Mühendislik Bilimleri ve Tasarım Dergisi
10(4), 1400 – 1416, 2022
e-ISSN: 1308-6693

Journal of Engineering Sciences and Design
DOI: 10.21923/jesd.1011915

Araştırma Makalesi Research Article

NESNELERİN İNTERNETİ AĞLARINDA TIKANIKLIK KONTROL MEKANİZMALARI İLE RPL AMAÇ FONKSİYONLARININ KARŞILIKLI PERFORMANS ANALİZİ

Rıdvan SÖYÜ [1*], Alper Kamil DEMİR[2], Selma Ayşe ÖZEL[1]

[1] Çukurova Üniversitesi, Mühendislik Fakültesi, Bilgisayar Mühendisliği Bölümü Adana/TÜRKİYE
[2] Adana Alparslan Türkeş Bilim ve Teknoloji Üniversitesi, Mühendislik Fakültesi, Bilgisayar Mühendisliği Bölümü, Adana/TÜRKİYE

Anahtar Kelimeler
Nesnelerin İnterneti,
Tıkanıklık Kontrolü,
CoAP,
RPL Amaç Fonksiyonları.

Öz

Fiziksel nesnelerin duyargalar ve yazılımlar aracılığıyla sürekli olarak haberleşmesini sağlamak için geliştirilen Nesnelerin İnterneti (IoT) teknolojileri gün geçtikçe hayatımıza daha fazla girmektedir. Ancak, IoT ağları henüz IP ağlarındaki gibi standartlara sahip değildir. Bu ağlar için mevcut protokollerin ne kadar yeterli olduğu ve yeni protokollere ihtiyaç olup olmadığı hala araştırma konusudur. Tıkanıklık kontrolü de bu konulardan biri olup, IoT ağlarındaki sürekli ve yoğun bilgi akışından dolayı büyük öneme sahiptir. IoT ağlarında IP ağlarının aksine UDP tercih edildiğinden tıkanıklık kontrolü uygulama katmanında CoAP tarafından yapılır. Literatürde farklı CoAP tıkanıklık kontrol mekanizmaların karşılaştırıldığı performans analizleri bulunsa da CoAP tıkanıklık kontrol mekanizmaları ile yönlendirme protokolü RPL'in farklı amaç fonksiyonları arasındaki ilişkiyi ve performansa etkisini inceleyen bir çalışma bulunmamaktadır. Bu çalışma kapsamında farklı protokoller kullanılarak tasarlanan ağ yığını kombinasyonları Cooja benzetim ortamında araştırılmıştır. İstemci sayısının 3 ve 9, paket teslim oranı değerinin ise %80, %90 ve %100 olarak alındığı tüm benzetimlerden elde edilen ortalama gecikme ve işlem hacmi metrikleri incelendiğinde Objective Function 0 (OF0)'ın, Minimum Rank of Hysteresis Objective Function (MRHOF) algoritmasına göre daha iyi performans gösterdiğini ve CoCoA Strong'un en iyi performans gösteren tıkanıklık kontrolü mekanizması olduğu görülmüştür.

PERFORMANCE ANALYSIS OF CONGESTION CONTROL MECHANISMS WITH RPL OBJECTIVE FUNCTIONS IN IOT NETWORKS

Keywords
Internet of Things,
Congestion Control,
CoAP,
RPL Objective Functions.

Abstract

The Internet of Things (IoT) was developed to allow physical objects to communicate continuously via sensors and software, to take more place in our daily lives. Unlike IP networks, IoT networks do not yet have established protocols. Existing protocols' appropriateness and the need for new protocols are being researched. Congestion control is one of important research topics because of continuous and intense information flow in IoT networks. Since UDP is favored in IoT networks unlike IP networks, congestion control is handled by CoAP at the application layer. In the literature, there are performance analyses comparing different CoAP congestion control mechanisms, but no study investigating the relationship and its effects on performance between the CoAP congestion control mechanisms and the various objective functions of the RPL. In this study, all network stack combinations designed using different protocols are simulated in Cooja simulator. The average latency and throughput metrics acquired from all simulations where the number of clients is 3 to 9 and the packet delivery ratios are 80%, 90%, and 100% are investigated. CoCoA Strong was determined to be the best performing congestion control method, outperforming Objective Function 0 (OF0) and the Minimum Rank of Hysteresis Objective Function (MRHOF).

Alıntı / Cite

Soyu, R., Demir, A. K., Ozel, S. A., (2022). Nesnelerin İnterneti Ağlarında Tıkanıklık Kontrol Mekanizmaları ile RPL Amaç Fonksiyonlarının Karşılıklı Performans Analizi, Mühendislik Bilimleri ve Tasarım Dergisi, 10(4), 1400-1416.

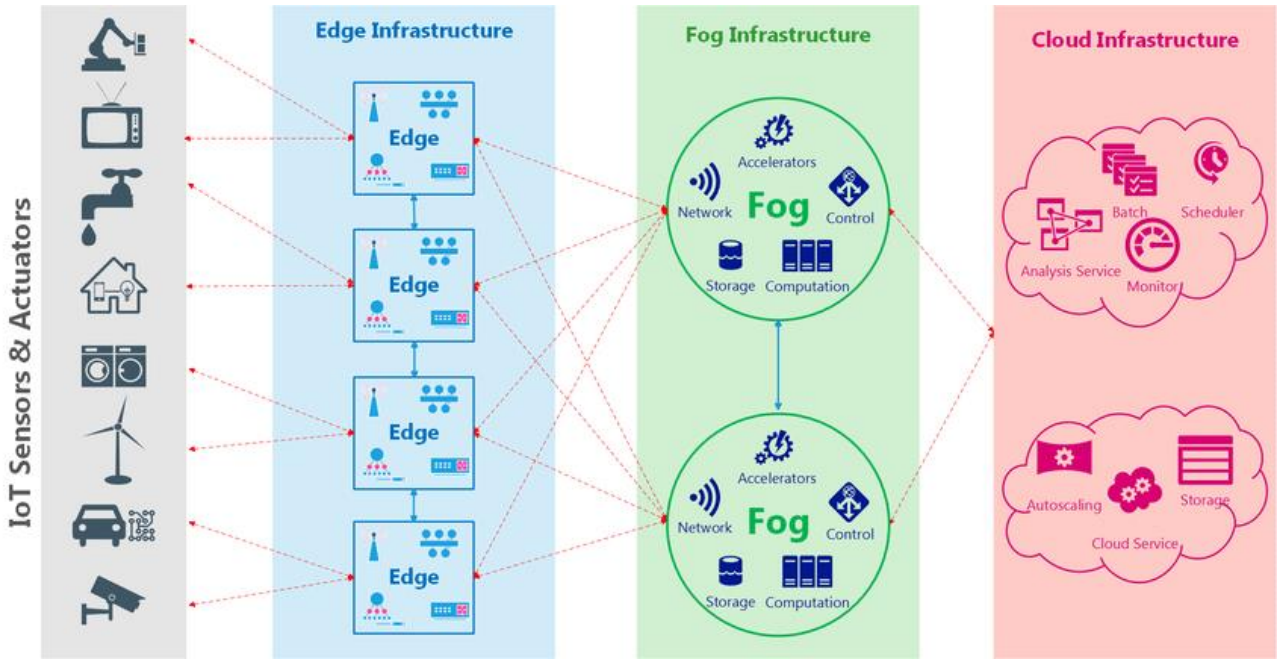
* İlgili yazar / Corresponding author: ridvan.soyu@toros.edu.tr, +90-538-248-1617

Yazar Kimliği / Author ID (ORCID Number)
R. Söyü, 0000-0001-8939-3063
A.K. Demir, 0000-0002-9256-0368
S.A. Özel, 0000-0001-9201-6349

Makale Süreci / Article Process	
Başvuru Tarihi / Submission Date	19.10.2021
Revizyon Tarihi / Revision Date	29.05.2022
Kabul Tarihi / Accepted Date	24.08.2022
Yayım Tarihi / Published Date	30.12.2022

1. Giriş (Introduction)

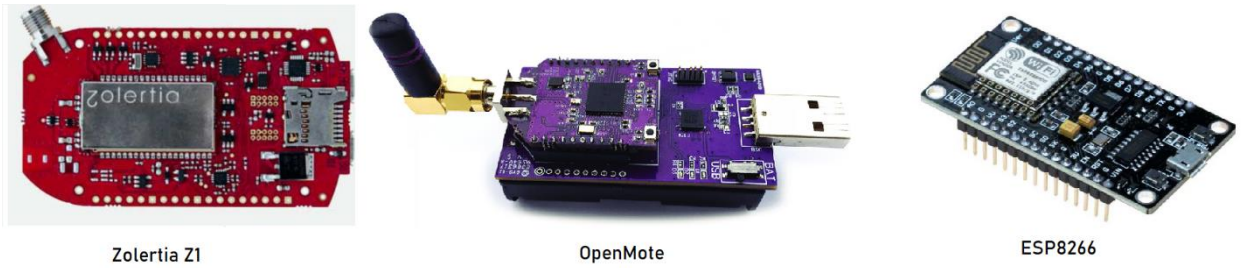
Kablosuz ağların kullanımı gün geçtikçe her türlü fiziksel nesneye ya da cihaza sirayet etmektedir. Örneğin; günümüzde akıllı evlerde bulunan sistemlerde klima ya da pencere gibi nesnelerde algılayıcılar kullanılabilmektedir. Bu algılayıcılar ile kurulan ağlar kullanıcıya evlerinin her detayı hakkında sürekli bilgi göndermektedir. Ancak bu cihazların sürekli hizmet vermesi, estetik açıdan boyutlarının yerleştirileceği konum ve çalışacağı ortama uygun tasarlanması gibi problemler bu ağlarda kullanılan donanım mimarisini ve tasarımını etkilemektedir. IoT ağları olarak nitelendirilen bu tür ağlarda birbirleriyle sürekli olarak haberleşen cihazlar kesintisiz bir güç kaynağı ve internet bağlantısına ihtiyaç duyar (Buratti vd., 2011).



Şekil 1. IoT ağlarının genel mimarisi (General architecture of IoT networks) (Cao vd., 2019)

Şekil 1 IoT ağlarının en genel mimarisini göstermekte olup standart bir kural olmasa da genellikle bu mimari 4 ana katmanda incelenir (Tiburski vd., 2019):

- Bulut (Cloud): Bulut katmanı en dış katman olup verilerin toplanıp depolanmasını, bu verilerin internet olan her yerden ulaşımının sağlanmasını ve bu verilerin güvenliğini sağlar.
- Sis (Fog): Sis katmanı verilerin uç noktalardan ağ geçitlerine taşınmasını sağlayan işlem gücü yüksek cihazlara sahip merkezi bir katmandır. Buradaki cihazlar veriyi işleyerek depolama yapar ve yerel ağa bağlı sınır cihazların yönetimini yaparlar.
- Sınır (Edge): Bu katman, ağ merkezine yakın konumda bulunan verilerin güvenliğini ve şifrelenmesini sağlayarak, darboğaz yaratılmaması ve trafiğin azaltılması için işlem yükünü üzerine alır. Genellikle bu katman içerisinde bulunan cihazlar genellikle dizüstü ve masaüstü bilgisayarlar, erişim noktaları ve iş istasyonları gibi cihazlardır. Cihaz katmanında alınan ham veriler burada işlenerek fog katmanına gönderilir.
- Cihaz (Device): En içte bulunan bu katmanda sürekli veri üretip ağa gönderen ve ağdan veri alan cihazlar bulunur. Burada bulunan cihazlara örnek olarak bir odanın sıcaklığını ölçen bir sensör, bir hastanın sağlık verilerini toplayan bir makine, ya da trafik verilerini ileten bir araba verilebilir.



Şekil 2. Nesnelerin İnterneti ağlarında sıkça kullanılan üç farklı kısıtlı cihaz (Three different constrained devices frequently used in IoT networks) ("Development tools Archives", 2017; Vilajosana v.d., 2015; İkizoğlu, 2020)

Şekil 2 IoT mimarisinde bahsedilen cihaz katmanında bulunan cihazlara örnek verilebilir. Resimde ilk sırada bulunan Zolertia Z1 (The Z1 Mote, 2018) MSP430 mikroişlemcisine sahiptir ve üzerinde CC2420 alıcısı bulunur. Bu çalışmada yapılan simülasyonlarda bu cihazın sanal bir örneği kullanılmıştır. Ortada bulunan cihaz OpenMote, IoT ağlarında çalışmak üzere özel tasarlanan bir cihaz olup 32KB RAM ile 512KB hafıza kapasitesine sahiptir ve üzerine işletim sistemi gömülebilir (Vilajosana v.d., 2015). Bu cihazların yanı sıra ESP8266 gibi üzerinde IoT modüllerinin ve dahili TCP/IP yığını barındıran çipler de IoT uygulamaları için sıklıkla kullanılır. 32-bit RISC mimarili işlemciye sahip bu cihaz dahili sıcaklık sensörü, WiFi ve 2.4 GHz kablosuz anten gibi modüller barındırmaktadır (Espressif, 2020).

Sürekli açık olan bu cihazlar birim zamanda olabilecek en düşük enerji sarfiyatı ile çalışmaları gerektiğinden kaynaklarını olabildiğince cimri tüketecek şekilde tasarlanmışlardır. IoT ağlarının yapıtaşları olan bu cihazlar, literatürde kısıtlı cihazlar olarak adlandırılırlar (RFC7228, 2014). Kısıtlı cihazlar bellek, bant genişliği, boyut ve enerji tüketimi açısından günümüzde kullanılan birçok cihaza göre oldukça küçük kaynaklara sahiptir.

IoT ağlarında kullanılan cihazlar günlük hayatta insanların kişisel amaçları için kullandıkları bilgisayarlar ya da akıllı telefonlar gibi olmadığından, kullanılan protokollerin de bu cihazların doğru haberleşebilmesi adına özel olarak tasarlanmaları gerekir. Örneğin günümüzde kullandığımız ağlarda HTTP (Hyper-Text Transfer Protocol) protokolü, HTML (Hyper-Text Markup Language) dokümanlarının, resimlerinin ve videolarının kaynaklar arasında transfer edilmesini sağlar. Ancak HTTP protokolü günlük hayatta kullandığımız ağ ve cihazlara göre tasarlanmış olup, hafıza ve enerji gibi endişelere yönelik tasarlanmadığından kısıtlı cihazlar üzerinde kullanılması her ne kadar mümkün olsa da ağın gereksinimleri dikkate alındığında etkili olduğu söylenemez. Bu yüzden uygulama katmanı için IoT ağlarında IETF (Internet Engineering Task Force) tarafından standart haline getirilen CoAP protokolü kullanılır. CoAP ile cihazlar aynı ya da farklı ağlar arasında birbirleri ile iletişim kurabilir ve HTTP protokolünde olduğu gibi bilgi alışverişi yapabilirler. Ayrıca CoAP, HTTP ile birlikte de çalışabilir.

IoT ağlarındaki düğümler sürekli olarak birbirleri ile çoklu gönderim (broadcast) yaparak haberleşirler. Birden fazla cihazın aynı anda aynı kaynağı kullanmak istediği durumlarda kaynak kapasitesini aşabilir ve tıkanıklık meydana gelebilir. Ayrıca trafik patlaması (traffic burst) gibi tüm algılayıcılar aynı anda mesaj göndermeye çalışacağı özel durumlarda (örneğin doğal afetler) trafikte çok kısa bir sürede artış ve beraberinde de tıkanıklık meydana gelebilir.

Geleneksel ağlarda tıkanıklık kontrolü TCP tarafından yapılır ancak CoAP, UDP üzerinde çalıştığından ve UDP'de tıkanıklık kontrolü olmadığından, CoAP'ın tıkanıklık kontrolünü de ele alması gerekir (Betzler vd., 2016). IoT ağlarındaki cihazların sürekli uyku moduna geçmesi, farklı iletişim örüntülerine sahip olması ve düşük gecikme süresine ihtiyaç duyulması bu ağlarda TCP kullanımını zorlaştıran nedenlerden birkaçıdır.

Cihazların kendi etrafındaki cihazları keşfetmesi ve yönlendirme yapabilmesi IoT ağlarında RPL (Routing Protocol for Low-Power and Lossy Networks) sayesinde gerçekleşir (RFC6550, 2012). RPL en hızlı ve etkili şekilde yönlendirme yapabilmek için uzaklık vektörleri kullanır ve oluşturduğu ağaç topolojisi ile de en düşük maliyetli yönlendirme rotasını bulmaya çalışır. Amaç fonksiyonları ise RPL'in tüm bu işlemleri nasıl yapacağını ve rotaları hangi metrikler kullanarak optimize edeceğini belirleyen yöntemlerdir (RFC6552, 2012). Bu amaç fonksiyonlarından biri OF0 (Objective Function 0) olup farklı metrikler kullanarak belirlediği rank sistemi sayesinde düğümlerin değerini sürekli olarak hesaplar ve günceller. MRHOF (Minimum Rank of Hysteresis Objective Function) ise OF0'a göre seçmeli metrik sistemi kullanan bir amaç fonksiyonudur. Örneğin metrik olarak gecikme seçilirse MRHOF en az gecikmeye sahip rotayı bulmaya çalışır (RFC6719, 2012).

Şimdiye kadar CoAP için tekrar iletim zaman aşımı (RTO - Retransmission Timeout) hesaplama yöntemleri kullanılarak farklı tıkanıklık kontrol mekanizmaları geliştirilmiştir. Kaynak araştırması bölümünde de değinildiği

üzere daha önceki performans analizi çalışmalarında genellikle farklı metrik ve ölçümlerle bu algoritmaların karşılaştırmalı analizlerinin yapıldığı görülmektedir.

Bu çalışmada ise, farklı CoAP algoritmalarının, RPL amaç fonksiyonları MRHOF ve OF0 ile bir korelasyona sahip olup olmadıkları ve performansı etkileyip etkilemedikleri olası tüm senaryolar dikkate alınarak yapılan benzetimlerle incelenmiştir. Benzetim senaryoları, her bir katman için deney sonuçlarını etkileyebilecek protokollerin kombinasyonları ile oluşturulmuştur. Benzetim sonuçlarının performansını incelemek için de ağ performans ölçümlerinde sıklıkla kullanılan işlem hacmi ve ortalama gecikme metrikleri kullanılmıştır.

2. Kaynak Araştırması (Literature Survey)

Tıkanıklık kontrol mekanizmaları ile amaç fonksiyonlarının performansa etkisi daha önceki çalışmalarda birlikte ele alınmamış olsa da iki konunun ayrı ayrı performans analizleri bazı çalışmalarda mevcuttur. Bu çalışmada hazırlanan test senaryoları ve kullanılan performans metrikleri de literatürde yer alan çalışmaların incelenmesi ile belirlenmiştir.

Qasem vd. (2015) yaptıkları çalışmada ağların yoğunluğunu rastgele ve ızgara topolojileri kullanarak dikkate almış ve paket alma oranı, güç tüketimi gibi parametreler kullanarak az yoğunluklu ağlarda OF0'ın, çok yoğunluklu ağlarda ise MRHOF'un daha iyi çalıştığını göstermişlerdir.

Pradeska vd. (2016) amaç fonksiyonlarının performanslarını ağ yakınsama zamanı, enerji tüketimi, paket teslim oranı (PDR – Packet Delivery Ratio) ve gecikme gibi parametreler kullanarak karşılaştırdıkları çalışmalarında MRHOF'un ağ kalitesi açısından daha iyi olduğunu ve güvenilir ağlar için kullanıma daha uygun olduğunu, OF0'ın ise enerji tüketimi konusunda ve ağın kalitesinden ziyade hızlı aktarım gerektiren ağların yapısına daha uygun olduğunu göstermişlerdir. Benzer bir çalışma yapan Lamaazi vd. (2017) ise parametre olarak kayıp paketler, enerji, beklenen iletim (ETX – Expected Transmission) ve atlama (hop) sayısını aldıkları çalışmalarında konuyu farklı bir yönden ele almış ve düğümler hareketli iken OF0, düğümler hareketsiz iken MRHOF'un daha etkili olduğu, farklı düğüm dağılımlarında ise iki amaç fonksiyonunun da benzer sonuçlar ürettiği sonucuna varmışlardır.

Abuein vd. (2016) orta yoğunluklu ağlar olarak bahsettikleri topolojilerde OF0 ve MRHOF amaç fonksiyonlarını PDR, güç tüketimi ve paket alma oranı (RX - Packet Reception Ratio) gibi metrikler kullanarak ızgara (grid) ve rastgele (random) topolojilerde incelemişlerdir. Rastgele topolojilerde MRHOF'un, ızgara topolojilerde ise OF0'ın daha çok enerji tükettiğini, ancak düğüm sayısı ve paket alma oranlarının farklı olması durumunda da farklı sonuçlar alınabileceğini göstermişlerdir.

Cooja benzetim ortamı kullanarak yaptıkları çalışmada Mardini vd. (2017), OF0 ve MRHOF amaç fonksiyonlarının performans analizini gelen paket sayısı, kayıp paket sayısı, enerji tüketimi ve görev döngüsü (duty cycle) parametrelerinin ortalamalarını baz alarak yapmıştır. Farklı topolojiler (static-grid ve mobile-random) kullanarak yaptıkları çalışmalarının sonucunda enerji tüketimi ele alındığında OF0'ın MRHOF'a göre daha verimli sonuçlar ortaya çıkardığını bulmuşlardır.

Onwuegbuzie vd. (2020) tarafından yapılan bir çalışmada Contiki işletim sistemi için yazılan bir RPL mekanizması olan ContikiRPL kullanılarak MRHOF ve OF0 amaç fonksiyonlarının performansları incelenmiştir. Bu amaç fonksiyonlarının davranış ve performanslarının düğüm sayısına göre nasıl değiştiğini ölçmek amacıyla sayıları 10 ile 100 arasında değişen düğümler kullanmışlardır. Çalışmada performans değerlendirme metrikleri olarak paket iletim oranı, gecikme ve güç tüketimi ölçümleri yapılmış ve bu değerlerin MRHOF kullanıldığı zaman OF0'a göre daha iyi sonuçlar verdiği bulunmuştur.

Arabellek taşması durumunda gerçekleşen tıkanıklığı önlemek amacıyla Al-Kashoash vd. (2016) yaptıkları çalışmada Buffer Occupancy (Arabellek Sahipliği) adlı yeni bir RPL yönlendirme metriği geliştirmişlerdir. Bunun yanı sıra Congestion-Aware Objective Function adlı yeni bir amaç fonksiyonu geliştiren araştırmacılar, geliştirilen bu fonksiyonu ETX-OF, ENERGY-OF ve OF0 amaç fonksiyonları ile karşılaştırmıştır. Sonuç olarak bu fonksiyonun tıkanıklık problemini kayıp paket, işlem hacmi, paket iletim oranı ve enerji tüketimi baz alındığında %37.4 iyileştirdiğini görmüşlerdir.

Yeni bir RPL yönlendirme metriğinin geliştirildiği Xiao vd. (2014) tarafından yapılan bir başka çalışmada ise PER-HOP-ETX metriği önerilmiştir. Bir düğümden başlayarak ETX değerini toplayarak giden ETX metriklerinin aksine, ETX değerini bir yoldan düğümlerden köke ulaştıran bu metrik MRHOF amaç fonksiyonu optimize edilerek

geliştirilmiştir. Ağ gecikmesi, paket iletim oranı ve enerji karşılaştırıldığında ise büyük ölçekli ağlarda OF0 ve ETX-OF gibi diğer metriklere göre daha iyi çalıştığı gözlenmiştir.

Sonia ve Virani (2021) tarafından yapılan çalışmada bulanık mantık teknikleri ile hop sayısı, bir düğümün artık enerjisi ve ETX kullanılarak yeni bir amaç fonksiyonu olan RPL-FZ tasarlanmıştır. Contiki işletim sistemi üzerinde yapılan performans değerlendirmesinden sonra RPL-FZ'nin MRHOF ve OF0'a göre özellikle yüksek yoğunluklu ağlarda daha performans verdiği sonucuna ulaşılmıştır. Farklı senaryolar ve değişken ağ yoğunluğu ile yapılan testlerde paket iletim oranı, enerji tüketimi, gecikme ve trafik yükü parametre alınmıştır.

Betzler vd. tıkanıklık kontrol mekanizmalarının performansına dair iki farklı çalışma yapmışlardır. Bunların ilkinde (2016) Default CoAP CC (ön tanımlı CoAP tıkanıklık kontrol mekanizması), farklı algoritma takviyeleri kullanılarak CoCoA adlı bir tıkanıklık kontrol mekanizmasına evirilmiş ve Default CoAP CC'nin performansı iyileştirilmeye çalışılmıştır. Bu mekanizma ile 5 farklı mekanizmanın aynı koşullarda performansını karşılaştırarak CoCoA'nın daha üstün olduğu sonucuna varmışlardır. Diğer çalışmada (2015) ise CoCoA+ adında yeni bir algoritma geliştirerek önceki algoritma CoCoA'yı geliştirmeyi esas almışlardır. Yaptıkları çalışma sonucunda ise CoCoA+ algoritmasının Default CoAP CC ve CoCoA'dan daha iyi olduğu sonucuna varmışlardır.

Demir ve Abut (2018), Default CoAP CC ve CoCoA algoritmalarının performansını farklı ızgara topolojilerinde sürekli mesajlar kullanarak incelemişlerdir. Çalışmaları göstermiştir ki, önceki çalışmaların aksine işlem hacmi metriği baz alındığında CoCoA her zaman Default CoAP CC den üstün değildir.

3. Materyal ve Yöntem (Material and Method)

3.1. Materyal (Material)

Bu çalışmada yazılım, donanım ve ağ kapsamında birçok protokol ve programdan yararlanılmıştır. Yazılımlar benzetim ortamını hazırlamak ve kurmak için, donanımlar benzetimi gerçekleştirebilmek için kullanılmıştır. Ağ katmanında ise farklı ağ yığınları farklı protokoller ile kurulmuş, performans metrikleri ile de benzetim sonuçları elde edilip analizi yapılmıştır.

3.1.1. Yazılım Ortamları (Software Environments)

Benzetimleri gerçekleştirmek için kullanılacak cihazlara gömülmek üzere IoT ağları için özel olarak geliştirilmiş ve açık kaynak kodlu Contiki-NG ("Contiki-NG: The OS for Next Generation IoT Devices", 2020) işletim sistemi kullanılmıştır. İçerisinde ağ benzetimleri yapma imkânı sunan Cooja ("Cooja Simulator", 2016) ağ benzetim ortamını barındırması, farklı mikroişlemci mimarilerine sahip sanal cihazları (ARM Cortex-M3/M4 ve MSP430 gibi) hazır olarak bulundurması, birçok ağ protokolü ve sürücüsünü desteklemesi, Contiki-NG işletim sisteminin tercih edilmesinde en büyük etken olmuştur.

CoAP istemcileri için; Java programlama dili kullanılarak geliştirilmiş ve içerisinde birden fazla hazır tıkanıklık kontrol mekanizması bulunduran Californium ("Eclipse Californium", 2020) uygulaması kullanılmıştır. CoAP sunucuları ise Contiki-NG işletim sistemi ile gelen örnek uygulamalardan faydalanılarak oluşturulmuştur. Bu sunucular ise Erbium ("A Quick Introduction to the Erbium (Er) REST Engine", 2016) adında C programlama dili ile geliştirilen tıkanıklık kontrol mekanizmasına sahiptir. Erbium ve Californium tıkanıklık kontrol mekanizmaları geliştirilirken RFC 7252 baz alınmıştır.

3.1.2. Donanım Ortamları (Hardware Environments)

Cooja ağ benzetim ortamı, üzerinde gerçeğe oldukça yakın şekilde hazırlanmış sanal cihazlar barındırır. Benzetimler için fiziksel donanımlar kullanılmamış olsa da Cooja'nın desteklediği bu sanal cihazlardan biri olan Zolertia Z1 Mote ("The Z1 mote", 2018) kullanılmıştır. Z1 platformu genel amaçlı bir platform olup kablosuz algılayıcılar ağları için özel olarak tasarlanmıştır. Genel özellikleri aşağıda verildiği gibidir:

- İkinci nesil MSP430F2617 düşük güç mikroçip ve CC2420 alıcısı
- 16-bit RISC mimarisine sahip 16 MHz CPU
- 8KB RAM bellek
- 92KB Flash bellek (depolama alanı)
- 250Kbps veri aktarım oranı için 2.4GHz çekim gücü

3.1.3. Ağ Yığını (Network Stack)

Her ne kadar günümüzde TCP/IP ağ yığını her alanda kullanılıp yeterli olsa da IoT ağlarının gereksinimleri ve çalışma şekli farklı olduğundan geleneksel ağ yığınlarına göre farklı bir yapı kullanılmalıdır. Bundan ötürü IoT ağlarında nasıl bir ağ yapısı olması gerektiğine dair bilgi vermek faydalı olacaktır.

3.1.3.1. Uygulama Katmanı (Application Layer)

IoT ağlarında uygulama katmanı tarafından servisler ve uygulamalar son kullanıcının kullanabileceği şekilde sağlanır. Bu katman IoT cihazları ile ağ arasındaki iletişimi sağlamak amacıyla genelde internet tarayıcıları tarafından kullanılabilen bir ara yüze de sahiptir. Ancak her ne kadar bu iletişimi HTTP ile yapmak mümkün olsa da IoT ağının kısıtlı ortamına (enerji, depolama alanı vs.) uygun olmayacağından sadece kısıtlı cihazlarda kullanılmak üzere HTTP'ye oldukça benzer RESTful mimarisi ile UDP üzerinde çalışan CoAP protokolü geliştirilmiştir. IoT ağlarında TCP kullanılmadığından tıkanıklık kontrol görevini CoAP protokolü yerine getirir. Bu yüzden de farklı RTO hesaplamalarının baz alındığı birden fazla CoAP protokolü bulunmaktadır. Ayrıca IoT ağ yığınında TCP üzerinde çalışan ve IBM tarafından geliştirilmiş MQTT (Message Queuing Telemetry Transport) protokolü de bulunmaktadır (Locke, 2010).

3.1.3.2. İletim Katmanı (Transport Layer)

İletim katmanında geleneksel ağlarda TCP ve UDP protokolleri kullanılır. IoT ağlarında her ne kadar TCP ile çalışmak mümkün olsa da ağın gereksinimi ve çalışma şekli dikkate alındığında UDP kullanmak daha faydalı olacaktır. Çünkü TCP düşük enerji gerektiren ağlarda iletim için uygun değildir ve başlık değeri oldukça uzundur. UDP de ise daha kısa olup bağlantısız olması nedeniyle TCP'ye göre daha az kaynak harcar ve TCP'den daha hızlı çalışır (RFC768, 1980).

3.1.3.3. Ağ Katmanı (Network Layer)

IoT ağlarında ağ katmanında yönlendirmeyi yapan protokol olan RPL anahtar rol oynamaktadır. Bu protokol bir ağ topolojisinde kullanılan tüm cihazlara ait adres ve konum verisini toplayarak ağın adeta bir haritasını çıkarır. Uzaklık vektörleri kullanarak cihazlar arası mesafeyi ağa getireceği yük bakımından hesaplar. Bu sayede veri aktarımının yapılacağı en uygun ve hızlı yolu belirlemeye çalışır. Uzaklık vektörlerinin yanı sıra RPL içerisinde barındırdığı amaç fonksiyonlarını kullanarak belirli bazı metrikler ile iki cihaz arasındaki transferin maliyetini belirler. Bu amaç fonksiyonlarından en çok bilinenleri OF0 (RFC6552, 2012) ve MRHOF'tur (RFC6719, 2012). Bunların yanı sıra ağ katmanında IoT ağlarında kullanılacak yoğun cihaz sayısı da düşünüldüğünde IPv4 adres sayısının yetmeyeceğinden dolayı IoT ağlarında tercih edilen ve daha fazla sayıda cihaz adresleme yeteneğine sahip IPv6 (Rayes ve Salam, 2017) kullanılmıştır.

3.1.3.4. Uyarlama Katmanı (Adaptation Layer)

IoT ağları için IPv4 yerine ölçeklenebilirlik ve kararlılık açısından daha uygun olan IPv6 tercih edilir. IPv6 1280 bayt MTU (Maximum Transmission Unit – Maksimum İletim Birimi) kullanır. Ancak IEEE 802.15.4 bağlantıları 127 bayt MTU kullandığından IPv6 paketlerinin de 127 baytlık çerçeveler halinde uygun bir şekilde parçalanması ve yeniden birleştirilmesi için ekstra bir katman bulunur. Uyarlama katmanı adı verilen bu katmanda bulunan 6LoWPAN (IPv6 over Low-Power Wireless Personal Area Networks) protokolü verilerin parçalanmasını ve başlıkların sıkıştırılmasını sağlayarak cihazların daha az enerji ile çalışmasını sağlayarak dışardaki farklı ağlarla da haberleşmesini sağlar. Bu yüzden IoT ağlarında bir sınır yönlendirici kullanılır ve ağ bu yönlendirici vasıtasıyla dışarıya açılır (Vasseur ve Dunkels, 2010).

3.1.3.5. Ağ Erişim Katmanı (MAC Layer)

MAC (Medium Access Control – Ortam Erişim Yönetimi) katmanı ağ yığınında en önemli görevlerden biri olan paket çarpışmalarını tespit etmeye ve önlemeye sağlayan protokoller bulundurur. Bu katmanda bulunan protokoller bir çarpışma tespit ettiklerinde farklı yeniden gönderim mekanizmaları kullanarak paket kayıplarının telafi edilmesini sağlarlar (Halcu vd., 2016). Contiki işletim sistemi üzerinde farklı MAC protokolleri (TSCH, NullMAC) bulunsa da NullMAC protokolü ağ üzerinde paket çarpışması yokmuş gibi tasarlandığından gerçek senaryoların test edilmesi açısından uygun değildir ve çalışma sonuçlarının yorumlanması konusunda olumsuz etkilere neden olacaktır. Bu nedenle bu çalışmada sadece SMA (Carrier-Sense Medium Access – Taşıyıcı Dinleyen Çoklu Erişim) kullanılmıştır.

3.1.3.6. RDC Katmanı (RDC Layer)

Geleneksel ağların aksine IoT ağlarında kullanılan cihazlar sürekli açık (always-on devices) olduklarından radyo aktivitesini düzenlemek ve harcanan enerjiyi minimum seviyeye getirip cihazların az enerji ile uzun süreler çalışabilmesi için ağ yığınında bu görevi üstlenen farklı protokollerin bulunduğu RDC (Radio Duty Cycle – Radyo Görev Döngüsü) katmanı yer alır. Cihazların enerji tasarrufu yapması için kullanılabilecek en basit yöntem cihazların iş yapmadığı süre içerisinde kapalı kalmasını sağlayıp, iş yapacağını anlayacağı esnada ise kendini açmasıdır. Bunun için de kullanılan RDC protokolü ortamı periyodik olarak dinleyerek aktivite tespit ettiğinde cihazı uyandırmalıdır ("MAC Protocols in ContikiOS",2014). Bu katmanda kullanılan ContikiMAC, NullRDC, X-MAC, CX-MAC gibi birden fazla protokol bulunsa da bu çalışmanın gereksinimine en uygun olan ContikiMAC ve NullRDC seçilmiştir. ContikiMAC protokolü cihazların çalışma süresinin %99'u kadar bir süre boyunca kapalı kalmasını sağlayarak oldukça iyi bir enerji tasarrufu vaat eder (Dunkels, 2011). NullRDC kullanıldığında ise radyo sürekli olarak açık konumda bulunur.

3.1.3.7. Fiziksel Katman (Physical Layer)

IoT ağ yığınında radyo katmanı olarak da adlandırılan fiziksel katman; paketlerin bit bazında yorumlanarak elektrik sinyali haline dönüştürülüp iletim ortamına aktarımından veya bu ortamdan gelen elektrik sinyallerinin yorumlanmasından sorumludur. Bu katmanda standart olarak kabul edilen IEEE 802.15.4 protokolü kullanılır ("IEEE 802.15 WPAN™ Task Group 4 (TG4)", t.y.). Bu protokol fiziksel katmanın ve veri bağlantı katmanının standartlarını ve kurallarını belirler. IoT ağlarının kısıtlı ortamlarda çalıştığı dikkate alınarak bu protokol kısa mesafe ve düşük enerjili iletişim için tasarlanmıştır. Bu yüzden paket boyutu 127 bayt, bant genişliği de 250 kbps ile sınırlandırılmıştır.

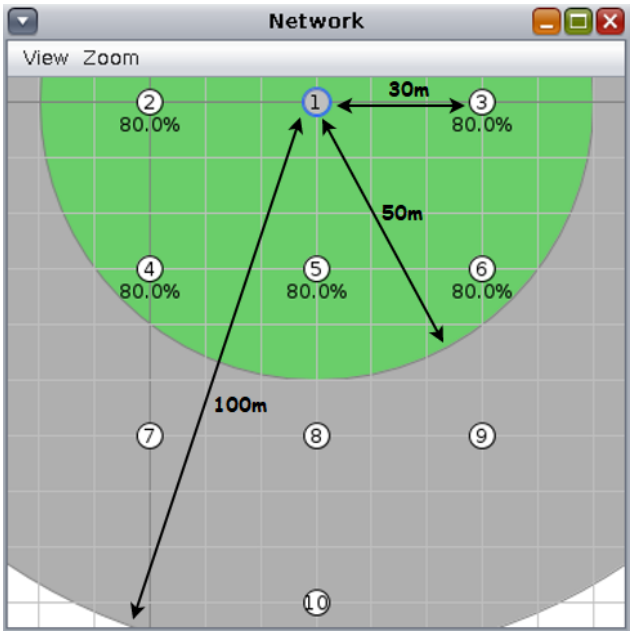
3.2. Yöntem (Method)

3.2.1. Performans Metrikleri (Performance Metrics)

Bu çalışmada işlem hacmi ve ortalama gecikme olmak üzere en yaygın kullanılan iki performans metriği ile analiz yapılmıştır. Birim zamandaki başarılı transferlerin sayısı dikkate alınarak her bir testin sonuçlarından ortalama işlem hacmi elde edilmiştir. İstemcinin hiç mesaj alamadığı durumlar ayıklanarak hesaplamalarda göz ardı edilmiştir. Ayrıca gecikme bazlı ölçümler bir ağın performansını test etmek için oldukça etkili bir yöntem olduğundan, bir paketin kaynaktan hedefe ulaşması için gereken süre, yani ortalama gecikme, ikinci metrik olarak kullanılmıştır.

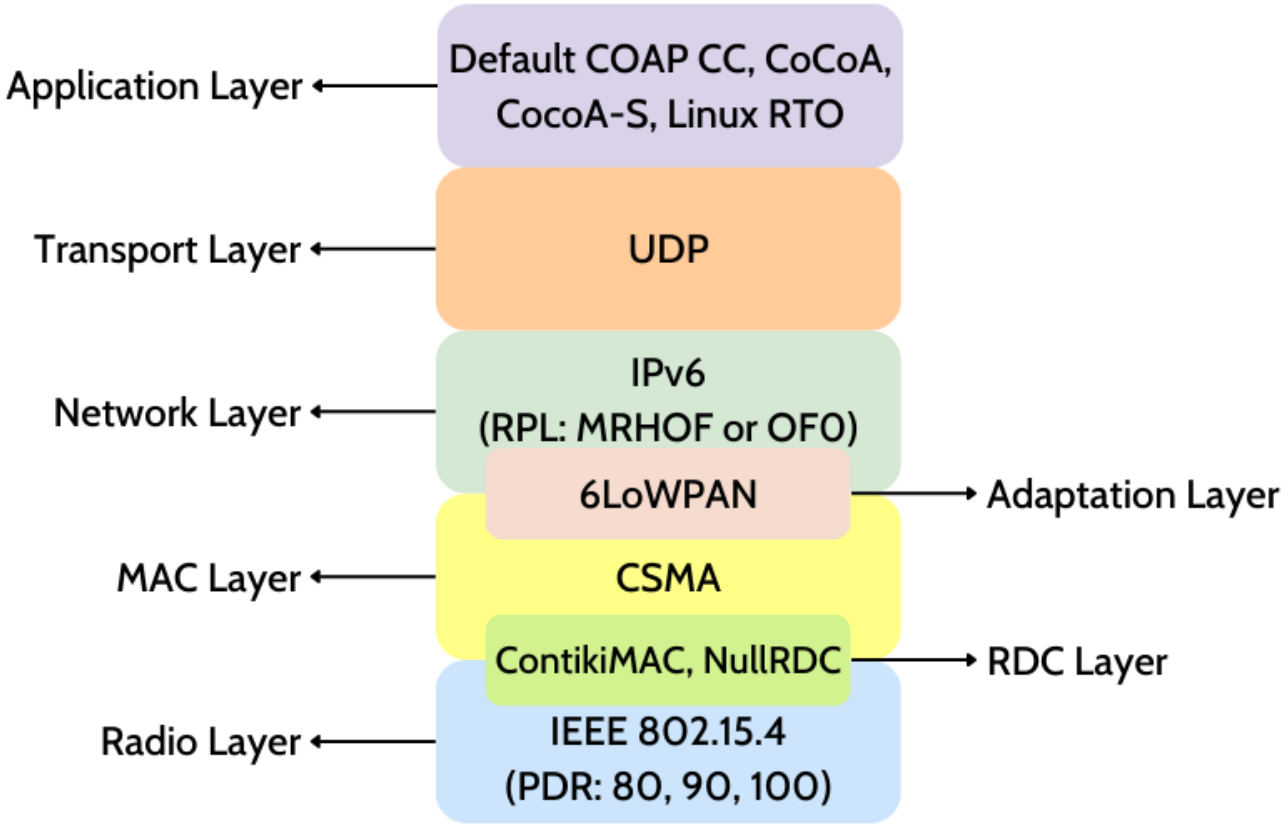
3.2.2. Benzetim Ortamı (Simulation Environment)

Deneyler hazırlanırken olabildiğince fazla kombinasyon kullanılarak ağ performansının ölçülmesinde tüm etmenlerin ne derece etki ettiğini analiz etmek esas alınmıştır. Contiki-NG içerisinde bulunan ağ benzetim ortamı Cooja üzerinde 1 tane sınır yönlendiricisi (border router) ve 9 adet CoAP serverdan oluşan bir ağ topolojisi hazırlanmış, farklı trafik yoğunluğunda tıkanıklığı gözlemlemek için de istemci sayısı 3 ve 9 olarak belirlenmiştir. Cooja ortamındaki sunucular üzerinde CoAP'ın Erbium uygulaması kullanılmış, istemci tarafında ise testlerde kullanılacak tüm CoAP uygulamalarını içeren Californium kullanılmıştır. Şekil 3 deneyler için tasarlanmış ağ topolojisini göstermektedir. 1 numaralı düğüm sınır yönlendiricisidir. Her bir düğüm, en yakın düğüme 30 metre uzakta olacak şekilde yerleştirilmiştir. İletim menzili 50 metre, girişim menzili ise 100 metre olarak belirlenmiştir. Düğümler üzerinde görünen %80 ise PDR değerini göstermektedir.



Şekil 3. Benzetimler için tasarlanan ağ topolojisi (Network topology designed for simulations)

Her benzetim senaryosu düğümler üzerindeki kuyruğun tamamen boşalması ve sonuçlara etki edecek olası bir gürültüyü engellemek adına aralarında 1 dakikalık bekleme süresi olacak şekilde 3 kez tekrarlanmıştır. İstemciler aynı anda çalışmaya başlayarak 3 dakika boyunca sunucudan veri çekmeye çalışmıştır. Tıkanıklığın daha iyi gözlenebilmesi için birden fazla istemci art-arda (back-to-back) olarak mesaj göndermiştir.



Şekil 4. Simülasyonlarda kullanılan ağ yığını ve her katmanda kullanılan protokoller (The network stack used in the simulations and the protocols used at each layer)

Şekil 4'te görülebileceği üzere CoAP sunucularının ağ yığınları farklı protokoller kullanılarak programlanmıştır. RDC katmanı için ContikiMAC ve NullRDC, ağ katmanındaki RPL için ise MRHOF ve OF0 kullanılmıştır. Taşıma

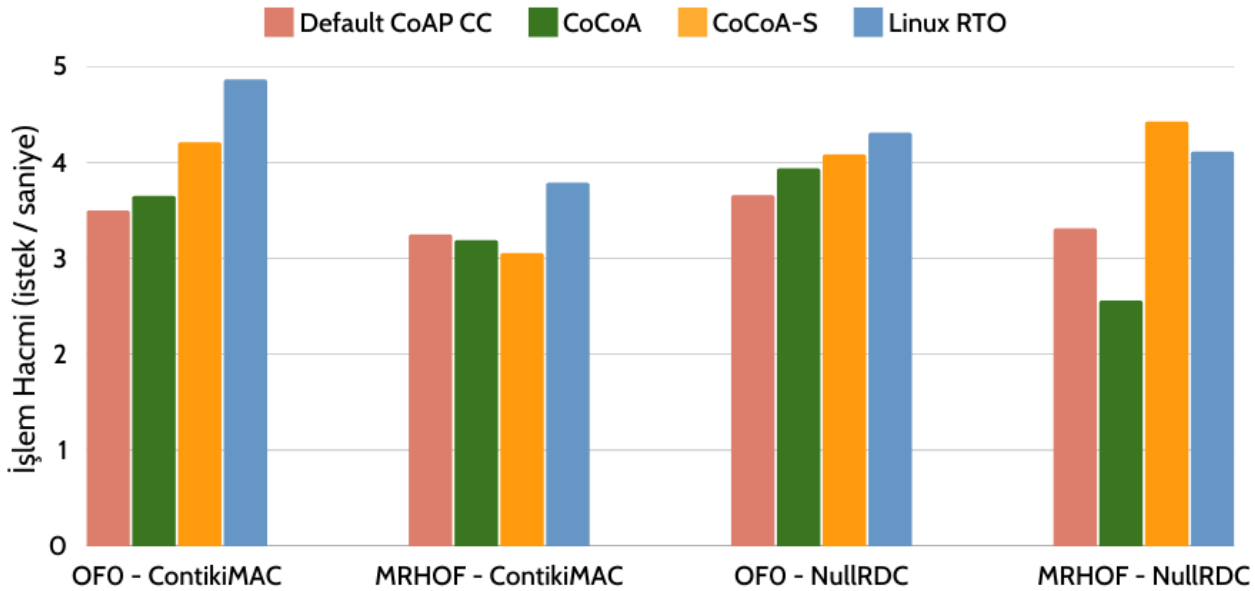
katmanında ise UDP kullanılmış, TCP ise kullanılmamıştır. NullMAC protokolü Contiki-NG üzerinde kullanıma hazır olsa da deneylere etkisi olmayacağından MAC katmanında sadece CSMA kullanılmıştır. Radyo katmanı için PDR değerleri Cooja üzerinde %80, %90, ve %100 olarak 3 farklı şekilde alınmıştır. İstemciler için ise uygulama katmanında Default CoAP CC, CoCoA, CoCoA-S ve Linux-RTO protokolleri kullanılmıştır. Bu protokollere benzer yöntemler kullanan tıkanıklık kontrol mekanizmaları PeakHopper-RTO ve Basic RTO daha önceki çalışmalar (Betzler vd., 2015; Betzler vd., 2016) incelendiğinde benzetim sonuçlarına etki etmeyeceği görüldüğü için kullanılmamıştır.

4. Benzetim Sonuçları (Simulation Results)

Hazırlanan deney ortamı üzerinde toplam 96 farklı senaryo kullanılmıştır. Her bir senaryo üç kez tekrarlandığı için de 288 adet test yapılmıştır. Bu testlerden alınan sonuçlar iki farklı metrik kullanılarak analiz edilmiştir. Bu metrikler işlem hacmi ve ortalama gecikmedir.

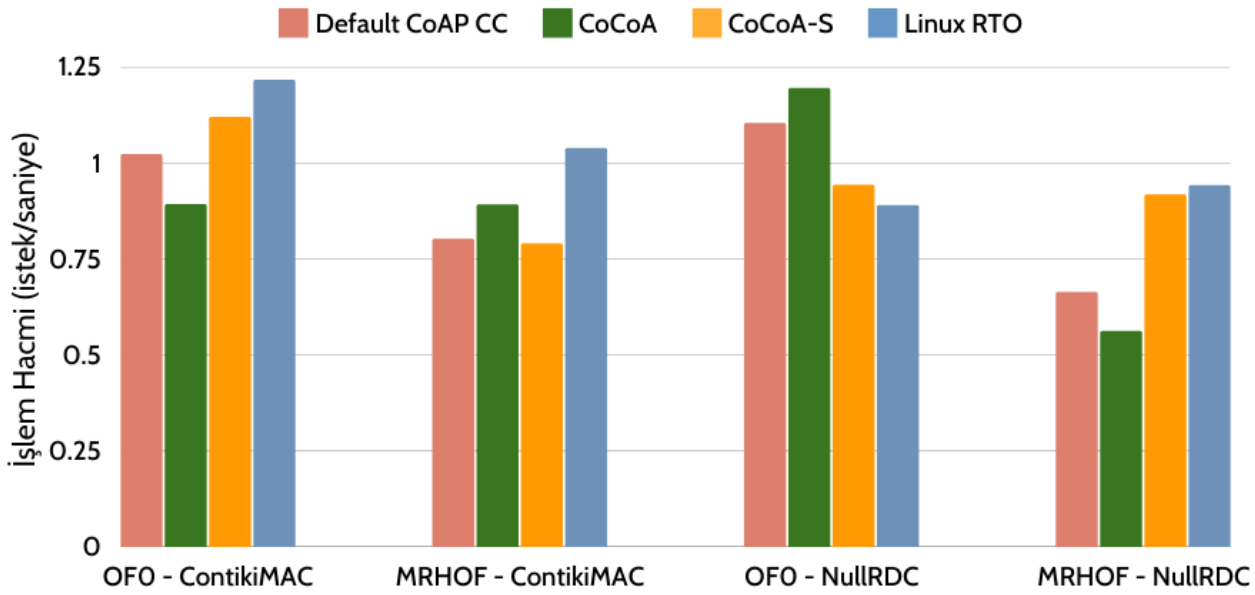
4.1. İşlem Hacmi (Throughput)

Bu bölümde performans kriteri olarak işlem hacmi alınmış, oluşturulan grafiklerle de tıkanıklık kontrol mekanizmaları ile amaç fonksiyonlarının karşılaştırması yapılmıştır.



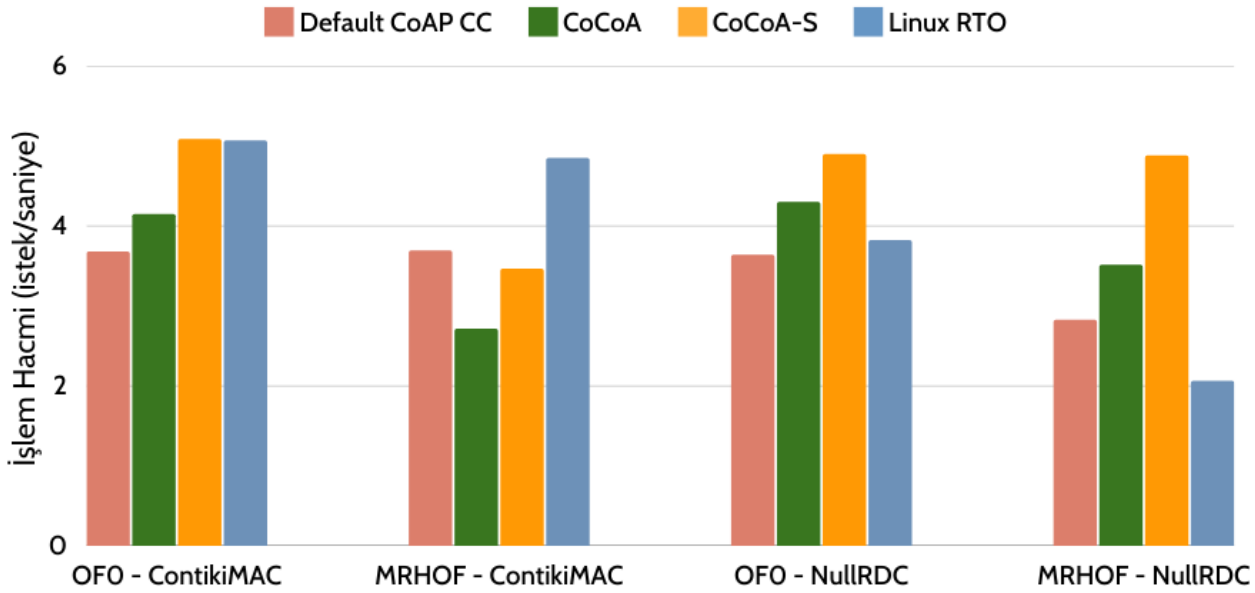
Şekil 5. PDR değeri %80 ve istemci sayısı 3 iken alınan işlem hacmi sonuçları
(Throughput results when the PDR value is 80% and the number of clients is 3)

Şekil 5'teki grafiğe bakıldığı zaman PDR değeri %80, istemci sayısı 3 iken hem ContikiMAC hem de NullRDC için OF0, MRHOF'tan daha iyi sonuçlar vermiştir. Bununla birlikte ContikiMAC yerine NullRDC kullanıldığında da OF0 yine daha iyi performans sergilemiştir. Grafiğe tıkanıklık kontrol mekanizmaları açısından bakıldığında genel olarak Linux-RTO'nun diğer mekanizmalardan daha başarılı olduğu görülmektedir.



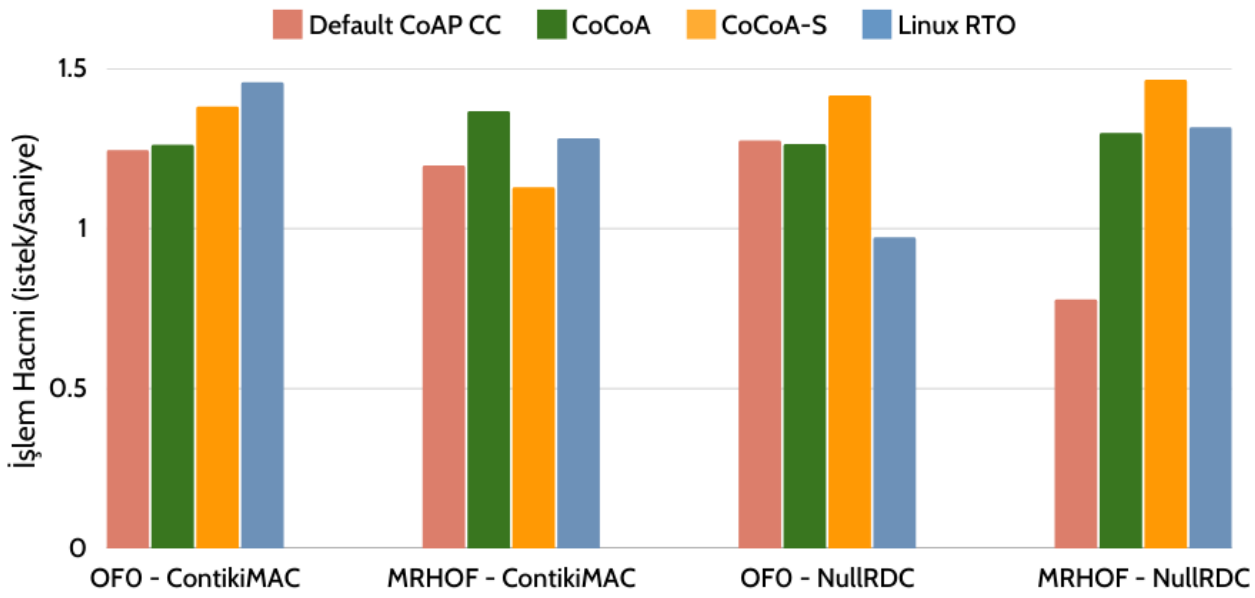
Şekil 6. PDR değeri %80 ve istemci sayısı 9 iken alınan işlem hacmi sonuçları
(Throughput results when the PDR value is 80% and the number of clients is 9)

Şekil 6'da görüldüğü üzere PDR değeri % 80 olduğunda ve istemci sayısı 9'a çıkarıldığında, MRHOF kullanılan senaryolarda en iyi performansı Linux-RTO gösterirken, OF0 kullanılan senaryolarda ise tıkanıklık kontrol mekanizması performansının RDC katmanındaki protokole bağlı olarak değiştiği görülmektedir. Genel anlamda bir değerlendirme yapılırsa işlem hacmi, Linux-RTO kullanılan senaryolarda daha yüksek çıkmıştır. OF0 kullanılan senaryolar incelendiğinde işlem hacminin MRHOF kullanılan senaryolara göre daha yüksek çıktığı görülmektedir.



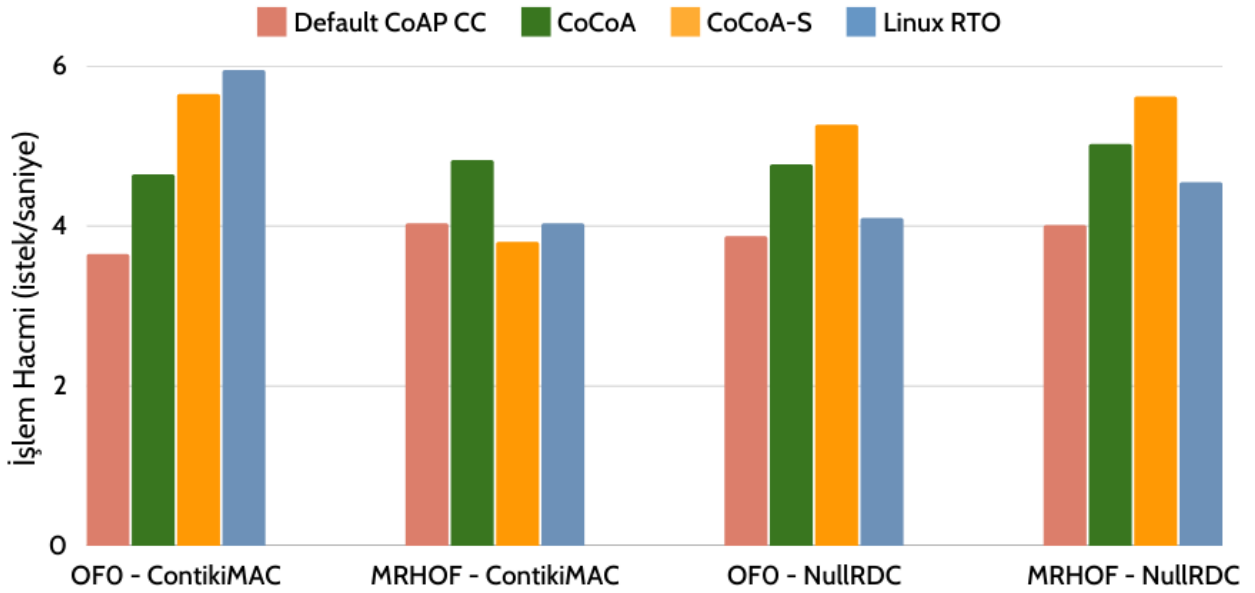
Şekil 7. PDR değeri %90 ve istemci sayısı 3 iken alınan işlem hacmi sonuçları
(Throughput results when the PDR value is 90% and the number of clients is 3)

PDR değerinin % 90 ve istemci sayısının 3 olarak belirlendiği benzetimlerde alınan sonuçların yer aldığı Şekil 7'ye bakıldığında genel itibariyle CoCoA-S diğer tıkanıklık algoritmalarına göre genel anlamda üstünlük sağlamıştır. OF0'ın tıkanıklık kontrol mekanizmalarının performansı açısından genellikle MRHOF'a göre daha üstün olduğu görülmektedir.



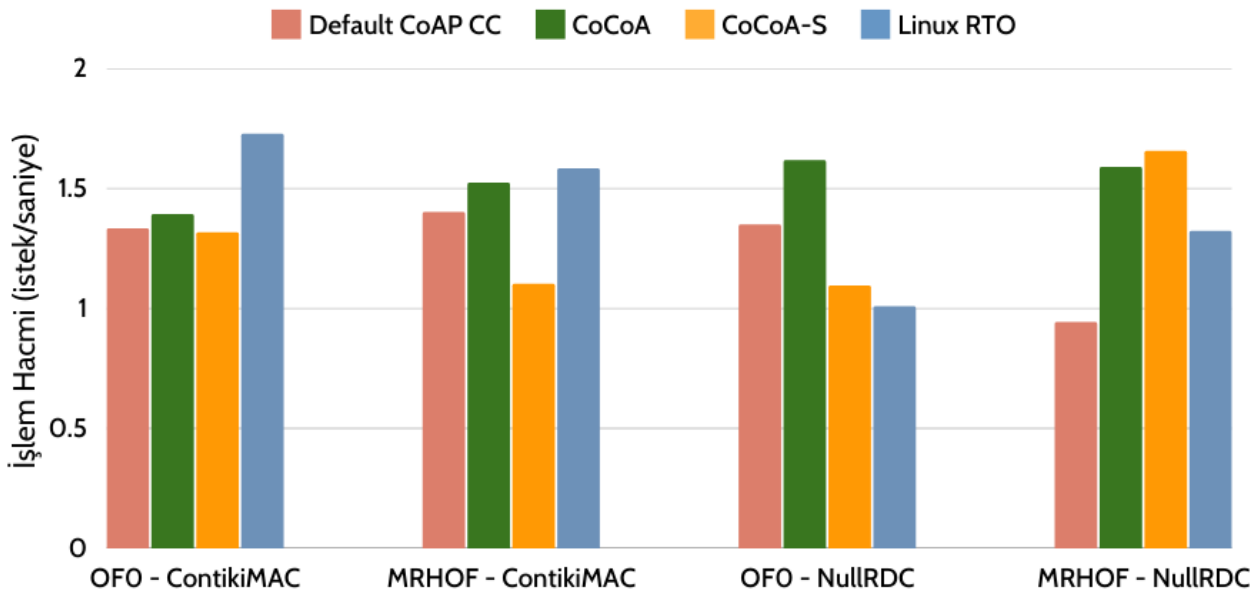
Şekil 8. PDR değeri %90 ve istemci sayısı 9 iken alınan işlem hacmi sonuçları
(Throughput results when the PDR value is 90% and the number of clients is 9)

İstemci sayısının 9 ve PDR değerinin ise % 90 olarak alındığı senaryolarda ise alınan sonuçlar benzer olsa da Şekil 8 dikkatli incelendiğinde ContikiMAC kullanılan senaryolarda OF0 ile tıkanıklık kontrol mekanizmalarının daha iyi performansa sahip olduğu, ancak CoCoA için bu durumun tam tersinin mevcut olduğu görülmektedir. NullRDC kullanılan senaryolarda MRHOF'un OF'a göre Linux-RTO üzerinde pozitif, Default CoAP CC açısından negatif etki yarattığı görülmüştür. Tıkanıklık kontrol mekanizmalarından CoCoA performans olarak CoCoA-S ten daha iyi sonuçlara sahiptir.



Şekil 9. PDR değeri %100 ve istemci sayısı 3 iken alınan işlem hacmi sonuçları
(Throughput results when the PDR value is 100% and the number of clients is 3)

PDR değerinin % 100 olarak alındığı ve istemci sayısının 3 olduğu senaryoların sonuçlarının gösterildiği Şekil 9'a bakıldığında, OF0 kullanıldığında ContikiMAC; MRHOF kullanıldığında ise NullRDC daha etkili performans vermiştir. Tıkanıklık kontrol mekanizmaları, NullRDC kullanılan senaryolarda amaç fonksiyonlarının performansa etkisi pek olmasa da ContikiMAC kullanılan senaryolarda OF0 ile elde edilen sonuçlarının MRHOF'tan daha iyi olduğu görülmektedir. Her şart altında en stabil sonuçları CoCoA verse de en başarılı performansı CoCoA-S sağlamıştır.

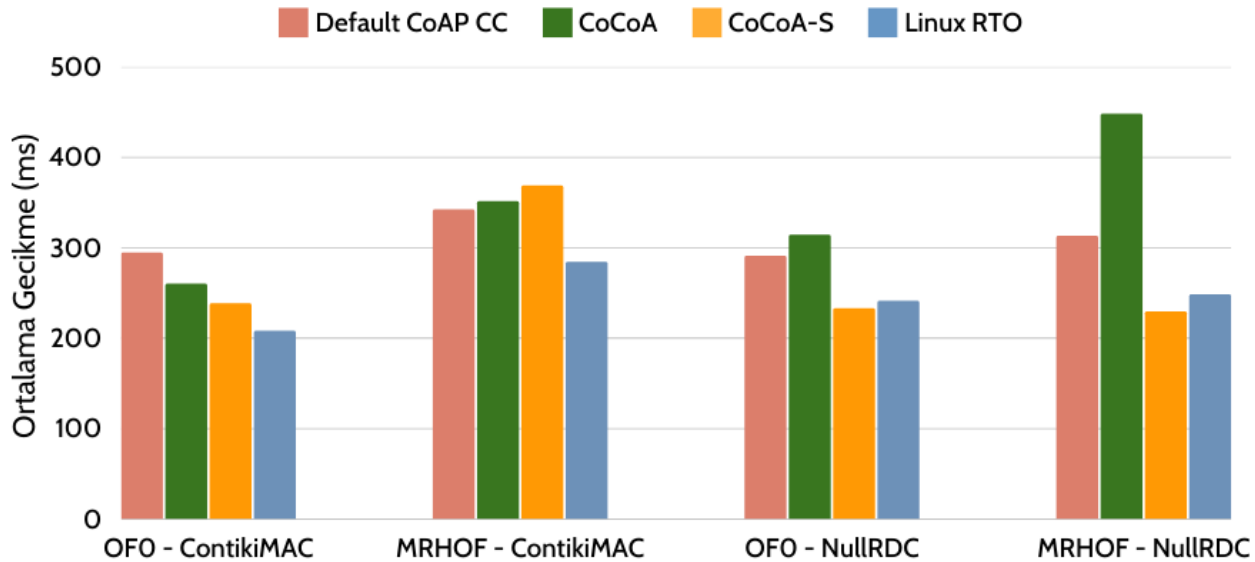


Şekil 10. PDR değeri %100 ve istemci sayısı 9 iken alınan işlem hacmi sonuçları
(Throughput results when the PDR value is 100% and the number of clients is 9)

Şekil 10 PDR değerinin % 100 olarak alındığı ve istemci sayısının 9 olduğu senaryoların sonuçlarını göstermektedir. Bu sonuçlar incelendiğinde ContikiMAC kullanıldığında Linux-RTO daha verimli sonuçlar verirken, NullRDC kullanılan testlerde CoCoA'nın daha iyi sonuçlar verdiği görülmüştür. ContikiMAC kullanılan testlerde MRHOF'un kullanılması CoCoA-S ile Linux-RTO'nun verimliliğini düşürdüğü, Default CoAP CC ile CoCoA'nın daha verimliliğini artırdığı gözlenmiştir. NullRDC kullanılan senaryolar baz alındığında ise bunun tam tersi olarak CoCoA-S ile Linux-RTO'nun performansının yükseldiği, Default CoAP CC ile CoCoA'nın performansının ise düştüğü görülmüştür.

4.2. Ortalama Gecikme (Average Delay)

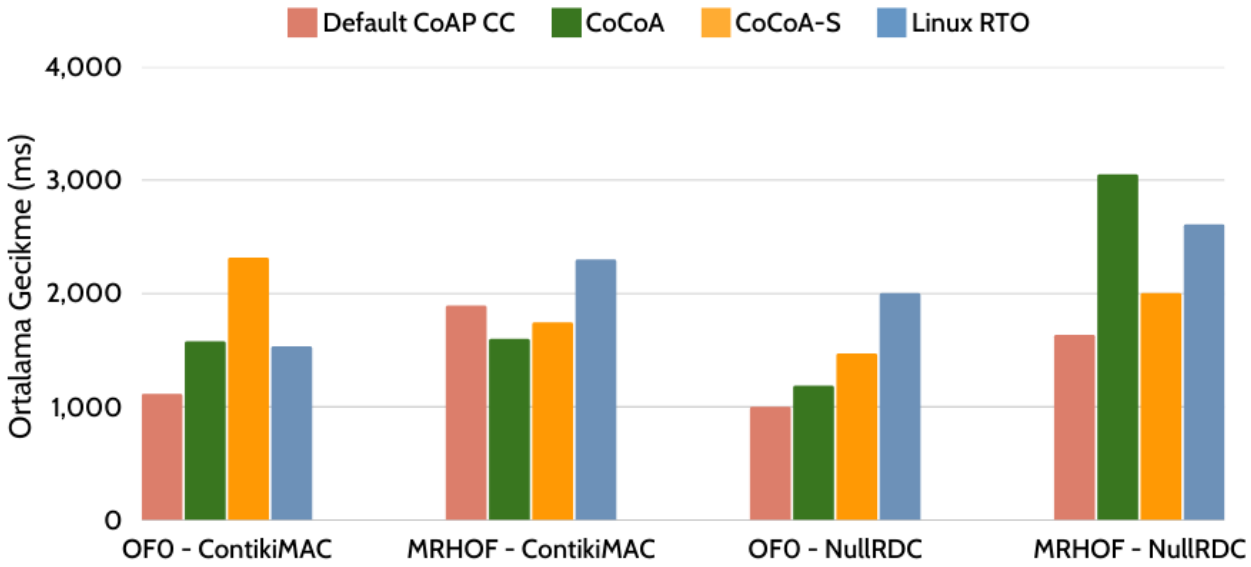
Bu bölümde performans kriteri olarak ortalama gecikme süresi alınmış, bir önceki bölümde de olduğu gibi oluşturulan grafiklerle tıkanıklık kontrol mekanizmaları ile amaç fonksiyonlarının karşılaştırması yapılmıştır.



Şekil 11. PDR değeri %80 ve istemci sayısı 3 iken alınan ortalama gecikme sonuçları
(Average latency results at 80% PDR and 3 clients)

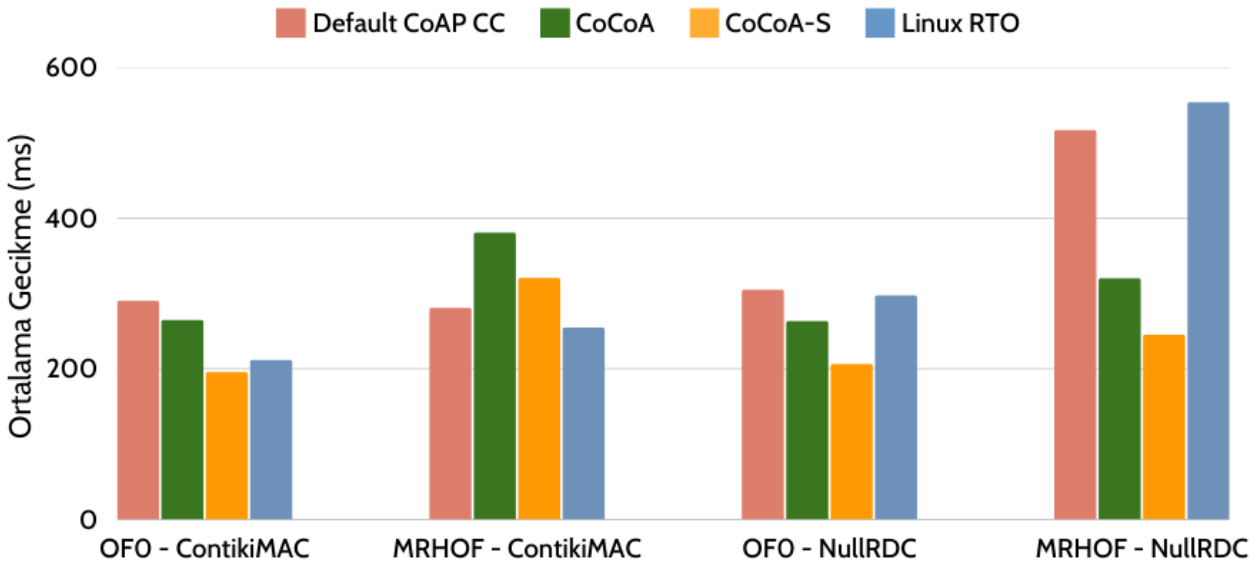
Ortalama gecikme sonuçlarını, PDR değeri % 80 ve istemci sayısı 3 iken Şekil 11 yardımıyla incelediğimizde Linux-RTO diğer mekanizmalara göre performans konusunda üstünlük sağlamıştır. NullRDC kullanılan senaryolarda CoCoA için OF0 yerine MRHOF kullanılmasının performansı düşürerek daha fazla gecikmeye sebebiyet verdiği, ancak diğer algoritmaların çok fazla etkilenmediği görülmektedir. Ancak ContikiMAC için elde edilen sonuçlar

incelendiğinde MRHOF'un her algoritma için daha fazla gecikmeye neden olduğu ve OF0'ın daha başarılı olduğu sonucu çıkmıştır.



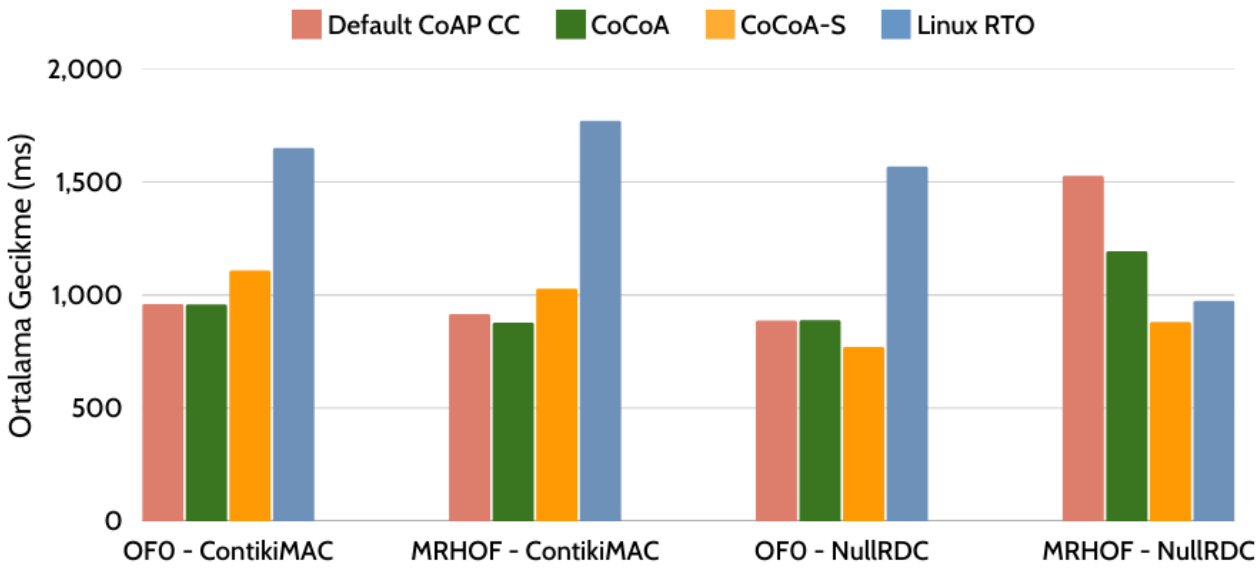
Şekil 12. PDR değeri %80 ve istemci sayısı 9 iken alınan ortalama gecikme sonuçları
(Average latency results at 80% PDR and 9 clients)

Ancak Şekil 12 incelendiğinde PDR değeri % 80 iken istemci sayısı 9 olduğunda Default CoAP CC daha verimli sonuçlar vermiştir. MRHOF ise OF0 ile kıyaslandığında daha önceki sonuçlara benzer şekilde daha fazla gecikmeye sebebiyet vermiştir. OF0 kullanıldığında NullRDC, MRHOF kullanıldığında ise ContikiMAC protokolleri kullanılan senaryolarda daha iyi performans alınmıştır.



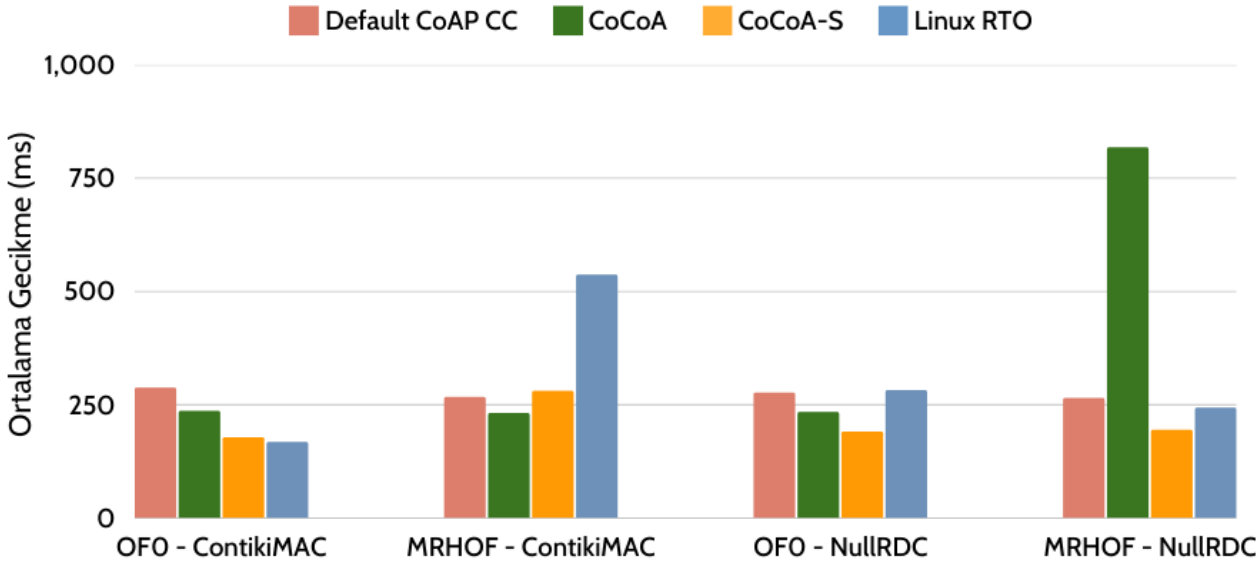
Şekil 13. PDR değeri %90 ve istemci sayısı 3 iken alınan ortalama gecikme sonuçları
(Average latency results at 90% PDR and 3 clients)

Şekil 13'te görüldüğü üzere PDR değerini % 90 olarak aldığımızda CoCoA-S diğer algoritmalara nazaran daha iyi performans sergilemiştir. Yine OF0'ın MRHOF'a göre üstünlüğü kendini göstermektedir. Genel itibariyle ContikiMAC, NullRDC'ye göre daha iyi sonuçlar vermiştir.



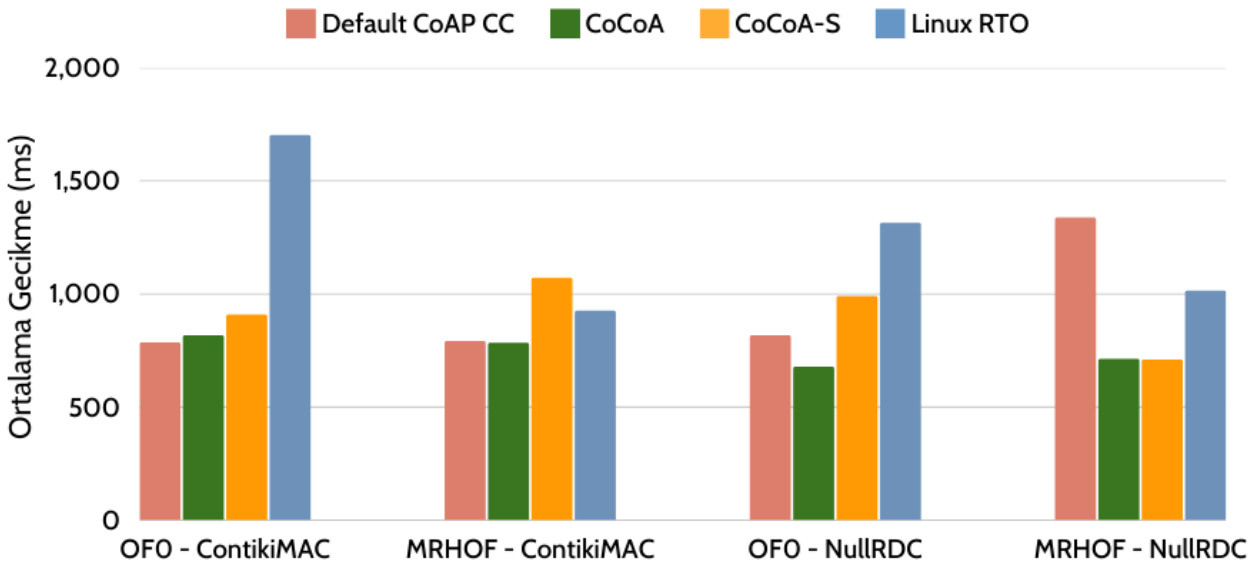
Şekil 14. PDR değeri %90 ve istemci sayısı 9 iken alınan ortalama gecikme sonuçları
(Average latency results when PDR is 90% and number of clients is 9)

PDR değeri %90 ve istemci sayısı 9 olarak alındığında, Şekil 14'te de görüldüğü üzere, dikkat çeken husus Linux-RTO'nun diğer mekanizmalara göre ciddi anlamda kötü performans sergilediğidir. Bunun dışında ise istemci sayısının 3 olduğu senaryolarla benzer sonuçlar alınmıştır.



Şekil 15. PDR değeri %100 ve istemci sayısı 3 iken alınan ortalama gecikme sonuçları
(Average latency results when PDR is 100% and number of clients is 3)

PDR değeri %100 olarak belirlenen, 3 istemci kullanılan testlerin sonuçlarının gösterildiği Şekil 15 incelendiğinde dikkat çeken iki sonuç bulunmaktadır. CoCoA'nın MRHOF ve NullRDC kullanılan testlerde, Linux-RTO'nun ise MRHOF ve ContikiMAC ile bir arada kullanıldığında oldukça kötü performans sergilediği görülmüştür. Bu testlerde tıkanıklık kontrol mekanizmaları içerisinden en başarılı sonuçları CoCoA-S almıştır.



Şekil 16. PDR değeri %100 ve istemci sayısı 9 iken alınan ortalama gecikme sonuçları
(Average latency results when PDR is 100% and number of clients is 9)

Ancak PDR değeri %100 iken 9 istemci ile veri çekilen testler Şekil 16 yardımıyla incelendiğinde CoCoA'nın daha verimli olduğu görülmektedir. OF0 kullanılan testlerde alınan sonuçların birbirine daha yakın olduğu görülürken en belirgin sapmalar MRHOF kullanılan testlerde görülmektedir. Bu sebeple OF0'ın hem daha verimli hem de daha stabil bir çalışma performansına sahip olduğu görülmüştür.

5. Sonuç ve Tartışma (Result and Discussion)

Bu çalışmada RPL amaç fonksiyonları ve tıkanıklık kontrol mekanizmaları arasında performansa dair bir ilişkinin olup olmadığı araştırılmıştır. Tüm sonuçlar incelendiğinde, en iyi performansın CoCoA-S'in OF0 ile beraber kullanıldığında alındığı görülmüştür. Ancak sonuçlar çoğunlukla bunu gösterse de bazı istisnai durumlar da vardır. Bu yüzden her durum için amaç fonksiyonları ve tıkanıklık kontrol mekanizmalarının en iyi kombinasyonunu önermek yerine, kurulacak ağın karakterine ve ihtiyaçlarına göre bu mekanizmaların belirlenmesi gerekmektedir. Bu yüzden farklı benzetimler ile ağ kurulmadan önce test yapılması faydalı olacaktır.

Örneğin, CoCoA-S 3 istemci ile aynı anda veri çekilen bir ağda daha iyi sonuçlar verirken, istemci sayısı 9 olduğunda CoCoA daha iyi sonuçlar vermektedir. Buradan da istemci sayısının fazla olduğu ağ topolojilerinde CoCoA'nın kullanılmasının daha iyi olacağı söylenebilir. Benzer şekilde, genelde OF0'ın daha iyi sonuçlar verdiği görülse de istemci sayısı arttıkça Linux-RTO kullanılan topolojilerde MRHOF'un OF0 dan daha iyi sonuçlar verdiği görülmüştür. Bu nedenle bu çalışmada elde edilen sonuçların göz önüne alınmasıyla, probleme özgü ağ topolojisinin karakterinin belirlenmesi en iyi yöntem olacaktır.

Bu çalışma ile, RPL amaç fonksiyonları ile farklı CoAP tıkanıklık kontrol mekanizmalarının birbirleri ile kombinasyonlarının kullanıldığı ağ topolojilerinin performansı iki farklı metrik kullanılarak ölçülmüş ve sonuçlar analiz edilmiştir. Şimdiye kadar IoT ve WSN ağları için tıkanıklık mekanizması olarak tamamen standart haline dönüştürülmüş bir protokol bulunmaması, IoT ve WSN ağlarındaki ağ örüntüsünün geleneksel ağlardan farklılığından ötürü mevcut TCP tıkanıklık kontrol mekanizmalarının yeterli olmamasından dolayı, bu çalışmadaki performans analizinin bir genelleme oluşturma adına oldukça faydalı olacağına inanılmaktadır.

Benzetim sonuçları incelendiğinde OF0 ve CoCoA-S birlikte kullanıldığında sonuçlar diğer olası kombinasyonlardan daha iyi performans göstermiştir. Ayrıca RDC katmanında ContikiMAC kullanılmasının da yine performansı olumlu yönde etkilediği görülmüştür. Bir IoT ağı kurmadan önce bu çalışmada olduğu gibi hazırlanacak farklı test senaryoları ile bir ön çalışma yapılması, kurulacak IoT ağ katmanının en iyi performansı vermesi adına oldukça faydalı olacaktır.

Sonraki çalışmalarda Default CoAP CC ile daha uyumlu çalışacak amaç fonksiyonlarının tasarlanması araştırmaya açıktır. Bununla birlikte OF0 ve MRHOF'tan daha efektif bir RPL amaç fonksiyonu geliştirilecektir. Ayrıca farklı CoAP CC mekanizmaları ile beraber bu çalışmada yer almayan 6TiSCH standardının (Vilajosana vd., 2020) performanslarının incelenmesi hedeflenmektedir.

Çıkar Çatışması (Conflict of Interest)

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

Kaynaklar (References)

A Quick Introduction to the Erbium (Er) REST Engine. (2016). Çevrimiçi: https://github.com/contiki-os/contiki/tree/master/examples/er-rest-example (Erişim tarihi: 11.12.2020)

Al-Kashoash, H. A., Al-Nidawi, Y., & Kemp, A. H. (2016). Congestion-aware RPL for 6L0WPAN networks. 2016 Wireless Telecommunications Symposium (WTS), 1-6.

Abuein, Q., Yassein, M. B., Shatnawi, M. Q., & Bani-Yaseen, L. (2016). Performance Evaluation of Routing Protocol (RPL) for Internet of Things. International Journal of Advanced Computer Science and Applications, 7.

Betzler, A., Gomez, C., Demirkol, I., & Paradells, J. (2015). CoCoA+: An advanced congestion control mechanism for CoAP. Ad Hoc Networks, 33, 126-139.

Betzler, A., Gomez, C., Demirkol, I., & Paradells, J. (2016). CoAP Congestion Control for the Internet of Things. IEEE Communications Magazine, 54(7), 154-160.

Buratti, C., Martalo, M., Ferrari, G., & Verdone, R. (2011). Sensor Networks with IEEE 802.15.4 Systems: Distributed Processing, MAC, and Connectivity (1 ed.). Berlin: Springer.

Buratti, C., Martalo, M., Verdone, R., & & Ferrari, G. (2011). Sensor Networks with IEEE 802.15.4 Systems. In Signals and Communication Technology. Berlin: Springer.

Cao, H., Wachowicz, M., Renso, C., & Carlini, E. (2019). Analytics Everywhere: Generating Insights From the Internet of Things. IEEE Access, 7, 71749-71769.

CoAP in Java. (2020). Çevrimiçi: http://www.eclipse.org/californium (Erişim tarihi: 14.10.2020)

Contiki-NG: The OS for Next Generation IoT Devices. (2020). Çevrimiçi: https://github.com/contiki-ng/contiki-ng (Erişim tarihi: 18.03.2021)

Cooja Simulator. (2016). Çevrimiçi: http://anrg.usc.edu/contiki/index.php/Cooja_Simulator (Erişim tarihi: 14.10.2020)

Demir, A. K., & Abut, F. (2018). Comparison of CoAP and CoCoA Congestion Control Mechanisms in Network Topologies. GÜFBED CMES 2018 Sempozyum Ek Sayısı, (pp. 53-60). Gümüşhane.

Development tools Archives. (2017). Çevrimiçi: https://zolertia.io/product-category/development-tools/ (Erişim tarihi: 23.03.2020)

Dunkels, A. (2011). The ContikiMAC Radio Duty Cycling Protocol.

Dunkels, A., Gronvall, B., & Voigt, T. (2004). Contiki - a lightweight and flexible operating system for tiny networked sensors. 29th Annual IEEE International Conference on Local Computer Networks., (pp. 455-462). Tampa, FL, USA.

Espressif. (2020). ESP8266EX Datasheet. Retrieved from Esspressif: https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf

Halcu, I., Stamatescu, G., Stamatescu, I., & Sgârciu, V. (2016). IPV6 Sensor Networks Modeling for Security and Communication Evaluation. In E. Pricop, & G. Stamatescu, Recent Advances in Systems Safety and Security (p. 251). Berlin: Springer International Publishing.

IEEE 802.15 WPAN™ Task Group 4 (TG4). (2010). Çevrimiçi: https://www.ieee802.org/15/pub/TG4.html (Erişim tarihi: 12.05.2020)

Insense Examples. (2011). Çevrimiçi: https://insense.cs.st-andrews.ac.uk/examples (Erişim tarihi: 23.03.2020)

İkizoğlu, K. (2020) Nesnelerin İnterneti için NodeMCU. Çevrimiçi: http://blog.ikizoglu.com/2020/02/nesnelerin-interneti-icin-nodemcu/ (Erişim tarihi: 12.04.2022)

Kovatsch, M., Lanter, M., & Shelby, Z. (2014). Californium: Scalable cloud services for the Internet of Things with CoAP. International Conference on the Internet of Things, (pp. 1-6). Seoul, Korea.

Lamaazi, H., Benamar, N., & Jara, A. J. (2017). Study of the Impact of Designed Objective Function ın the RPL-Based Routing Protocol. In R. El-Azouzi, D. Menasche, E. Sabir, F. De Pellegrini, & M. Benjillai, Advances in Ubiquitous Networking 2. UNet 2016. Lecture Notes in Electrical Engineering (Vol. 397). Singapore: Springer.

Locke, D. (2010). MQ telemetry transport (MQTT) v3. 1 protocol specification. IBM developerWorks Technical Library.

MAC protocols in ContikiOS. (2014). Çevrimiçi: https://anrg.usc.edu/contiki/index.php/MAC_protocols_in_ContikiOS (Erişim tarihi: 23.12.2019)

Mardini, W., Ebrahim, M., & Al-Rudaini, M. (207). Comprehensive performance analysis of RPL objective functions in IoT networks. International Journal of Communication Networks and Information Security, 9, 323-332.

Onwuegbuzie, I., Ajibade, S., Fele, T., & Akinwamide, S. (2020). Performance Evaluation of RPL Objective Function: A Case of Contiki Operation System. Preprints.

Pradeska, N., Widyawan, Najib, W., & Kusumawardani, S. S. (2016). Performance analysis of objective function MRHOF and OF0 in routing protocol RPL IPV6 over low power wireless personal area networks (6LoWPAN). 2016 8th International Conference on Information Technology and Electrical Engineering (ICITEE), (pp. 1-6). Yogyakarta.

Qasem, M., Altwassi, H. S., Yassein, M. B., & Al-Dubai, A. Y. (2015). Performance Evaluation of RPL Objective Functions. 2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing, 1606-1613.

Rayes, A., & Salam, S. (2017). Internet of Things From Hype to Reality: The Road to Digitization. Springer.

RFC768, "User Datagram Protocol", (2012).

RFC6550, "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks", (2012).

RFC6552, "Objective Function Zero for the Routing Protocol for Low-Power and Lossy Networks (RPL)", (2012).
RFC6719, "The Minimum Rank with Hysteresis Objective Function", (2012).
RFC7228, "Terminology for Constrained-Node Networks", (2014).
Sonia, K., & Virani, H. (2021). Design of an Efficient RPL Objective Function for Internet of Things Applications. *International Journal of Advanced Computer Science and Applications, 12*(6), 228-235.
The Z1 Mote. (2018). Çevrimiçi: https://github.com/Zolertia/Resources/wiki/The-Z1-mote (Erişim tarihi: 25.12.2019)
Tiburski, R., Moratelli, C. R., Filho, S. J., & Neves, M. V. (2019). Lightweight Security Architecture Based on Embedded Virtualization and Trust Mechanisms for IoT Edge Devices. *IEEE Communications Magazine, 57*(2), 67-73.
Xiao, W., Liu, J., Jiang, N., & Shi, H. (2014). An optimization of the object function for routing protocol of low-power and Lossy networks. *Systems and Informatics (ICSAI) 2014 2nd International Conference on. IEEE*, 515-519.
Vasseur, J.-P., & Dunkels, A. (2010). Interconnecting Smart Objects with IP. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.
Vilajosana, X., Tuset, P., Watteyne, T., & Pister, K. (2015). OpenMote: Open-Source Prototyping Platform for the Industrial IoT. ADHOCNETS, 155.
Vilajosana, X., Watteyne, T., Chang, T., Vučinić, M., Duquennoy, S., & Thubert, P. (2020). IETF 6TiSCH: A Tutorial. IEEE Communications Surveys & Tutorials, 22(1), 595-615.