModel, PingParameterModel, PongParameterModel, CustomCommandParameterModel sınıflarından biri ile veri girişi yapılmaktadır. İlgili komut tipi için komutla ilgili diğer parametreler bu alanın içinde yer almaktadır.

- Commands.CommandParameters.CustomParameters: Protokolün genişletilebilme özelliği kullanılmak istediğinde genişletilmiş protokolün özelliklerine ait alanları içermektedir. Genişletilebilme özelliği kullanılarak yeni bir protokol oluşturulabildiği gibi desteklenen protokollerde de veri girişi yapıldığında ek özellikler sağlanabilmektedir.
- Commands.CommandParameters.IsRetain: Bu parametre "true" olarak girilerek mesaj yayınlandığında ilgili konuya daha sonradan abone olan uç birimlere son gönderilen mesaj tek seferliğine o an yayınlanmış gibi iletilmektedir [35].
- Commands.CommandParameters.SenderSessionId: Sorguyu yapan uç birime ait aracı sunucu tarafından oluşturulan istemcinin seri numarasıdır. "CALLER" komutu aracı sunucuya gönderilirken bu parametreye veri girilmemektedir ve aracı sunucudan ilgili uç birime gönderilirken aracı sunucu tarafından otomatik olarak doldurularak sorguyu cevaplayabilen uç birime gönderilmektedir. "CALLEE" komutunda bu parametre sorguyu cevaplayan uç birim tarafından sorguyu yapan uç birimin seri numarası girilmektedir.
- Commands.CommandParameters.ExpiryDateTime: Kanala ait kuyruğa bir veri eklenirken eklenen verinin geçerlilik tarihini bildirmektedir. Girilen tarih bilgisinden sonraki bir tarihte "POP" komutuyla kuyruktan veri alma komutu gönderildiğinde ilgili verinin tarihi geçersiz olduğu için kuyruktaki bir sonraki veri geri döndürülmektedir.
- Commands.CommandParameters.IsDeleted: Kuyruktan veri alındığında alınan verinin kuyruktan silinip silinmeyeceğini belirtmektedir.
- Commands.CommandParameters.IsWaitForPop: Kuyruktan veri alma talebi yapıldığında eğer kuyrukta veri yoksa yeni veri eklenene kadar ve kendisine sıra gelene kadar aracı sunucu uç birime "POP" komutunun cevabını göndermeyi bekletmektedir.
- Commands.CommandParameters.Username-Password: Eğer aracı sunucuda bir kullanıcı girişi sistemi oluşturulmuşsa uç birimler giriş yapabilmek için bu alanları doldurmaktadır.
- Commands.CommandParameters.CommandType: JTP protokolünün genişletilebilme özelliği kullanarak JTP protokolünün desteklediği komutların

dışında yeni bir komut tasarlanmak istendiğinde yeni komutun ismi bu alana girilmektedir.

- Payload: Gönderilmek istenen verinin veya mesajın girildiği alandır. Veri tipi object olması sebebiyle bu alana metin, dizi ve kompleks yapı şeklinde veri türleri JSON sözdizimiyle girilmektedir.
- Result: Uç istemciler aracı sunucuya bir komut gönderdiklerinde komutun özelliğine göre "OK" cevabı gelebilmektedir. Eğer gönderilen komutta bir sözdizimi hatası veya sistemsel bir arıza oluştuğunda vb. Hata durumlarında "Error" cevabı dönmektedir. Yapılmaya çalışılan işlemde uç birimin yetkisi yoksa "AccessDenied" ve yapılmaya çalışılan işlem zaman aşımına uğradıysa "Timeout" cevabı dönmektedir. "OK" cevabı dışında diğer hata kodları gelmesi durumunda hatanın detayları ile ilgili bilgiler "Payload" alanında yer almaktadır.

Modeldeki Topic alanına kanal girişi yapılırken uyulması gereken bir sözdizimi mevcuttur. Bu sözdizimine göre kanal bilgisindeki konu seviyelerinin arasına "/" karakteri eklenmelidir. Ayrıca kanal bilgisindeki herhangi bir konu seviyesi yerine "+" karakteri girilerek ilgili konu seviyesindeki tüm konu seviyelerini temsil etmektedir. Kanal bilgisinin sonuna "#" karakteri eklendiğinde ilgili konu seviyesi ve sonraki tüm seviyeleri temsil etmektedir. Aracı sunucu üzerinde sistemsel işlem yapmak veya aracı sunucudan istatiksel veriler almak için kanal bilgisinin ilk seviyesindeki konu "$" karakteri ile başlamaktadır [36].

3.3.1. Yayımcı-abone tipi iletişim desteği

Aracı sunucuya sahip iletişim protokollerinin en temel iletişim tipidir. Bu nedenle JTP modeli varsayılan olarak "Publish" komutunu oluşturacak şekilde tasarlanmıştır. Aşağıda "TestTopic/Topic1" adlı kanala "Test Message" metin verisini yayınlayan protokol örneği verilmiştir.

```
{"Topics":["TestTopic/Topic1"],"Payload":"Test Message"}\r\n
```

Protokol örneğinde verilen alanların dışındaki diğer alanlar varsayılan olduğundan girilmemiştir. Eğer tüm alanlar girilmek istendiğinde aşağıdaki protokol örneği kullanılmalıdır ve yukarıdaki örnek ile aynı görevi görmektedir. Metnin devamında varsayılan alanlar protokol örnekleri üzerinde girilmeyecektir.

```
{"Topics":["TestTopic/Topic1"],"IsCompressed":false,"IsReset":false,"Commands":{"QoS":0,"CommandType":0,"Command
Parameters":{"IsRetain":false,"CustomParameters":null}},"Payload":"Test Message"}\r\n
```

Yayınlanan verilerin diğer uç birimlere gönderilebilmesi için diğer uç birimlerin bu gönderilen mesajın kanalına abone olmaları gerekmektedir. Abone işlemi için "Subscribe" komutu kullanılmaktadır.

```
{"Topics":["TestTopic/Topic1"],"Commands":{"CommandType":1}}\r\n
```

Belirli bir kanalın ("TestTopic/Topic1") aboneliğinden çıkmak için "Unsubscribe" komutu ile aşağıdaki protokol örneği kullanılır.

```
{"Topics":["TestTopic/Topic1"],"Commands":{"CommandType":2}}\r\n
```

3.3.2. Arayan-aranan tipi iletişim desteği

Bu iletişim tipi HTTP protokolünün çalışma yapısına benzemektedir ancak HTTP protokolünden farklı olarak ilgili uç birimlerden gerçek zamanlı olarak sorgu yapılmaktadır ve sorgu cevabı eşzamansız olarak geri gönderilmektedir. İstek sorguları "Caller" komutu ile yapılırken, isteklerin cevapları veya sonuçları "Callee" komutuyla geri gönderilmektedir. "Caller" komutu gönderilirken modeldeki "Topics" alanına sadece tek kanal girişi yapılmalıdır. Bu kanala abone olan veya ilgili konuya gönderilecek sorguları işleyip cevaplayabilme yeteneği olan uç birimlerden sadece bir tanesine veri gönderilmektedir. Kanala abone olan uç birimlerden verinin hangisine gönderilmesi gerektiğine karar vermek için Round Robin [37] vb. algoritmalar kullanılmalıdır. Böylelikle bu kanala cevap verebilen uç birim veya servisler arasında

yük dengeleme yapılmaktadır. Ayrıca sorguya cevap veren uç birim “Callee” komutunda modeldeki “Result” alanını da doldurabilmektedir. Bu sayede yapılan işlemle ilgili hata, zaman aşımı, yetki hataları gibi bildirimleri geriye döndürebilmektedir. Eğer “Result” alanı doldurulmamışsa aracı sunucu tarafından varsayılan olarak “OK” değeri girilmektedir. Sırasıyla aşağıda “Caller” ve “Callee” komutlarına ait örnekler verilmiştir.

```
{"Topics":["TestTopic/Topic1"],"Commands":{"CommandType":3},"Payload":"Test Message"}\r\n
```

```
{"Topics":["TestTopic/Topic1"],"Commands":{"CommandType":4,"CommandParameters":{"SenderSessionId":"xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"}},"Payload":"Test Message"}\r\n
```

3.3.3. Yük dengeleyicili kuyruk desteği

Kuyruğa eklenecek olan veriler “Push” komutuyla gönderilmektedir. “Push” komutunda modeldeki “Commands.QoS” değeri sadece “AtMostOnce” ve “AtLeastOnce” parametrelerini desteklemektedir. Kuyruktaki veriyi almak için “Pop” komutu kullanılmaktadır. “Pop” komutu aracı sunucuya gönderilirken modeldeki “Commands.CommandParameters.IsDeleted” alanı false olarak girilmişse kuyruğun en başındaki veri uç birime gönderilir ancak kuyruktan veri silinmez, eğer bu alan “true” olarak girilmişse yine kuyruğun en başındaki veri uç birime gönderilir ve sunucudan ilgili veri silinmektedir. Kuyruk verisi uç birime ulaştırıldıktan sonra uç birim kuyruktaki bir sonraki elemanı çekmek istediğinde tekrar aracı sunucuya “Pop” komutu göndermektedir. “Push” ve “Pop” komutu gönderilirken modeldeki “Topics” alanına sadece tek kanal girişi yapılmalıdır. “Pop” komutunda “Commands.CommandParameters.IsWaitForPop” alanı true olarak girilmişse ve kuyrukta hiç veri yoksa kuyruğa yeni veri eklenene kadar istemciye geri dönüş yapılmayıp sadece kuyruğa yeni veri eklendiğinde geri dönüş yapılmaktadır. Aynı boş kuyruk için birden fazla uç birim “Pop” komutu gönderdiyse Round Robin [37] vb. bir algoritma ile uç birimlerden sadece bir tanesine veri gönderilmektedir. Aşağıda JTP protokolüyle sırasıyla “Push” ve “Pop” komutlarına örnek verilmiştir.

```
{"Topics":["TestTopic/Topic1"],"Commands":{"CommandType":5},"Payload":"Test Message"}\r\n
```

```
{"Topics":["TestTopic/Topic1"],"Commands":{"CommandType":6}}r\n
```

3.3.4. Süre limitli veri iletimi desteği

JTP protokolünün yük dengeleyicili kuyruk desteği ile benzer bir iletim desteğidir. Farklı olarak kuyruğa eklenen verilerin belirli biz zaman değerinden sonra uç birimlerden gelen "Pop" komutuna karşılık uç birimlere gönderilmemektedir ve aracı sunucu hafızasından silinmektedir. Modeldeki "Commands.CommandParameters.ExpiryDateTime" alanına girilen zaman bilgisi UTC +00:00 zaman dilimine ait olmalıdır. Süre limitli veri iletimi için "Push" komutu örneği aşağıda verilmiştir.

```
{"Topics":["TestTopic/Topic1"],"Commands":{"CommandType":5,"CommandParameters":{"ExpiryDateTime":"2019-04-21T00:00:00.0000000Z"}},"Payload":"Test Message"}\r\n
```

3.3.5. QoS (Quality of Service) iletim denetimi desteği

QoS iletim denetimi sayesinde gönderilen komut ve verilerin aracı sunucuya veya karşı uç birim veya birimlere ulaşıp ulaşmadığı konusunda bilgi alınmaktadır. Örneğin bir "Publish" komutu gönderilirken modeldeki "Commands.QoS" alanı "AtMostOnce" olarak girilirse komutun takibi yapılmayacaktır, "AtLeastOnce" olarak girilirse "Publish" komutu aracı sunucuya ulaştığında komutu gönderen uç birime "OK" cevabı gönderilecektir, eğer bu alan "ExactlyOnce" olarak girilirse gönderilen verinin kanalını dinleyen tüm uç birimlere veri iletildikten sonra veriyi gönderen uç birime "OK" cevabı gönderilmektedir. Sırasıyla "Commands.QoS" değeri yüklenmiş "Publish" komutu ve "OK" cevabı gönderilen protokol örneği aşağıda verilmiştir.

```
{"Topics":["TestTopic/Topic1"],"Commands":{"QoS":2},"Payload":"Test Message"}\r\n
```

```
{"Topics":["TestTopic/Topic1"],"Commands":{"CommandType":0},"Result":0}\r\n
```

3.3.6. Kalp atışı desteği

Uç istemciler ile aracı sunucu arasında bağlantının hala olup olmadığı ve aracı sunucunun hala hizmet verip vermediği gibi bilgilerin alınabilmesini sağlamaktadır. Bir uç birim aracı sunucuya "Ping" komutu gönderdiğinde aracı sunucu cevaben "Pong" komutu göndermelidir. Eğer belirli bir süre içerisinde gönderilmezse bağlantının koptuğu ve aracı sunucunun artık hizmet vermediği çıkarımı yapılmaktadır. Sırasıyla "Ping" ve "Pong" komutları aşağıda verilmiştir.

```
{"Commands":{"CommandType":8}}\r\n
```

```
{"Commands":{"CommandType":9}}\r\n
```

3.3.7. Protokolün genişletilebilir olması

JTP protokolünün desteklemediği bir iletişim tipine ihtiyaç duyulduğunda veya desteklenen protokollere ek özellikler kazandırılmak istendiğinde JTP protokolü buna izin vermektedir. Yeni bir protokol oluşturmak için "Commands.CommandType" alanına "CustomCommand" değeri girilmelidir ve bu protokole ait parametre bilgileri "Commands.CommandParameters.CommandType" ve "Commands.CommandParameters.CustomParameters" alanlarına veri girişi yapılmaktadır. Yeni protokol oluşturmayı amaçlayan protokol örneği aşağıda verilmiştir.

```
{"Topics":["TestTopic/Topic1"],"Commands":{"CommandType":10,"CommandParameters":{"CommandType":"NewComma
nd","CustomParameters":"NewCommandParameters"}},"Payload":"Test Message"}\r\n
```

Ayrıca desteklenen protokollere yeni özellikler eklemek için "Commands.CommandParameters.CustomParameters" alanına gerekli parametrelerin eklenmesi gerekmektedir. Aşağıda "Publish" komutuna "NewCommandParameters" verisi eklenmiştir ancak buraya kompleks yapılar da girilebilmektedir.

{"Topics":["TestTopic/Topic1"],"Commands":{"CommandType":0,"CommandParameters":{"IsRetain":false,"CustomParameters":"NewCommandParameters"}},"Payload":"Test Message"}\r\n

3.3.8. Verilerin gerçek zamanlı sıkıştırılarak iletimi

Ağ veri trafiği iletişimin performansını etkileyen önemli unsurlardandır. Verinin uç birimden sıkıştırılarak aracı sunucuya iletilebildiği gibi aracı sunucuya normal bir veri olarak gelen verinin aracı sunucu tarafından sıkıştırılmış olarak dağıtılması da mümkündür. İçerisinde sıkıştırılmış bir veri bulunduğunu ifade eden örnek JTP protokolü aşağıda verilmiştir.

{"Topics":["TestTopic/Topic1"],"IsCompressed":true,"Payload":"VGVzdCBNZXNzYWdl"}\r\n

3.3.9. Kimlik doğrulama desteği

Aracı sunucu tarafından bir oturum özelliği barındırıyorsa aşağıda verilen JTP protokolü ile oturum açma işlemi yapılabilmektedir.

{"Commands":{"CommandType":7,"CommandParameters":{"IsRetain":false,"CustomParameters":{"Username":"deneme","Password":"password"}}}}\r\n

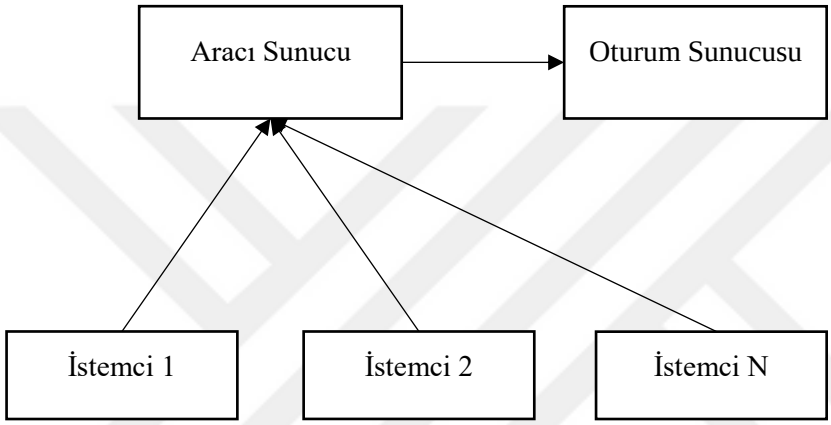
3.3.10. İzleme desteği

Aracı sunucuyla ilgili işlem yapmak ve aracı sunucu üzerindeki istatistiksel verileri almak için kanal bilgisinin ilk konu seviyesi “$” karakteri ile başlamalıdır. Aracı sunucunun ne tür istatistiksel veriler verdiği veya hangi kanal ismiyle yayın yaptığı bu çalışmanın kapsamı dışındadır. Örnek bir izleme komutu aşağıda verilmiştir.

{"Topics":["$BrokerServer/Statistics"],"Commands":{"CommandType":1}}\r\n

3.4.Sunucu Mimarisi

Tüm uç birimlerin yani istemcilerin ağ üzerinden aracı sunucusuna bağlı olmaları gerekmektedir. Eğer bir oturum mekanizması kullanılacaksa bir oturum sunucusuna aracı sunucu bağlanmaktadır. Aracı sunucu ve oturum sunucusu bu çalışmanın kapsamı dışında olup bu çalışmada JTP protokolünün çalışması için gereken ortam üzerinde durulmaktadır. Sunucu mimari şeması aşağıda verilmiştir.



Şekil 2. Aracı sunucu mimari şeması

IV. GERÇEKLEŞTİRME

Bu bölüm, önerilen JTP protokolünün gerçekleştirilebilmesinde referans oluşturmayı amaçlamaktadır. Aracı sunucu ve aracı sunucuyla iletişim kurmaya yarayan uç birim kütüphaneleri gelişen teknoloji ve sistemlerle uyumluluğu sağlamak için yazılımın gerçekleştirilmesinde farklı teknolojiler kullanılabilmektedir.

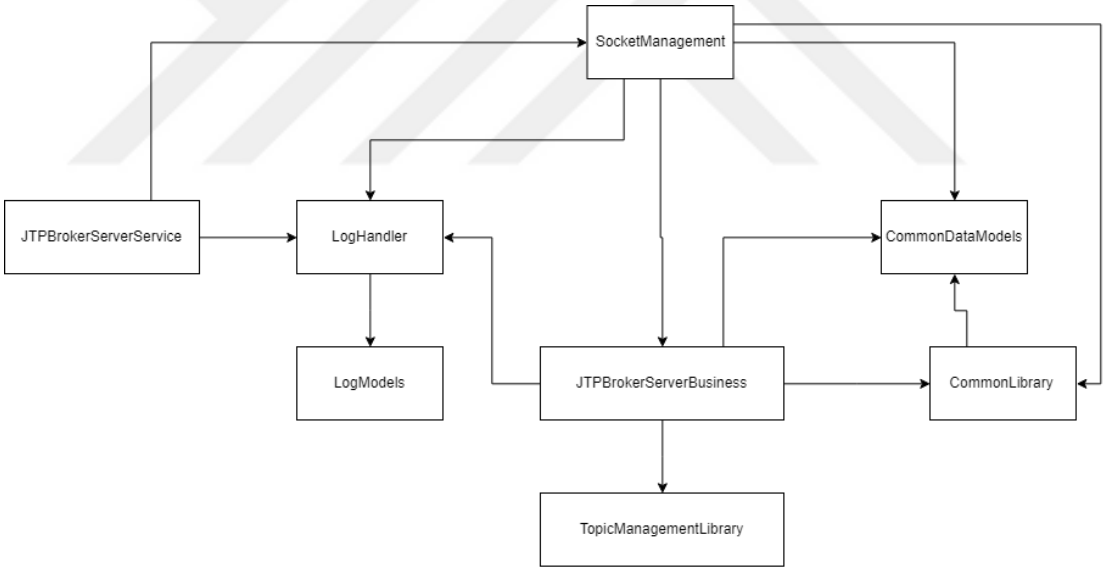
JTP protokolünü ve özelliklerini kullanarak iletişimi sağlayan aracı sunucu ve uç birim yazılımı geliştirilmiştir. Aracı sunucu başlatıldığında App.config dosyasındaki ayar bilgileri okunur ve bu ayarlara göre çalışmaya başlamaktadır. Sunucu soket tarafından bağlantı talebi kabul edilir ve ayrı bir iş parçacığı içerisinde çalışmasına olanak sağlayacak şekilde ortam oluşturulmaktadır. Bundan sonra bu uç birim ile ilgili tüm işlemler bu iş parçacığı içerisinde gerçekleştirilmektedir. Daha sonra sunucu soket yeni bağlantı talepleri için portu dinlemeye devam etmektedir. İş parçacığı içerisinde soket'ten sürekli okuma işlemi yapılır ve gelen verilerde çerçeve çözümlemesi yapılarak protokol verisinden çerçeve verisi çıkartılmaktadır. Buradan JSON formatındaki protokol paketi elde edilmektedir. JTPBrokerServerBusiness kütüphanesine gönderilen JSON formatındaki protokol paketi JTP modeline (GlobalProtocolModel.cs) dönüştürülmektedir. JTP modeli JTPBrokerServerBusiness kütüphanesinde yorumlanarak protokolün gerektirdiği işlemler gerçekleştirilmektedir. Bu işlemler yapılırken konu yönetimi TopicManagementLibrary kütüphanesi yardımıyla yapılmaktadır. Eğer gelen verinin bir uç birime gönderilmesi gerekiyorsa TopicManagementLibrary kütüphanesinden gönderilecek uç birimlerin benzersiz numaraları alınarak SocketManagement kütüphanesi üzerinden ilgili uç birimlere gönderilmektedir. Oluşan logların yönetimini LogHandler kütüphanesi üstlenmektedir.

Aracı sunucu yazılımında kullanılan kütüphanelerin kullanım şeması Şekil 3'te verilmiştir. Aracı sunucu normal olarak çalıştırıldığında komut istemi üzerinden

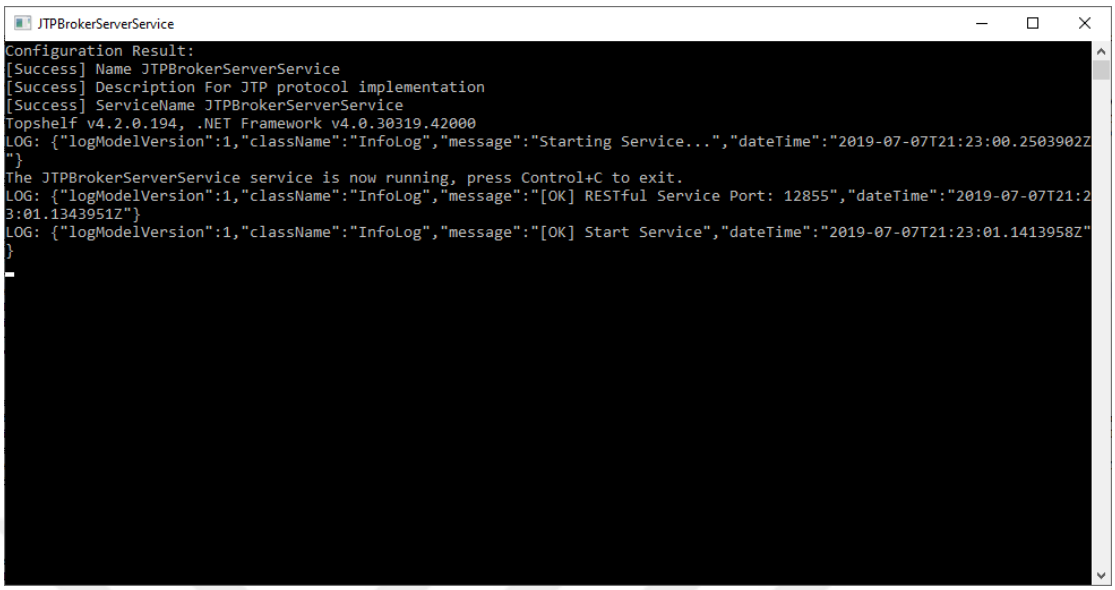
çalışmaktadır. Aracı sunucunun komut istemi üzerinden çalıştırıldığındaki ekran görüntüsü Şekil 4'te verilmiştir. Aracı sununun Windows Servisi olarak sisteme kayıt etmek ve çalıştırmak için aşağıdaki komutlar kullanılmaktadır:

```
JTPBrokerServerService.exe install –autostart
JTPBrokerServerService.exe start
```

Projenin kaynak kodları Github sitesi üzerinden Git yazılımı kullanılarak "https://github.com/serkanayaz/JTPBrokerServer.git" linkinden indirilebilmektedir. Kaynak kodlar indirildikten sonra Visual Studio programı ile açılmaktadır.



Şekil 3. Aracı sunucu yazılımında kullanılan kütüphanelerin bağımlılık şeması

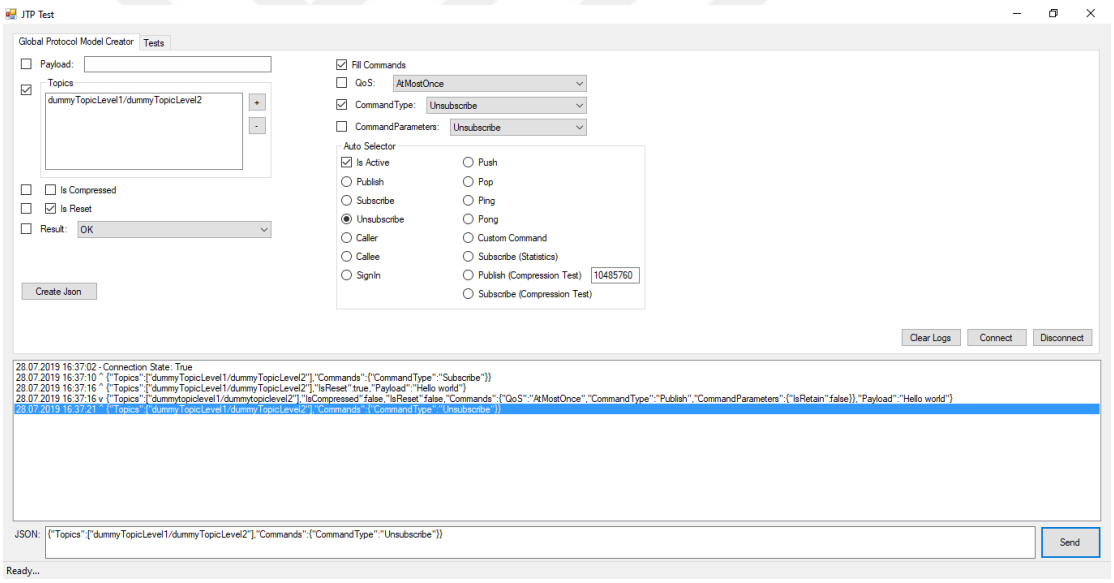


Şekil 4. Aracı sunucunun çalıştırıldığındaki ekran görüntüsü

Uç birimlerde kullanılmak üzere aracı sunucuya istemci olarak bağlanarak JTP protokolü üzerinden haberleşmeyi sağlayan bir istemci kütüphanesi ve bu kütüphaneyi kullanan bir arayüz yazılımı hazırlanmıştır. İstemci kütüphanesi .NET Framework 4.5.2 ve C# dili ile yazılmıştır.

İlgili projenin kaynak kodları JTPClientLibrary ve JTPClientTest projelerinde yer almaktadır. JTPClientLibrary kütüphanesinin kullanılabilmesi için ana uç birim yazılımından referans verilmesi gerekmektedir. Ayrıca ana uç birim projesinde aşağıdaki kodlar yardımıyla istemci kütüphanesinin kullanılması sağlanmaktadır. JTPClientTest projesi istemci kütüphanesinin testini yapmak amacıyla tasarlanan bir arayüz uygulamasıdır. Uç birim arayüzünün ekran görüntüsü Şekil 5'te verilmiştir.

```
JTPClient client = new JTPClient();
bool isConnected = this.client.Connect(/*Aracı sunucunun IP adresi*/, /*Aracı sunucunun port numarası*/,
connectionState =>
{
   //Bağlantı durumu değiştiğinde çalıştırılacak komutlar
}, receivedText =>
{
   //Aracı sunucudan veri geldiğinde çalıştırılacak komutlar
}, sendText =>
{
   //Uç birimden aracı sunucuya veri gönderildiğinde çalıştırılacak komutlar
}, ex =>
{
   //Herhangi bir hata durumunda çalıştırılacak komutlar
});
this.client.Send(/*JTP'nin JSON formatındaki verisi veya protokol parametreleri girilmelidir*/); //JTP protokolüyle veri
gönderir
this.client.Disconnect(); //Bağlantıyı kapatır
```



Şekil 5. Uç birim kütüphanesinin testini yapan arayüz uygulaması

V. PROTOKOL TESTLERİ

5.1.Karşılaştırma Kriterlerine Göre Protokolün Testi

Araştırılan protokollerin karşılaştırma kriterlerine göre protokol testleri, araştırılan protokollerin ve JTP protokolünün desteklediği yayımcı-abone tipi iletişimin yayınlama protokolü örneği üzerinden Bölüm 2.3.5'te yapılmıştır. JTP protokolünün testi için aşağıdaki JTP protokol örneği kullanılacaktır.

```
{"Topics":["konular/konu1"],"Payload":"Merhaba Dünya"}\r\n
```

Protokol karşılaştırma için incelendiğinde verinin sonunu belirtmek için 2 bayttan oluşan "\r\n" karakterleri kullanıldığından bu karakterler çerçeve verileridir. Verilerin çerçeve çözümlemesi için gelen her bayt analiz edilerek içerisinde "\r\n" karakterleri arandığından çerçeve çözümleme algoritma karmaşıklığı O(n) olarak belirlenir. Çerçeve kodlanırken de protokol verilerinin sonuna sadece "\r\n" karakterleri eklendiğinden kodlamanın algoritma karmaşıklığı O(1) olarak belirlenir. Mesaj boyutu 13 bayttan oluşmaktadır ve çerçeve dışında kalan veri protokol verisi olduğundan protokol verisinin boyutu 41 bayttan oluşmaktadır. Ancak ikinci ve sonrasında gönderilen "Publish" komutunda kanal bilgisi hafızaya alınacağından modeldeki "Topics" adlı alanın girilmesine gerek yoktur. Bundan dolayı ikinci ve sonrasında protokol boyutu 14 bayt olduğu görülmektedir. Protokol verisi JSON sözdiziminden oluşmaktadır. Dolayısıyla protokol verisi çözümlenirken her karakter kontrol edilerek protokol verileri anlamlandırılacağından algoritma karmaşıklığı O(n) olarak belirlenir. Aynı şekilde protokol kodlanırken komut ve parametrelerin başına sonuna JSON sözdizimine uygun karakterler ekleneceğinden algoritma karmaşıklığı O(1) olarak belirlenir. İnsanlar tarafından anlaşılması ve kodlanması incelendiğinde "{", """, ":", "[", "/", ",", "]", "}", "\r" ve "\n" olmak üzere 10 adet karakter tipi bulunmaktadır. Ancak ikinci ve sonraki komut gönderimlerinde modeldeki "Topics" alanı olmasına gerek olmadığından 7 adet karakter tipi bulunmaktadır. Elde edilen bu bulgularla Tablo 1'de verilen bulgularla karşılaştıran tablo aşağıda verilmiştir.

Tablo 2. JTP protokolünün diğer araştırılan protokollerle karşılaştırılması

	STOMP	**XMPP**	**RESP**	**NATS**	**JTP**
Çerçeve Boyutu	2 bayt	19 bayt	2 bayt	2 bayt	2 bayt
Çerçeve Kodlama	O(1)	O(1)	O(1)	O(1)	O(1)
Çerçeve Çözümleme	O(n)	O(n)	O(n)	O(n)	O(n)
Protokol Boyutu	60 bayt	520 bayt	22 bayt	22 bayt	41 bayt (14 bayt)
Protokol Kodlama	O(1)	O(1)	O(1)	O(1)	O(1)
Protokol Çözümleme	O(n)	O(n)	O(n)	O(n)	O(n)
İnsanlar Tarafından Kodlanabilme Değeri	6 veri tipi	9 veri tipi	4 veri tipi	3 veri tipi	10 veri tipi (7 veri tipi)

5.2.İşlevsel Özelliklere Göre Protokolün Testi

Protokoller, belirli sözdizimlerinin bir araya getirilmesiyle protokolün oluşturulma amacına uygun olarak birçok bildirim, komut ve direktif belirtimi yapılabilmektedir. Araştırılan protokollerin, protokol yapıları Bölüm 2.3'de anlatılmıştır. Araştırılan protokollerin oluşturdukları komut setleri yardımıyla destekledikleri özellikler Tablo 3'de verilmiştir ve JTP protokolünün işlevsel özellikleriyle karşılaştırılması yapılmıştır. JTP protokolünün işlevsel özellikleri Bölüm 3.2'de anlatılmıştır. STOMP protokolü, basit komutsal iletişim yapabilmek için tasarlandığından işlevsel özellik sayısı bakımından en az işlevsel özelliğe sahip olduğu görülmüştür. XMPP protokolünün genişletilebilir özelliği sayesinde birçok işlevsel özellik sonradan

eklenmiştir ve yeni eklenen protokol ekleri standartlaştırılmıştır. RESP protokolü, Redis sunucunun özelliklerini kullanmak amacıyla tasarlanmıştır. NATS protokolü, yayımcı-abone tipi iletişim işin tasarlanmıştır ancak bunun dışında Tablo 3'te belirtilen işlevsel özelliklere de sahiptir. Ayrıca araştırılan protokoller arasında en az sayıda komut sayısına sahip olduğu görülmüştür. Araştırılan protokollerin ve JTP'nin metin tabanlı protokoller olması, aracı sunucu üzerinden haberleşme ve gerçek zamanlı iletişim gibi temel ortak özellikleri Bölüm 2.1'de anlatılmıştır.

Tablo 3. JTP protokolünün işlevsellik yönünden diğer araştırılan protokollerle karşılaştırılması

İşlevsel Özellik	STOMP	XMPP	RESP	NATS	JTP
Gerçek zamanlı çok hedefli iletişim desteği (Bölüm 2.1.3)	Var	Var	Var	Var	Var (Bölüm 3.2.1)
Platform bağımsız veri iletimi	Var	Var	Var	Var	Var
Yayımcı-abone tipi iletişim desteği	Var	Var [12]	Var	Var	Var
Arayan-aranan tipi iletişim desteği	Yok	Var [28]	Yok	Yok	Var
Yük dengeleyicili kuyruk desteği	Yok	Var [29]	Var	Var	Var
Süre limitli veri iletimi desteği	Yok	Yok	Var	Yok	Var
QoS (Quality of Service) iletim denetimi desteği	Yok	Var [30]	Yok	Yok	Var (Bölüm 3.3.5)
Kalp atışı desteği	Var	Yok	Yok	Var	Var
Protokolün genişletilebilir olması	Yok	Var [31]	Yok	Yok	Var
Verilerin gerçek zamanlı sıkıştırılarak iletimi	Yok	Var [32]	Yok	Yok	Var (Tablo 5)
Kimlik doğrulama desteği	Yok	Var [33]	Var	Var	Var
İzleme desteği	Yok	Yok	Var	Var	Var

Veri sıkıştırma desteği Tablo 3'te belirtildiği gibi XMPP ve JTP protokolü desteklemektedir. XMPP protokolü sıkıştırma algoritması olarak ZLIB algoritmasını kullanmaktadır [32]. JTP protokolünde ise ZLIB algoritmasına göre sıkıştırma ve çözümleme performansının yüksek olması ve sıkıştırma seviyesinin kullanılacak ortamdaki ihtiyaca göre ayarlanabilmesi sebebiyle LZ4 sıkıştırma algoritmasının kullanılması önerilmektedir. ZLIB, LZ4, QuickLZ, Snappy, Zstandart, LZF sıkıştırma

algoritmalarının sıkıştırma oranı, sıkıştırma ve çözümleme performansları Tablo 4'te verilmiştir.

Tablo 4. Sıkıştırma algoritmalarının performans değerleri [40].

Sıkıştırma Algoritması	Sıkıştırma Oranı	Sıkıştırma Hızı	Çözümleme Hızı
LZ4 default (v1.9.0)	2.101	780 MB/s	4970 MB/s
LZO 2.09	2.108	670 MB/s	860 MB/s
QuickLZ 1.5.0	2.238	575 MB/s	780 MB/s
Snappy 1.1.4	2.091	565 MB/s	1950 MB/s
Zstandard 1.4.0 -1	2.883	515 MB/s	1380 MB/s
LZF v3.6	2.073	415 MB/s	910 MB/s
zlib deflate 1.2.11 -1	2.730	100 MB/s	415 MB/s
LZ4 HC -9 (v1.9.0)	2.721	41 MB/s	4900 MB/s
zlib deflate 1.2.11 -6	3.099	36 MB/s	445 MB/s

Tablo 5. LZ4 ve ZLIP algoritmalarının JTP aracı sunucu üzerindeki performans karşılaştırması.

Sıkıştırılmamış Veri Boyutu	LZ4		ZLIB	
	Protokol Boyutu (Bayt)	Süre (ms)	Protokol Boyutu (Bayt)	Süre (ms)
128 B	436	0,96	584	1,96
256 B	596	0,98	688	1,98
512 B	872	1,01	860	1,99
1 KB	1328	1,03	1164	2,03
1 MB	6812	48,99	27544	93,02
10 MB	56156	552,98	266940	892,95

Aracı sunucusu üzerinde LZ4 ve ZLIP sıkıştırma algoritmalarının performans değerleri ölçülmüştür ve sonuçları Tablo 5'te verilmiştir. Testler Intel Core i7 Q720 1.60 GHz, 8 GB RAM, Windows 10 64 Bit konfigürasyonlarına sahip bir bilgisayarda

yapılmıştır. Test uygulamaları olarak iletişimi sağlayan bir aracı sunucu, “Publish” komutuyla veri gönderen bir uç birim ve gönderilen verileri “Subscribe” komutuyla alan uç birim olmak üzere 3 adet uygulama kullanılmıştır. Ağ iletişimindeki gecikmeyi en aza indirebilmek için aracı sunucu ve uç birimler aynı bilgisayar üzerinde lokal ağı kullanarak haberleşmektedir. Tablo 5’teki “Sıkıştırılmamış Veri Boyutu” sütunu “Publish” komutuyla protokolün “payload” alanına tarih-saat bilgisi dışındaki girilen normal verinin (Lorem Ipsum) boyutunu ifade etmektedir. Gönderilen veri aracı sunucuya ulaştığında sıkıştırılarak konuya üye olan diğer uç birimlere aktarılmaktadır. Hedef uç birimine ulaşan verinin boyutu ve ne kadar sürede ulaştığı bilgisi ZLIP ve LZ4 için ayrı ayrı hesaplanarak uç birim üzerinde istatistikleri hesaplanmaktadır. İlgili test ortamı ve JTP protokolü üzerinde yapılan testlerin Tablo 5’te verilen performans değerleri incelendiğinde LZ4 algoritmasının ZLIP algoritmasına göre daha hızlı ve daha fazla sıkıştırma oranına sahip olduğu gösterilmiştir.

VI. SONUÇ

JTP adında JSON sözdizimi ile formatlanmış aracı sunucu üzerinden gerçek zamanlı iletişim için kullanılan yeni bir protokol önerilmiştir. Bu protokolün kurumsal uygulamalarda, IoT projelerinde, gerçek zamanlı iletişim gereksinimini karşılamak ve tüm platformlar arasında iletişim kurmak için uygun olduğu gösterilmiştir. Araştırılan protokollerden işlevsel özellikleri bakımından daha fazla özelliğe sahip bir protokoldür. Ancak insanlar tarafından okunabilme ve kodlanabilme özelliği bakımından iyi seviyede olmadığı değerlendirilmektedir. JTP protokolünün desteklemediği ancak literatürde kabul görmüş iletişim tipleri araştırılarak JTP protokolünün geliştirilmesi sağlanabilir. Ayrıca JTP protokolünün kullandığı standart model JSON yerine çalışmanın kapsamı dışında olan daha hızlı kodlama ve çözümleme performansı sunan sözdizimleri kullanılırsa daha da performanslı hale getirilebilecektir. Aracı sunucu ile uç birimler arasında TLS protokolüyle veriler kriptolanarak daha güvenli bir iletişim gerçekleştirilebilecektir.

VII. KAYNAKLAR

1. BRAY, T. (2017), *The JavaScript Object Notation (JSON) Data Interchange Format*, 23 Nisan 2019 tarihinde IETF sitesi: https://tools.ietf.org/html/rfc8259 adresinden alındı.
2. STOMP (2012), *STOMP Protocol Specification, Version 1.2*, 23 Nisan 2019 tarihinde STOMP sitesi: https://stomp.github.io/stomp-specification-1.2.html adresinden alındı.
3. XMPP, *An Overview of XMPP*, 23 Nisan 2019 tarihinde XMPP sitesi: https://xmpp.org/about/technology-overview.html adresinden alındı.
4. SAINT-ANDRE, P., CRIDLAND, D., MEIJER, R. (2019), *XEP-0001: XMPP Extension Protocols*, 23 Nisan 2019 tarihinde XMPP sitesi: https://xmpp.org/extensions/xep-0001.html adresinden alındı.
5. SHAFRANOVICH, Y. (2005), *Common Format and MIME Type for Comma-Separated Values (CSV) Files*, 23 Nisan 2019 tarihinde IETF sitesi: https://tools.ietf.org/html/rfc4180 adresinden alındı.
6. REDIS, *Redis Protocol specification*, 23 Nisan 2019 tarihinde REDIS sitesi: https://redis.io/topics/protocol adresinden alındı.
7. HUNKELER, U., TROUNG H.L., STANFORD-CLARK A. (2008), *MQTT-S – A Publish/Subscribe Protocol For Wireless Sensor Networks, COMSWARE 2008. 3rd International Conference*, Communication Systems Software and Middleware and Workshops.
8. ECMA INTERNATIONAL (2017), *The JSON Data Interchange Syntax*, 23 Nisan 2019 tarihinde Ecma-International sitesi: http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf adresinden alındı.
9. NAMIOT, D., SNEPS-SNEPPE, M. (2014), On Micro-services Architecture, *International Journal of Open Information Technologies*, 2(9), 2307-8162.
10. GROS-DUBOIS, J. (2018), *SocketCluster Protocol V1*, 23 Nisan 2019 tarihinde Github sitesi: https://github.com/SocketCluster/socketcluster/blob/master/socketcluster-protocol.md adresinden alındı.
11. SAINT-ANDRE, P. (2011), *Extensible Messaging and Presence Protocol (XMPP): Core*, 23 Nisan 2019 tarihinde XMPP sitesi: https://xmpp.org/rfcs/rfc6120.html adresinden alındı.

12. SAINT-ANDRE, P., CRIDLAND, D., MEIJER, R. (2019), *XEP-0060: Publish-Subscribe*, 23 Nisan 2019 tarihinde XMPP sitesi: https://xmpp.org/extensions/xep-0060.html adresinden alındı.
13. REDIS, *Redis Commands*, 23 Nisan 2019 tarihinde REDIS sitesi: https://redis.io/commands adresinden alındı.
14. RICART, A. (2018), *NATS Client Protocol*, 23 Nisan 2019 tarihinde NATS sitesi: https://nats.io/documentation/internals/nats-protocol adresinden alındı.
15. SOCOLOFSKY, T. (1991), *A TCP/IP Tutorial*, 23 Nisan 2019 tarihinde IETF sitesi: https://tools.ietf.org/html/rfc1180 adresinden alındı.
16. Wikipedia, *Text-based protocol*, 23 Nisan 2019 tarihinde Wikipedia sitesi: https://en.wikipedia.org/wiki/Text-based_protocol adresinden alındı.
17. Wikipedia, *Message broker*, 23 Nisan 2019 tarihinde Wikipedia sitesi: https://en.wikipedia.org/wiki/Message_broker adresinden alındı.
18. BETTATI, S.S.R. (1997), *Distributed Connection Management for Real-Time Communication over Wormhole-Routed Networks*, 23 Nisan 2019 tarihinde TEXAS A&M UNIVERSITY sitesi: http://faculty.cs.tamu.edu/bettati/Papers/icdcs1997.myrinet/html/paper.html adresinden alındı.
19. FRIEDMAN, S.L., KLIVINGTON, K.A., PETERSAN, R.W. (1986), *The Brain, Cognition and Education*, Cambridge: Academic Press Inc.
20. BIANCHI, G. (2000), Performance analysis of the IEEE 802.11 distributed coordination function, *IEEE Journal on Selected Areas in Communications*, Vol: 18.
21. SITARAMAN, M., KULCZYCKI, G., KRONE, J., OGDEN, W.F., REDDY, A.L.N. (2001), *Performance specification of software components*, New York: ACM.
22. FIELDING, R., GETTYS, J., MOGUL, J., FRYSTYK, H., MASINTER, L., LEACH, P., BERNERS-LEE, T. (1999), *Hypertext Transfer Protocol -- HTTP/1.1*, 23 Nisan 2019 tarihinde RFC-Editor sitesi: https://www.rfc-editor.org/info/rfc2616 adresinden alındı.
23. FETTE, I., MELNIKOV, A. (2011), *The WebSocket Protocol*, 23 Nisan 2019 tarihinde RFC-Editor sitesi: https://www.rfc-editor.org/info/rfc6455 adresinden alındı.
24. MOORE, K. (2003), *Simple Mail Transfer Protocol (SMTP) Service Extension for Delivery Status Notifications (DSNs)*, 23 Nisan 2019 tarihinde RFC-Editor sitesi: https://www.rfc-editor.org/info/rfc3461 adresinden alındı.

25. POSTEL, J., REYNOLDS, J. (1985), *File Transfer Protocol*, 23 Nisan 2019 tarihinde RFC-Editor sitesi: https://www.rfc-editor.org/info/rfc959 adresinden alındı.
26. ROSENBERG, J., SCHULZRINNE, H. (2002), *Session Initiation Protocol (SIP): Locating SIP Servers*, 23 Nisan 2019 tarihinde RFC-Editor sitesi: https://www.rfc-editor.org/info/rfc3263 adresinden alındı.
27. NSQ, *TCP PROTOCOL SPEC*, 23 Nisan 2019 tarihinde NSQ sitesi: https://nsq.io/clients/tcp_protocol_spec.html adresinden alındı.
28. PODGOREANU, M., CHITESCU, P., SAINT-ANDRE, P. (2009), *XEP-0251: Jingle Session Transfer*, 23 Nisan 2019 tarihinde XMPP sitesi: https://xmpp.org/extensions/xep-0251.html adresinden alındı.
29. TUCKER, M. (2018), *XEP-0142: Workgroup Queues*, 23 Nisan 2019 tarihinde XMPP sitesi: https://xmpp.org/extensions/xep-0142.html adresinden alındı.
30. WAHER, P. (2015), *XEP-xxxx: Quality of Service*, 23 Nisan 2019 tarihinde XMPP sitesi: https://xmpp.org/extensions/inbox/qos.html adresinden alındı.
31. PATERSON, I., PERLOW, J., SAINT-ANDRE, P., KARNEGES, J., TSVYASHCHENKO, A., LEBOULANGER, Y. (2017), *XEP-0136: Message Archiving*, 23 Nisan 2019 tarihinde XMPP sitesi: https://xmpp.org/extensions/xep-0136.html adresinden alındı.
32. HILDEBRAND, J., SAINT-ANDRE, P. (2009), *XEP-0138: Stream Compression*, 23 Nisan 2019 tarihinde XMPP sitesi: https://xmpp.org/extensions/xep-0138.html adresinden alındı.
33. DOBSON, R. (2018), *XEP-0101: HTTP Authentication using Jabber Tickets*, 23 Nisan 2019 tarihinde XMPP sitesi: https://xmpp.org/extensions/xep-0101.html adresinden alındı.
34. KARNEGES, J. (2011), *XEP-0291: Service Delegation*, 23 Nisan 2019 tarihinde XMPP sitesi: https://xmpp.org/extensions/xep-0291.html adresinden alındı.
35. HiveMQ Team (2015), *MQTT Essentials Part 8: Retained Messages*, 23 Nisan 2019 tarihinde HiveMQ sitesi: https://www.hivemq.com/blog/mqtt-essentials-part-8-retained-messages adresinden alındı.
36. HiveMQ Team (2015), *MQTT Essentials Part 5: MQTT Topics & Best Practices*, 23 Nisan 2019 tarihinde HiveMQ sitesi: https://www.hivemq.com/blog/mqtt-essentials-part-5-mqtt-topics-best-practices/ adresinden alındı.
37. SHREEDHAR, M., VARGHESE, G. (1996), Efficient fair queuing using deficit round-robin, *IEEE/ACM Transactions on Networking*, 3(4), 375-385.

38. EUGSTER, P.T., FELBER, P.A., GUERRAOUI, R., KERMARREC, A.M. (2003), The many faces of publish/subscribe, *ACM Computing Surveys*, 35(2), 114-131.
39. SELMER, R. (2009), *What Is Real Time Communications?*, 30 Haziran 2019 tarihinde Vonage Business sitesi: https://www.vonage.com/business/perspectives/real-time-communications adresinden alındı.
40. COLLET, Y., SKIBINSKI, P., TERRELL, N., LUO, J. (2017), *LZ4 - Extremely fast compression*, 23 Nisan 2019 tarihinde Github sitesi: https://github.com/lz4/lz4 adresinden alındı.
41. LE MOIGNE, R., PASQUIER, O., CALVEZ, J.P. (2004), *A generic RTOS model for real-time systems simulation with systemC*, IEEE, 0-7695-2085-5

EK A

JTP PROTOKOLÜNÜN .NET C# İLE YAZILMIŞ MODEL KODU

```
public class GlobalProtocolModel {
    public List<string> Topics { get; set; }
    public Nullable<bool> IsCompressed { get; set; }
    public Nullable<bool> IsReset { get; set; }
    public CommandModel Commands { get; set; }
    public object Payload { get; set; }
    public Nullable<EnumResult> Result { get; set; }

    public GlobalProtocolModel() {
        this.Topics = new List<string>();
        this.IsCompressed = false;
        this.IsReset = false;
        this.Commands = new CommandModel();
    }

    public class CommandModel {
        public Nullable<EnumQoS> QoS { get; set; }
        public Nullable<EnumCommandType> CommandType { get; set; }
        public BaseParameterModel CommandParameters { get; set; }

        public CommandModel() {
            this.QoS = EnumQoS.AtMostOnce;
            this.CommandType = EnumCommandType.Publish;
            this.CommandParameters = new PublishParameterModel();
        }
    }

    public enum EnumCommandType {
        Publish,
```

```
    Subscribe,
    Unsubscribe,
    Caller,
    Callee,
    Push,
    Pop,
    SignIn,
    Ping,
    Pong,
    CustomCommand
}

public enum EnumResult {
    OK,
    Error,
    AccessDenied,
    Timeout
}

public enum EnumQoS {
    AtMostOnce,
    AtLeastOnce,
    ExactlyOnce
}

public abstract class BaseParameterModel {
    public object CustomParameters { get; set; }
}

public class PublishParameterModel : BaseParameterModel {
    public Nullable<bool> IsRetain { get; set; }
```

```
    public PublishParameterModel() {
        this.IsRetain = false;
    }
}

public class SubscribeParameterModel : BaseParameterModel { }

public class UnsubscribeParameterModel : BaseParameterModel { }

public class CallerParameterModel : BaseParameterModel {
    public Nullable<Guid> SenderSessionId { get; set; }

    public CallerParameterModel() {
        this.SenderSessionId = null;
    }
}

public class CalleeParameterModel : BaseParameterModel {
    public Guid SenderSessionId { get; set; }

    public CalleeParameterModel() {
        this.SenderSessionId = Guid.Empty;
    }
}

public class PushParameterModel : BaseParameterModel {
    public Nullable<DateTime> ExpiryDateTime { get; set; }

    public PushParameterModel() {
        this.ExpiryDateTime = DateTime.MaxValue;
    }
}
```

```
    public class PopParameterModel : BaseParameterModel {
        public Nullable<bool> IsDeleted { get; set; }
        public Nullable<bool> IsWaitForPop { get; set; }

        public PopParameterModel() {
            this.IsDeleted = true;
            this.IsWaitForPop = false;
        }
    }

    public class SignInParameterModel : BaseParameterModel {
        public string Username { get; set; }
        public string Password { get; set; }
    }

    public class PingParameterModel : BaseParameterModel { }

    public class PongParameterModel : BaseParameterModel { }

    public class CustomCommandParameterModel : BaseParameterModel {
        public string CommandType { get; set; }
    }
}
```