T.C.
İSTANBUL ÜNİVERSİTESİ-CERRAHPAŞA
LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

YÜKSEK LİSANS TEZİ

YAZILIM TANIMLI AĞDA FARKLI YÖNLENDİRME ALGORİTMALARININ ÇALIŞTIRILMASI VE AKIŞ KONTROLÜ

HİLMİ TUNAHAN İLHAN

DANIŞMAN
Doç. Dr. Derya YILTAŞ KAPLAN

Bilgisayar Mühendisliği Anabilim Dalı

Bilgisayar Mühendisliği Programı

İSTANBUL-2020

Bu çalışma, 13.04.2020 tarihinde aşağıdaki jüri tarafından Bilgisayar Mühendisliği Anabilim Dalı, Bilgisayar Mühendisliği Programında Yüksek Lisans tezi olarak kabul edilmiştir.

Tez Jürisi

Doç. Dr. Derya YILTAŞ KAPLAN(Danışman)
İstanbul Üniversitesi-Cerrahpaşa
Mühendislik Fakültesi

Prof. Dr. Ahmet SERTBAŞ
İstanbul Üniversitesi-Cerrahpaşa
Mühendislik Fakültesi

Doç. Dr. Şerif BAHTİYAR
İstanbul Teknik Üniversitesi
Bilgisayar ve Bilişim Fakültesi

20.04.2016 tarihli Resmi Gazete'de yayımlanan Lisansüstü Eğitim ve Öğretim Yönetmeliğinin 9/2 ve 22/2 maddeleri gereğince; Bu Lisansüstü teze, İstanbul Üniversitesi-Cerrahpaşa'nın abonesi olduğu intihal yazılım programı kullanılarak Lisansüstü Eğitim Enstitüsü'nün belirlemiş olduğu ölçütlere uygun rapor alınmıştır.

Bu tez, İstanbul Üniversitesi Bilimsel Araştırma Projeleri Yürütücü Sekreterliğinin ..... numaralı projesi ile desteklenmiştir.
Bu tez, ... numaralı ..... projesi ile desteklenmiştir.

ÖNSÖZ

Tez çalışmam süresince akademik desteklerini esirgemeyen değerli hocam Sayın Doç. Dr. Derya Yıltaş Kaplan'a, 116E905 numaralı proje kapsamında çalışabildiğim için TÜBİTAK'a ve tezimi destekleyip akademik izin veren Türk Hava Yolları Ar-Ge ekibine şükranlarımı sunarım.

Hayatım boyunca arkamda durdukları için aileme ve tez süresince yanımda olan Hande Yavaş'a ayrıca teşekkür ederim.

Nisan 2020 Hilmi Tunahan İLHAN

İÇİNDEKİLER

Sayfa No

ÖNSÖZ ..... **iv**
İÇİNDEKİLER ..... **v**
ŞEKİL LİSTESİ ..... **viii**
TABLO LİSTESİ ..... **x**
SİMGE VE KISALTMA LİSTESİ ..... **xi**
ÖZET ..... **xii**
SUMMARY ..... **xiii**
1. GİRİŞ **1**
2. GENEL KISIMLAR **2**
2.1. AĞ 2
2.1.1. Ağ Bağlantı Türleri(Topoloji) 3
2.1.2. Ağ Cihazları 7
2.1.2.1. Göbek(Hub) *7*
2.1.2.2. Anahtar(Switch) *8*
2.1.2.3. Köprü(Bridge) *8*
2.1.2.4. Yönlendiriciler(Routers) *9*
2.1.3. 5 Katman Referans Modeli 10
2.2. YTA 12
2.2.1. YTA Temel Prensibi 12
2.2.1.1. Düzlem Ayrılığı *12*
2.2.1.2. Basitleştirilmiş Cihaz ve Merkezi Kontrol *13*
2.2.1.3. Ağ Otomasyonu ve Sanallaştırma *13*
2.2.1.4. Açıklık ve Anlaşılabilir Olmak *14*
2.2.1.5. Bilinen YTA Kontrolör Uygulamaları 14
2.3. DIJKSTRA ALGORITMASI 15
2.3.1. Algoritmanın Açıklaması 15
2.3.2. Algoritma Sözde Kodu(Pseudocode) 16
2.4. AÇIK ARTIRMA(AUCTION) ALGORİTMASI 16
2.4.1. Algoritmanın Açıklaması 16
2.4.2. Algoritma Sözde Kodu 17

2.5. İKİLİ TIRMANMA(DUAL ASCENT) ALGORİTMASI 18
2.5.1. Algoritmanın Açıklaması 18
2.5.2. Algoritma Sözde Kodu 19
3. MALZEME VE YÖNTEM 21
3.1. FLOODLIGHT 21
3.2. MININET 21
3.3. UYGULAMA 22
3.3.1. Kurulum 22
3.3.1.1. Manuel İşletim Sistemi Kurulumu 22
3.3.1.2. Hazır Sanal Makine Dosyası Kurulumu 23
3.3.2. Topoloji Oluşturma ve Python Kodu 25
3.3.2.1. Tek Alt Ağ İçeren Topoloji 25
3.3.2.2. Birden Fazla Alt Ağ İçeren Topoloji 27
3.3.2.3. Topoloji Kullanımları 29
3.3.3. Proje Gelişimi 29
3.3.3.1. Açık Artırma Algoritması(Auction) 30
3.3.3.2. İkili Tırmanma Algoritması(Dual Ascent) 30
3.3.3.3. Test Süreci 31
3.3.4. Algoritmaların Geliştirilmesi ve Uygunluğun Arıttırılması 32
3.3.5. Normalizasyon 32
3.3.5.1. Min-Max Ölçekleme(Rescaling) Yöntemi İle Normalizasyon 33
3.3.5.2. Standart Skorlama(Standart Scoring) Yöntemi İle Normalizasyon 34
3.3.6. Diğer Çalışmalar 36
3.3.7. Yazılım Geliştirmede Kullanılan Teknolojiler ve Ürünler 37
3.3.7.1. Java 37
3.3.7.2. Intellij IDEA 37
3.3.7.3. Ubuntu 18.04 38
4. BULGULAR 39
4.1. BULUNAN YOL KARŞILAŞTIRMALARI 39
4.1.1. Alt Ağ İçeren Topoloji Testleri 42
4.1.1.1. Algoritmaların Tek Parametreli Çalıştırılması Testi 42
4.1.1.2. Min-Max Ölçekleme Yöntemi Kullanılarak Bant Genişliği Değerinin Hesaba Katılması Testi 42
4.1.1.3. Standart Skorlama Yöntemi Kullanılarak Bant Genişliği Değerinin Hesaba Katılması Testi 42

4.1.2. Alt Ağ İçermeyen Topoloji Testleri 43
4.1.2.1. Algoritmaların Tek Parametreli Çalıştırılması Testi 43
4.1.2.2. Min-Max Ölçekleme Yöntemi Kullanılarak Bant Genişliği Değerinin Hesaba Katılması Testi 43
4.1.2.3. Standart Skorlama Yöntemi Kullanılarak Bant Genişliği Değerinin Hesaba Katılması Testi 44
4.2. AĞ YOĞUNLUK SONUÇLARI 44
4.2.1. Min-Max Ölçekleme Testleri 47
4.2.1.1. Açık Artırma Testi 47
4.2.1.2. Bellman-Ford Testi 47
4.2.1.3. Dijkstra Testi 48
4.2.2. İkili Tırmanma Testi 48
4.2.3. Standart Skorlama Testleri 49
4.2.3.1. Açık Artırma Testi 49
4.2.3.2. Bellman-Ford Testi 49
4.2.3.3. Dijkstra Testi 50
4.2.4. İkili Tırmanma Testi 50
5. TARTIŞMA VE SONUÇ 51
KAYNAKLAR 53
EKLER 55
ÖZGEÇMİŞ 56

ŞEKİL LİSTESİ

Sayfa No

Şekil 2.1: Tam Örgü Ağ(Full-Mesh) Topolojisi [2] 4

Şekil 2.2: Bus Topolojisi [2] 4

Şekil 2.3: Yıldız(Star) Topolojisi [2] 5

Şekil 2.4: Halka(Ring) Topolojisi [2] 6

Şekil 2.5: Ağaç (Tree) Topolojisi [2] 7

Şekil 2.6: Yönlendirici Çalışma Şekli[3] 10

Şekil 3.1: Tek Alt Ağ Topolojisi[18] 25

Şekil 3.1: Tek alt Ağ topolojisi Python Kodu 26

Şekil 3.2: Çoklu Alt Ağ Topolojisi 27

Şekil 3.3: Çoklu Alt Ağ Topolojisi Python Kodu 28

Şekil 3.4: Açık Artırma Algoritması Kodu 30

Şekil 3.5: İkili Tırmanma Algoritması Kodu 31

Şekil 4.1: Alt Ağ İçeren Topolojilerde Gerçekleştirilen Testler 41

Şekil 4.2: Alt Ağ İçermeyen Topolojilerde Gerçekleştirilen Testler 41

Şekil 4.3: Min-Max Ölçekleme Yöntemi Sonuçları 46

Şekil 4.4: Standart Skorlama Yöntemi Sonuçları 46

Şekil 4.5: Açık Artırma Algoritması Performans Sonucu 47

Şekil 4.6: Bellman Ford Algoritması Performans Sonucu 48

Şekil 4.7: Dijkstra Algoritması Performans Sonucu 48

Şekil 4.8: İkili Tırmanma Algoritması Performans Sonucu 48

Şekil 4.9: Açık Artırma Algoritması Performans Sonucu 49

Şekil 4.10: Bellman Ford Algoritması Performans Sonucu 50

Şekil 4.11: Dijkstra Algoritması Performans Sonucu 50

Şekil 4.12: Dijkstra Algoritması Performans Sonucu.....50

TABLO LİSTESİ

Sayfa No

Şekil tablosu öğesi bulunamadı.

SİMGE VE KISALTMA LİSTESİ

Simgeler **Açıklama**

X' : Normalizasyon Sonucu
Xmin : Dizinin En Küçük Elemanı
Xmax : Dizinin en büyük elemanı
σ : Ortalama Değer
μ : Standart Sapma

Kısaltmalar **Açıklama**

ADU : Application Data Unit
LAN : Local Area Network
MAC : Media Access Control
MAN : Metropolitan Area Network
OSI : Open Systems Interconnection
PAN : Personal Area Network
REST : Representational State Transfer
TCP : Transmission Control Protocol
UDP : User Datagram Protocol
VM : Virtual Machine (Sanal Makine)
WAN : Wide Area Network
YTA : Yazılım Tanımlı Ağ(Software Defined Network)

ÖZET

YÜKSEK LİSANS TEZİ

YAZILIM TANIMLI AĞDA FARKLI YÖNLENDIRME ALGORITMALARININ ÇALIŞTIRILMASI VE AKIŞ KONTROLÜ

Hilmi Tunahan İLHAN

İstanbul Üniversitesi-Cerrahpaşa

Lisansüstü Eğitim Enstitüsü

Bilgisayar Mühendisliği Anabilim Dalı

Danışman : Doç. Dr. Derya Yıltaş KAPLAN

Bu tez çalışması bir Yazılım Tanımlı Ağ(YTA) uygulamasının ağ yönlendirme kabiliyetinin başka algoritmalar da eklenerek geliştirilmesiyle yapılmıştır. Çalışma yapılırken YTA olarak Floodlight, yazılım geliştirme IDE'si olarak ise Intellij IDEA ve yazılım dili olarak ise Java 8 kullanılmıştır. Yönlendirme kabiliyetinin zenginleştirilmesi amacıyla yeni eklenen algoritmalar ve hâlihazırda Floodlight tarafından kullanılan Dijkstra algoritması kullanılmıştır. Açık Artırma ve İkili Tırmanma yaklaşımlı iki algoritma baştan geliştirilmiştir ve Dijkstra ile birlikte Floodlight içerisine entegre edilmiştir. Buna ek olarak tüm algoritmaların karar verme esnasında kullandıkları giriş parametreleri de artırılmıştır. Tüm algoritmalar iki düğüm arasındaki bağlantı maliyetini kullanırken bunlara başka parametreleri de normalize ederek karar verme kabiliyeti güçlendirilmiştir. Tüm sonuçlar bir araya toplanarak grafiksel bir yoruma çevrilmiştir. Tüm algoritmalar aynı topoloji üzerinde çalışmakta olup yine aynı gecikme ve mesafe değerleriyle çalışmaktadır. Bu girdilere göre ürettikleri en kısa yol hesaplamaları grafik katmanında yorumlanarak görsele dökülmüştür.

Şubat 2020, 56 sayfa.

Anahtar kelimeler: Açık Artırma, Bellman Ford, Dijkstra, Floodlight, İkili Tırmanma, Topoloji, Yazılım Tanımlı Ağ, Yönlendirme

SUMMARY

M.Sc. THESIS

PERFORMING SEVERAL ROUTING ALGORITHMS AND FLOW CONTROL IN A SOFTWARE DEFINED NETWORK

Hilmi Tunahan ILHAN

İstanbul University-Cerrahpaşa

Institute of Graduate Studies in Sciences

Department of Computer Engineering

Supervisor: Assoc. Prof. Dr. Derya YILTAŞ KAPLAN

This thesis has been done by developing the network routing capability of a Software Defined Network (SDN) application by adding different algorithms. During the study, Floodlight was used as SDN, Intellij IDEA was used as software development IDE and Java 8 was used as software language. Newly added algorithms and the Dijkstra's algorithm currently used by Floodlight have been used to enrich routing capabilities. Two algorithms with Auction and Dual Ascent approaches have been developed from scratch and integrated into Floodlight along with Dijkstra. In addition, the input parameters used by all algorithms in decision making have been increased. While all algorithms use the cost of the connection between the two nodes, their ability to make decisions has been strengthened by normalizing another parameters to them. All the results were gathered and translated into a graphical interpretation. All algorithms worked on the same topology and with the same delay and distance values. The shortest path calculations, which were produced according to these inputs, were interpreted in the graphic layer and visualized.

February 2020, 56 pages.

Keywords: Auction, Bellman Ford, Dijkstra, Floodlight, Dual Ascent, Topology, Software Defined Network, Routing

1. GİRİŞ

Teknolojinin gün geçtikçe gelişmesinin ve hayatımıza daha fazla girmesinin sonucu olarak etrafımızı saran teknolojik cihazların sayısı çok artmıştır. Bu da insan cihaz ve cihazlar arası etkileşimin artmasına yol açmıştır. Bunun anlamı da her geçen gün birbirine bağlı cihazların artması demektir. Birbirine bağlı cihazlar demek bir topoloji demektir. Bu da yönetilmesi gereken, en hızlı şekilde sorunsuzca çalışması gereken bir ağ anlamına gelmektedir. İşte bu noktada hizmet kalitesini artırmak, veri iletişimini hızlandırmak, veri iletişim güvenliğini artırmak başlı başına yönetilmesi gereken bir iş demektir. Yazılım Tanımlı Ağ (YTA) uygulama kolaylığı, donanım bağımsızlığı, dinamik, esnek ve ölçeklenebilir ağ mimarisi sağlar. Bu nedenle, büyük ve karmaşık ağ yönetimine etkili bir çözüm sunar.

İletişim; veri paketlerinin belirli yollar üzerinden bir noktadan başlayarak başka nokta veya noktalara aktarılması sürecidir. Bu aktarım sürecinin her türlü testinin, kontrolünün, geliştirilmesinin ve paket akışının devamı YTA ile sağlanabilir. Buradan hareketle farklı yönlendirme algoritmalarının uygulaması ağ üzerinde gerçekleştirilmesi YTA ile mümkün olmuştur. Bu projede de YTA'nın bu kolaylığından faydalanılarak farklı algoritmaların uygulanması mümkün olmuştur.

YTA'nın bize sunduğu soyutlama imkânı sayesinde yazılım ile ağı yönetebilmek mümkün olmuştur. Bu yeteneği kullanarak algoritmaların karar verme parametrelerinde geliştirme yapılmıştır. Bu geliştirme karar aşamasında sadece bağlantı maliyetini değil, ağ üzerinden okunan ikinci bir parametrenin normalizasyon işleminden geçirilerek kullanılmasıdır.

Bu tezin amacı bir YTA kullanarak yeni yönlendirme yaklaşımları uygulanmış algoritmaları gerçekleştirmek ve bu algoritmaların o topolojinin o anlık durumunu göz önünde bulundurarak karar vermelerini sağlamaktır.

2. GENEL KISIMLAR

2.1. AĞ

Birbirleri ile elektriksel anlamda bağlantılı olan birden fazla bilgisayarın oluşturduğu bilgisayar grubuna ağ adı verilir. Buradaki bilgisayardan kasıt sadece bildiğimiz anlamdaki kişisel bilgisayarlar olmayıp herhangi bir ağa bağlanma yetisi olan herhangi bir makine olabilir. Bu bağlantı o gruptaki tüm bilgisayarlardan diğer tüm bilgisayarlara olan kablolar gibi düşünülebilir ama durum fiziksel olarak genelde böyle değildir. Bilgisayarlar birbirlerine direkt olarak bağlanmayıp yönlendirici cihazları vasıtası ile bağlanır. Tüm bu yönlendirici cihazlar da başka yönlendirici cihazlara bağlı olabilir. Bunlar küçük sokakların caddelere bağlanması, caddelerin de bulvarlara bağlanması gibi düşünülebilir. Az trafik taşıyan yönlendirici cihazlar daha fazla yük taşıyan daha büyük ve merkezi cihazlara yönlendirme yapabilir veya yönlendirme alabilir.

İnternete bağlı her bir bilgisayar büyük bir ağın bir parçasıdır. Bu bağlamda düşünüldüğünde evimizdeki bilgisayardan çıkan bir veri sokağımızdaki yönlendiriciden geçip mahallemizdeki merkez yönlendiriciye ve ardından bir kaç merkezi yönlendiriciden geçerek hedef sunucu makineye ulaşır. İster bilgisayarlar arası olsun ister bilgisayarlar ile yönlendirici cihazlar arası olsun tüm fiziki bağlantılar eski teknoloji basit bakır kablolar ile de olabilir, son derece gelişmiş ve çok hızlı bağlantı sunan fiber optik kablolar ile de olabilir. Bu süreçte giden veriler evimizin içinden bakır kablo veya fiber kablo ile başlayarak yolculuğuna başlar. Toplam yolculuk boyunca basit bakır kablolardan, gelişmiş fiber kablolara kadar çeşitli kablo tiplerinden geçer. Günümüz İstanbul'u düşünüldüğünde sokakların çoğunda eski ve genellikle en çok 16 megabit/saniye hız destekleyen bakır kablolar ile döşenmiştir. Bazı kablolar kıtalar arası da döşenmiştir. Avrupa kıtası ve Amerika kıtası arasında okyanus altından giden çok sayıda fiber optik kablo vardır. Bu bağlantılar da içinde bulunduğumuz ağın bir parçasıdır.

Bilgisayar ağlarını boyutlarına göre bir sınıflandırmaya tabi tutarsak eğer bunların bir kaç alt sınıfa ayrıldığını görürüz. Bunlar Local Area Network (LAN), Metropolitan Area Network (MAN), Wide Area Network (WAN), Body Area Network (BAN), Personal Area Network (PAN) olarak ayrıldığı görülür. Sırası ile açıklayacak olursak; LAN bağlantıları oda, bina, kampüs boyutunda olan ve bir kaç kilometreyi geçmeyen ağlardır. Örneğin bir işyeri veya

fabrika gibi yerlerde iş sunucu makinelerini, çalışan bilgisayarlarını ve yazıcı gibi cihazları içine alan ağlardır.

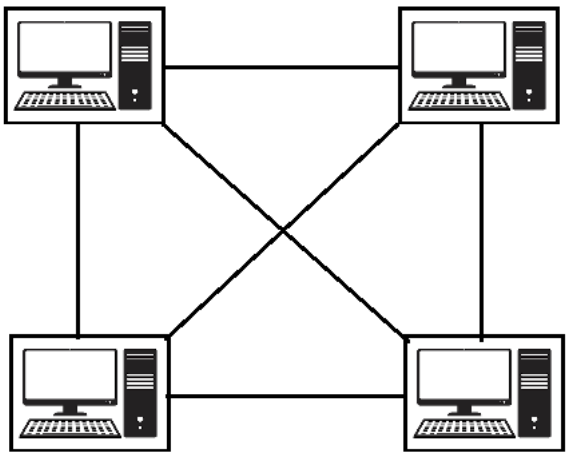
MAN bağlantıları şehir boyutunda ağlardır. Yani MAN'lar LAN ağın şehir çapı boyutunda olan türüdür. Bir MAN ağı ses ve veri iletebileceği gibi bir kablolu televizyon ağı da olabilir. Genel olarak bağlantı bir ya da iki kablodan sağlanır ve yayın türü iletim yapılır.

WAN bağlantıları ise ülke veya kıta çapında ağlardır. Daha da detaylı açıklanacak olursa, birçok LAN ağın ve ana bilgisayarların haberleşme alt ağları(subnet) sayesinde birleşerek ortaya getirdiği ülke veya kıta çapında ağdır. Alt ağ iki ana öğeden oluşur. İletim hatları ve yönlendiriciler. BAN ise vücuda takılan sağlık cihazları gibi cihazları kapsamaktadır. PAN ise kişisel alanımızda kullandığımız cihazları kapsamaktadır. Örneğin, bluetooth türü kısa mesafe cihazları bu tanıma iyi bir örnek olacaktır.

Ayrıca bir türden daha bahsedebiliriz. O da ağlar arası bağlantıdır(internet works). İnternet gibi birden çok ağın bağlantı kurmasıyla oluşan ve gezegen çapında olan ağlardır.[1]

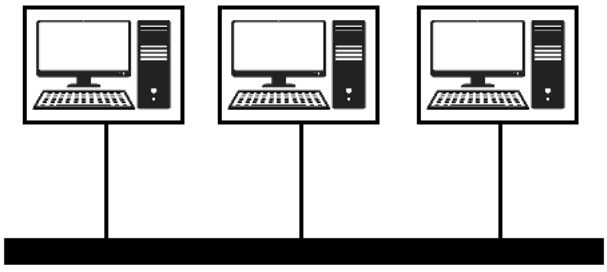
2.1.1. Ağ Bağlantı Türleri(Topoloji)

Bilgisayar ağları, birkaç ana bilgisayarın kendi aralarında bilgi alışverişinde bulunmasına izin vermek için kullanılır. Herhangi bir ana bilgisayarın ağdaki diğer herhangi bir ana bilgisayara mesaj göndermesine izin vermek için en kolay çözümün tam örgü ağ (full-mesh) adı verilen ağ türü olduğu söylenmektedir. Her ana bilgisayar çifti arasında doğrudan ve özel bir bağlantıyla bunları bu tam örgülü ağ olarak organize etmenin avantajları vardır. Böyle bir fiziksel topoloji bazen, özellikle az sayıda ana bilgisayar için yüksek performans ve yüksek fazlalık gerektiğinde kullanılır.



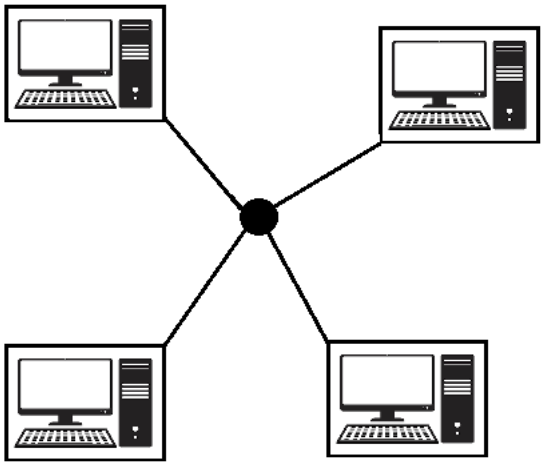
Şekil 2.1: Tam Örgü Ağ(Full-Mesh) Topolojisi [2]

Bilgisayarları birbirlerine bağlamak için kullanılan ikinci olası fiziksel organizasyon ise veri yolu yani "bus"tır. Bir veri yolu ağında, tüm ana bilgisayarlar ortak bir ortama, genellikle tek bir arabirim üzerinden bir kabloya bağlanır. Bir ana bilgisayar veri yoluna bir elektrik sinyali gönderdiğinde, veriyoluna bağlı tüm ana bilgisayarlar tarafından sinyal alınır. Veri yolu tabanlı ağların bir dezavantajı, ana iletim hattı kesilirse, ağın iki izole ağa bölünmesidir. Bu nedenle, bu tür ağların, özellikle kablo uzun olduğunda ve kablonun kopabileceği birçok yer olduğunda işletilmesi ve bakımı zor olmaktadır. Böyle bir veri yolu tabanlı topoloji, erken Ethernet ağlarında kullanılmıştır.



Şekil 2.2: Bus Topolojisi [2]

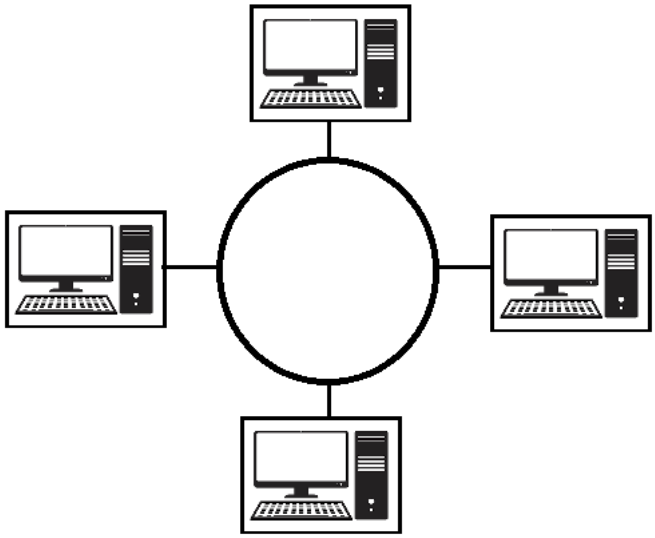
Bilgisayar ağının üçüncü organizasyonu da yıldız topolojisidir. Bu tür topolojilerde, bilgisayarlar tek bir fiziksel bağlantı noktasına sahiptir. Her bilgisayar ve yıldız topolojinin merkezindeki bilgisayar arasında bir fiziksel bağlantı vardır. Yıldız topolojinin ortasındaki düğüm farklı tipte cihazlardan olabilir. Elektrik sinyalini yükselten basit bir donanım ya da ağ üzerinden iletilen mesajların formatını anlayan aktif bir donanım olabilir. Bu topolojide merkezi düğümün olası bir arızası ağın genel bir arızası haline gelecektir. Bununla birlikte; bir fiziksel bağlantı koparsa eğer yalnızca bir düğüm ağdan çıkarılır. Genel bir soruna dönüşmeyecektir. Yıldız topoloji ağların kullanımı ve bakımı, veri yolu şeklindeki ağlardan daha kolaydır. Bir Web ara yüzünden veya konsol benzeri bir bağlantı aracılığıyla yönetilen merkezi bilgisayar ağın yönetilmesi konusunda etkin bir şekilde kullanılabilir. Örneğin, merkeze bağlı diğer aygıtları etkinleştirme veya devre dışı bırakma, kullanım istatistikleri toplama gibi konularda kullanılabilir.



Şekil 2.3: Yıldız(Star) Topolojisi [2]

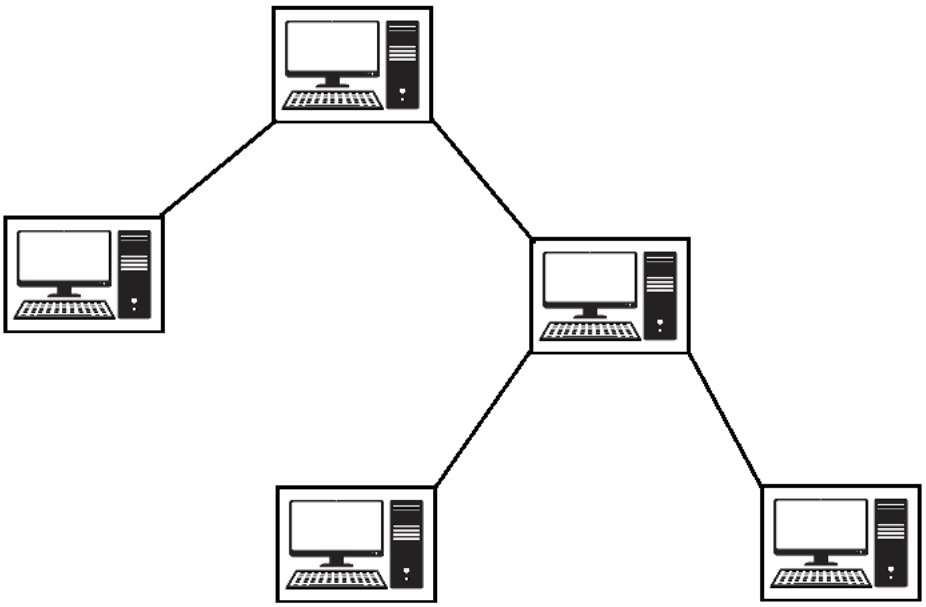
Bir ağın dördüncü bir fiziksel organizasyonu, halka topolojisidir. Veri yolu organizasyonu gibi, her ana bilgisayar da halkayı birbirine bağlayan tek bir fiziksel ara yüze sahiptir. Halkadaki bir ana bilgisayar tarafından gönderilen herhangi bir sinyal, halkadaki tüm ana bilgisayarlar tarafından alınacaktır. Devamlılık bakış açısından tek bir halka en iyi çözüm değildir, çünkü sinyal halka üzerinde sadece bir yönde hareket eder; bu nedenle, halkayı oluşturan

bağlantılardan biri kesilirse, tüm ağ başarısız olur. Uygulamada, bu tür halka topolojileri LAN'larda kullanılmıştır. Ancak şimdi genellikle yıldız şekilli ağlar ile değiştirilmektedir. Büyükşehir ağlarında, halkalar genellikle birden fazla konumu birbirine bağlamak için kullanılır. Bu durumda, yedeklilik için genellikle farklı kablolardan oluşan iki paralel bağlantı kullanılır. Bu tür bir ikili iletişim başarısız olduğunda tüm trafik hızlı bir şekilde diğer hat üzerinden geçebilir.



Şekil 2.4: Halka(Ring) Topolojisi [2]

Bir ağın beşinci fiziksel organizasyonu ağaçtır. Bu tür ağlar, genellikle çok sayıda müşterinin çok uygun maliyetli bir şekilde bağlanması gerektiğinde kullanılır. Kablolu TV ağları genellikle ağaç olarak düzenlenir.



Şekil 2.5: Ağaç (Tree) Topolojisi [2]

2.1.2. Ağ Cihazları

Bu aşağıda sıralanan cihazların her biri bir ağda belirli bir rolü yerine getirir. Ancak, yalnızca en büyük ve en karmaşık ortamlar hepsini kullanır. Ağ aygıtları anlatımıza, bugün kullanılan en basit ve en yaygın ağ aygıtı olan hub'ı kullanarak başlayacağız.

2.1.2.1.Göbek(Hub)

Göbekler basit ağ cihazlarıdır ve basitlikleri düşük maliyetlerine yansıtılmaktadır. Dört veya beş bağlantı noktasına sahip küçük göbekler (genellikle çalışma grubu hub'ı olarak adlandırılır) az maliyetlidir. Küçük bir ağ oluşturmak için gereken her şeyi sağlarlar. Daha fazla port'a (giriş-çıkış) sahip göbek cihazı daha fazla kapasite gerektiren ağlar için kullanılabilir.

Bir göbek cihaznın temel işlevi bağlı cihazlardan birinden veri almak ve onu göbek üzerindeki diğer tüm bağlantı noktalarına iletmektir. Bu işlem yöntemi verimsizdir çünkü çoğu durumda veriler bağlı cihazlardan yalnızca biri için gönderilmektedir.

Göbekler aktif veya pasif olarak adlandırılır. Aktif göbek bir güç kaynağı gerektirir ve gönderilen sinyalleri hedeflerine yönlendirirken sinyal güçlendirmesi yapar. Küçük çalışma grubu göbekleri normalde harici bir güç adaptörü kullanır. Daha büyük ünitelerde güç kaynağı yerleşiktir. Günümüzde yalnızca eski ağlarda görülen pasif göbekler ise güce ihtiyaç duymaz ve veri sinyalini yeniden oluşturmazlar.[3]

2.1.2.2.Anahtar(Switch)

Görünürde bir anahtar bir göbek cihazına çok benzer. Benzer görünümlerine rağmen, anahtarlar göbeklerden çok daha verimlidir ve günümüzün ağ ortamları için çok daha fazla tercih edilmektedir. Bir göbekte olduğu gibi, bilgisayarlar bir kabloyla(LAN kablosu) bir anahtara bağlanır. Birden fazla anahtar genellikle daha büyük ağlar oluşturmak için birbirine bağlanır. Görünümdeki benzerliklerine ve bilgisayarlarla aynı fiziksel bağlantılarına rağmen, anahtarlar göbeklere kıyasla önemli operasyonel avantajlar sunar.

Bir göbek verilerin hangi hedef makine için gönderildiğine bakmaksızın verileri tüm bağlantı noktalarına iletir. Bu yönlendirme yetersiz ve verimsizdir. Anahtarlar verileri tüm bağlantı noktalarına iletmek yerine verileri yalnızca hedef sistemin bağlı olduğu bağlantı noktasına iletirler.

Doğru bağlantı noktasını belirlemek için bağlı cihazların Medya Erişim Kontrolü (MAC) adreslerine bakar. Bir MAC adresi benzersiz bir numaradır. Anahtar verileri yalnızca verilerin adreslendiği sisteme ileterek tüm ağ bağlantısındaki trafik miktarını önemli ölçüde azaltır. Aslında anahtarlar kelimenin tam anlamıyla verileri portlar arasında yönlendirir.[3]

2.1.2.3.Köprü(Bridge)

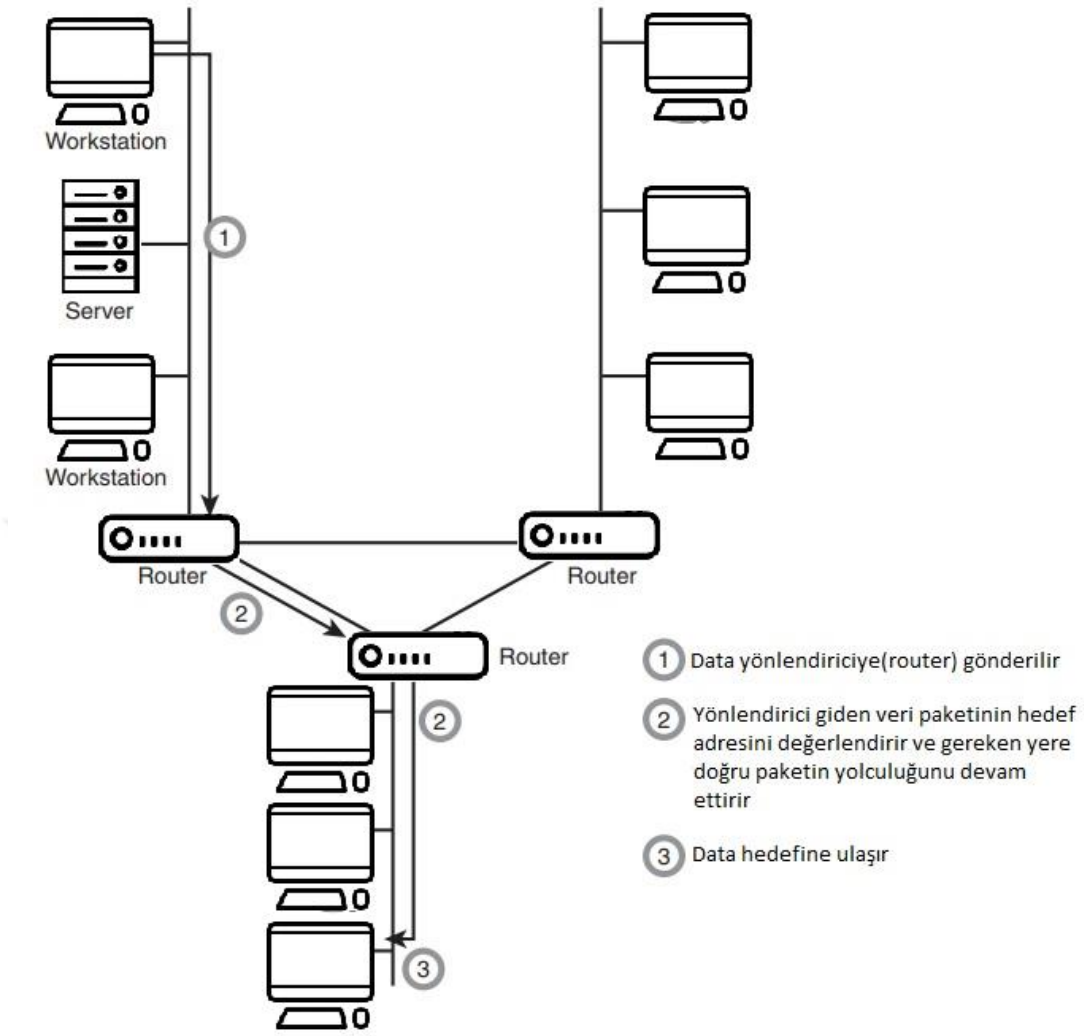
Köprüler; ağları bağlayan ağ aygıtlarıdır. Büyük ağlarda olan her alt ağdaki trafik miktarını azaltmak veya güvenlik nedeniyle ağları alt ağlara bölmek gerekir. Köprü ağ bölündükten sonra iki alt ağı birbirine bağlar ve aralarındaki trafik akışını yönetir. Günümüzde ağ anahtarları köprüleri büyük ölçüde değiştirmiştir. Bir köprü, bir veri çerçevesine yazılan hedef MAC adresini temel alır ve verileri engelleyerek veya ileterek yönetir. Köprü, hedef adresin verinin alındığından başka bir ağda olduğuna karar vermişse verileri bağlı olduğu diğer ağlara iletebilir. Adres köprünün diğer tarafında değilse, verilerin geçmesi engellenir. Köprüler ağ trafiğini

“dinleyerek” ve trafiğin geldiği ağı kaydederek bağlı ağlardaki cihazların MAC adreslerini “öğrenir”.[3]

2.1.2.4.Yönlendiriciler(Routers)

Yönlendiriciler, küçük bir ofiste çalışan ve bir internet servis sağlayıcısına (ISS) bağlanmaya uğraşan bir çalışandan tutun da farklı çalışma sahalarındaki sistemlerini bir arada çalıştırmak için çaba sarf eden büyük bir şirkete kadar her ortam için çok önemlidir. Yönlendiriciler, internet kavramını mümkün kılar. Yönlendiriciler, kelimenin tam anlamıyla ağın etrafındaki verileri yönlendiren ağ aygıtlarıdır. Yönlendiriciler, geldiği sırada verileri inceleyerek verinin hedef adresini belirleyebilirler; daha sonra tanımlanmış rota tablolarını kullanarak verinin yoluna devam etmesi için en iyi yolu belirlerler.

Verilerin hedefini belirlemek için donanım tarafından yapılandırılmış MAC adresini kullanan köprüler ve anahtarların aksine, yönlendiriciler karar vermek için yazılım tarafından yapılandırılmış ağ adresini kullanırlar. Bu yaklaşım, yönlendiricileri köprülerden veya anahtarlardan daha işlevsel hale getirir. Öte yandan çalışır halde olması için daha çok karar vermek zorunda olmaları onları daha karmaşık hale getirir. Şekil 2.6’da temel olarak bir yönlendiricinin nasıl çalıştığı gösterilmiştir. Bir yönlendirici için temel gereksinim, en az iki ağ arabirimine sahip olmasıdır. Eğer bunlar LAN bağlantıları ise, yönlendirici bilgileri iki LAN alt ağı arasında yönlendirebilir. Ama daha yaygın olan kullanım ise WAN bağlantıları arasında bağlantı sağlamak için bir yönlendirici kullanılmasıdır.[2]



Şekil 2.6: Yönlendirici Çalışma Şekli[3]

Bir yönlendirici özel bir donanım aygıtı olabilir veya içindeki gömülü bir yazılım tarafından kontrol ediliyor olabilir. Tüm yaygın ağ işletim sistemleri, işlevselliklerinin bir parçası olarak yönlendirici görevini yapma becerisi sunarlar. Özel donanım yönlendiricileri, sunucu tabanlı çözümlerden daha yüksek performans seviyeleri sunar. Ancak sınırlı sayıda özellik sunmak gibi bir dezavantajı vardır. Ancak özel bir donanım cihazı kullanmanın avantaı genellikle bu faktörden ağır basar[2] [3]

2.1.3. 5 Katman Referans Modeli

Tüm bu ağ iletişimi TCP/IP(*Transmission Control Protocol/Internet Protocol*) protokolü ile sağlanır. Bu protokolün mimarisini en yaygın şekliyle 5 katman referans modeliyle açıklayabiliriz.

Hiyerarşinin basit tarafından başladığımız durumda ilk katman Fiziksel katmandır. İki iletişim cihazı fiziksel bir ortam yoluyla bağlanır. Bu fiziksel ortam, doğrudan bağlı iki cihaz arasında bir elektrik veya optik sinyali aktarmak için kullanılır. Uygulamada çeşitli fiziksel ortam türleri kullanılır: Elektrik kablosu ve fiber kablo gibi.

Veri Bağı Katmanı, alttaki fiziksel katman tarafından sağlanan hizmet üzerine kurulur. Veri Bağı katmanı, fiziksel katman yoluyla doğrudan bağlanan iki ana bilgisayarın bilgi alışverişine izin verir. Veri Bağı katmanındaki iki varlık arasında değiş tokuş edilen bilgiler çerçeve adı verilen veri parçaları şeklinde aktarılır. Bir çerçeve sınırlı bir bit dizisidir. Çerçevelerinin hedef ve kaynağını belirtmek için MAC adresini kullanır. Bazı Veri Bağı katmanları değişken uzunlukta çerçeveler kullanırken diğerleri yalnızca sabit uzunlukta çerçeveler kullanır. Bazı Veri Bağı katmanları bağlantı yönelimli hizmet sağlarken, diğerleri bağlantısız hizmet sağlar. Ayrıca türlerine göre bazı Veri Bağı katmanları güvenilir teslimat sağlarken, diğerleri bilgilerin doğru teslimatını garanti etmez[2].

Ağ Katmanı, veri bağlantısı katmanının üzerine inşa edilmiştir. Ağ katmanı ağ üzerinde gezinen veri çerçevelerinin oluşturulmasından sorumludur. Bu katman veri paketlerinin hedefini ve kaynağını belirtmek için IP adreslerini kullanır.

Aktarım Katmanı, farklı ana bilgisayarlarda çalışan uygulamalar arasında bağlantı kurar. Güvenilir bağlantılar için TCP ve hızlı bağlantılar için UDP kullanır. Bağlantı noktası numaralarını atayarak yukarıdaki uygulamalarda çalışan işlemleri izler ve TCP / IP ağına erişmek için Ağ katmanını kullanır.

Mimarimizin üst katmanı Uygulama katmanıdır. Bu katman, uygulamalar için gerekli olan tüm mekanizmaları ve veri yapılarını içerir. Bu uygulamalara örnek olarak e-posta istemcileri ve web tarayıcıları verilebilir. Bu uygulamalar, uzak ana bilgisayarlara bağlanma istekleri göndermek için Aktarım Katmanını kullanır. Uygulama katmanının iki birimi arasında değiş tokuş edilen verileri belirtmek için Uygulama Veri Birimi (ADU – Application Data Unit) kullanılır.[2]

2.2. YTA

Bilgisayar ağlarında yönlendiriciler, anahtarlar ve güvenlik duvarları(firewall) gibi birçok ağ cihazı vardır. Bu cihazların çalışmasından farklı protokoller sorumludur. Gelişen teknoloji ile birlikte veri merkezleri hızla büyümekte ve bilgisayar ağ cihazlarının sayısı hızla artmaktadır. Bu nedenle, ağ yönetimi daha karmaşık ve zor hale gelmiştir.

Geleneksel ağ yönetimi yaklaşımı bu durumda yetersiz kalmıştır. Bu yaklaşımda esnekliğe çok müsait olmayan yöntemler kullanılmaktaydı. Örnek olarak buna yönlendiricilerdeki yönlendirme tabloları verilebilir. Bu nedenle, daha iyi bir ağ yaklaşımı ve yeni yöntemlere ihtiyaç doğmuştur. Bu noktada YTA bir çözüm olarak ortaya atıldı. YTA, bir ağın kontrol mekanizmasının ağın diğer tüm veri iletim mekanizmasından ayrılması anlamına gelmektedir. YTA uygulama kolaylığı, donanım bağımsızlığı, dinamik, esnek ve ölçeklenebilir ağ mimarisi gibi özellikleri kullanıcılara sunar. Bunun sonucunda büyük ve karmaşık ağ yönetimine etkili bir çözüm sunar [4].

Teknolojinin hızla değiştiği çağdayız ve bu durum internete bağlantı oranında ve internet adı verilen büyük ağa bağlanan makine sayısında hızlı bir artışa sebebiyet vermektedir. İnternet bağlantısı arttıkça, sürekli veri üretilir ve bu veriler büyür. Sonuç, büyük, karmaşık ve düzensiz bilgi anlamına gelen büyük verilerdir. Büyük veriler geleneksel yollarla işlenemez, yönetilemez ve saklanamaz. Anlamlı ve değerli bilgiler içerdiğinden büyük veriler işlenmelidir. Büyük verileri işlemek için, daha geniş bant genişliğine sahip bir işlem gereklidir. İşte YTA'ların önemi bu noktada öne çıkmaktadır.

2.2.1. YTA Temel Prensibi

YTA şu beş temel prensip üzerine çalışmaktadır. Bunlar; düzlem ayrılığı, basitleştirilmiş cihaz, merkezileştirilmiş kontrol, ağ otomasyonu ve sanallaştırılması ve açıklıktır.

2.2.1.1.Düzlem Ayrılığı

YTA'nın ilk temel özelliği, veri iletim ve kontrol mekanizmalarının ayrılmasıdır. MAC adresi, IP adresi ve VLAN kimliği gibi özelliklere sahip olan paketlerin nasıl yönlendirileceği gibi işlevsellikler yönlendirme düzleminde bulunur. Yönlendirme düzlemi tarafından gerçekleştirilen temel eylemler gelen paketlerin dağıtılması şeklinde tarif edilebilir. Gelen bir paketi iletebilir, düşürebilir, tüketebilir veya çoğaltabilir. Örneğin bir servis kalitesi hız

sınırlama özelliğinden kaynaklanan özel bir filtreleme nedeniyle bir paket düşürülebilir. Kontrol veya yönetim düzlemi tarafından işlem yapılmasını gerektiren özel durum paketleri alınır ve uygun düzleme iletilir.

Veri iletim düzlemini programlamak için kullanılan protokoller, mantık ve algoritmalar kontrol düzleminde bulunur. Bu protokoller ve algoritmalar çalışırken ağ hakkında bilgiye ihtiyaç duyar. Kontrol düzlemi, veri iletim düzlemindeki iletme tablolarının ve mantığının nasıl programlanması veya yapılandırılması gerektiğini belirler.[5]

Veri iletim düzlemi ise bildiğimiz şekliyle göbeklerden, anahtarlardan, köprülerden oluşan bir ağ olabileceği gibi tüm bu fiziki donanımları simüle eden bir uygulama da olabilir. Bu özelliği YTA'lara esneklik katmaktadır.

2.2.1.2.Basitleştirilmiş Cihaz ve Merkezi Kontrol

Yönlendirme ve kontrol düzlemlerinin ayrılması fikrine dayanan bir sonraki özellik, yönetim ve kontrol yazılımını çalıştıran merkezi bir sistem tarafından kontrol edilen cihazların basitleştirilmesidir. Aygıtta çalışan ve aygıtın bağımsız davranmasını sağlayan yüz binlerce satırlık karmaşık kontrol düzlemi yazılımı kullanılmaz. Bunun yerine bu kontrol düzlemi yazılımı merkezi bir denetleyiciye yerleştirilir.

Bu yazılım tabanlı denetleyici daha üst düzey protokoller kullanarak ağı yönetir. Daha sonra kontrolör, gelen paketlerle nasıl başa çıkılacağı konusunda hızlı kararlar verebilmeleri için uygun olduğunda basitleştirilmiş cihazlara ilkel talimatlar sağlar. Kontrol düzlemi yazılımı ise bu yazılımın veri paketleri ile ilgili verdiği kararlara dayanarak komutları daha alt seviye bir komut setine çevirir.[5]

2.2.1.3.Ağ Otomasyonu ve Sanallaştırma

YTA'nın temelini oluşturan üç temel soyutlaştırmadan söz edilebilir. Bu anlamda soyutlama, yapılacak bir işin detaylarını bilmeye gerek kalmadan o işin yapılmasını sağlamaktır. YTA şu üç soyutlaştırmadan meydana geliyor diyebiliriz. Bunlar dağıtık durum, iletim ve yapılandırma soyutlamalarıdır. Bu üç durum günümüzde ağ yönetim düzleminin karşılaştığı karmaşık problemlerin soyutlaştırılmasıyla meydana gelen kavramlardır. Günümüzde popüler olarak kullanılan Java programlama dili olmasına rağmen daha önceleri programlama düşük seviyeli diller ile yapılmaktaydı. Daha sonra C gibi orta seviye diller ortaya çıkmıştır. Daha sonra da

Java gibi gelişmiş diller ortaya çıkmıştır ve programcıların istediği şekilde aksiyon olmasına olanak sağlamışlardır. YTA'nın ağ kontrolü sorunu için benzer bir doğal evrim olduğunu iddia edilmektedir.

Dağıtık durum soyutlaması, ağ programcısının her biri farklı özel duruma sahip olan ağların farklılıklarından soyutlar. Böylece programcı her bir ağın kendi özel durumunu bilmek zorunda kalmaz. Aradaki soyutlama katmanı programcıyı bu tarz detayları bilmekten kurtarır. Yönlendirme soyutlaması, programcının donanım üreticisine özgü bilgi olmadan gerekli yönlendirme kararları üretebilmesine olanak sağlar. Yapılandırma soyutlaması, fiziksel ağın gönderilen komutları nasıl uygulayacağının ayrıntılarında kaybolmadan tüm ağın istenen hedeflerini uygulamasını sağlayan soyutlamadır.[5]

2.2.1.4.Açıklık ve Anlaşılabilir Olmak

Açık bir YTA'nın bir özelliği, ara yüzlerinin standart kalması, iyi belgelenmesi ve kişiye özel olmamasıdır. Sunulan servisler servisleri tüketen yazılıma çeşitli kontrol seçeneklerini denemek ve kontrol etmek için yeterli kontrol sağlamalıdır. Araştırma kurumları ve girişimciler, yeni fikirleri kolayca denemek ve test etmek için bu yetenekten yararlanabilir. Bu nedenle, ağ teknolojisinin geliştirilmesi ve devreye alınma hızı daha iyi noktadadır. Çok daha fazla sayıda birey ve kuruluş günümüzün ağ sorunlarının çözümüne katkıda bulunabildiklerinden, ağların yapısı ve işleyişinde daha iyi ve daha hızlı teknolojik ilerlemeye yol açmaktadır. Bu açık ara yüzlerin varlığı SDN ile ilgili açık kaynak projelerini de teşvik eder. [5]

2.2.1.5.Bilinen YTA Kontrolör Uygulamaları

Bugün piyasada bir dizi farklı YTA ürünü bulunmaktadır. Hem açık kaynaklı hem de ticari ve tescilli YTA denetleyicileri vardır. Açık kaynak YTA denetleyicisi NOX gibi temel C dili ile yazılmış denetleyicilerden başlayıp Beacon veya Floodlight gibi Java tabanlı sürümlere kadar birçok biçimde gelir. Trema adında Ruby tabanlı bir kontrolör bile mevcuttur. Bu deneytleyici ürünler farklı formlarda da hizmet vermektedir. Http protokolünü kullanarak REST protokolü ile hizmet verenlerin yansıra Python API olarak hizmet verenleri de vardır. OpenDaylight adlı açık kaynak denetleyicisi bir satıcı şirket grubu tarafından oluşturulmuştur. Bunun yanında diğer üreticilerin tek başlarına piyasaya sürdüğü ürünler de bulunmaktadır. NEC, IBM ve HP gibi satıcılar, öncelikle OpenFlow uygulamaları olan denetleyiciler sunar. Diğer üreticilerin

çoğu daha kısıtlı düzeyde OpenFlow desteği içeren ve satıcıya özgü standartları da bulunan YTA denetleyicileri sunarlar[6].

2.3. DIJKSTRA ALGORITMASI

2.3.1. Algoritmanın Açıklaması

Dijkstra algoritması sadece yazılım dünyasında değil Matematik dünyasında da adı geçen bir algoritmadır. Dijkstra algoritması adını kurucusundan almaktadır. 1959 yılında Dijkstra tarafından bulunmuştur. Fakat aynı zamanda farklı yerlerde birbirlerinden farklı şekilde de keşfedilmiştir.[7]

Çalışma şekli ise şöyle özetlenebilir. Dijkstra algoritmasının ana fikri, başlangıç düğümü ile olası tüm hedefler arasındaki daha uzun yolları sürekli olarak ortadan kaldırıp kısa yolları öne çıkarmaktır. Süreci takip etmek için kalıcı ve geçici iki farklı düğüm listesine ihtiyacımız var. Kalıcı düğümler, kaynaktan asgari mesafeye sahip olan düğümlerdir. Geçici düğüm listesindeki düğümler ise kaynaktan erişebileceğimiz düğümlerdir ancak başlangıç düğümünden o düğüme olan minimum mesafeyi henüz nihai olarak bilmiyoruz demektir.

Bu algoritmayı ilkleme ve sonrasında değerlendirme evresi olmak üzere toplamda iki aşamalı olarak düşünebiliriz.

Grafikteki tüm yolları keşfetmeye başlamadan önce kaynak dışında tüm düğümleri sonsuz bir mesafeye eşitlememiz gerekir. Daha sonra başlatma işleminin bir parçası olarak, başlangıç düğümüne 0 değerini atamamız gerekir (başlangıç düğümü ile başlangıç düğümü arasındaki mesafe sıfırdır). Bu nedenle, grafiğin geri kalanındaki her düğüm bir öncül düğüm ve bir mesafe olarak ele alınacaktır. Başlatma işlemini bitirmek için geçici düğümlere başlangıç düğümünü eklememiz gerekir. Bu, değerlendirme adımında ilk önce seçilmek üzere ayarlanır.

Başlangıç düğümünü geçici listeye ekledikten sonra başlangıç düğümünün komşusu olan düğümleri dolaşarak ulaşım maliyetlerini kaydediyoruz. Bu güncelleme işleminden sonra güncellenen düğümlerin komşularını da güncelliyoruz. Tüm düğümler güncellenene kadar ve şekil üzerinde yeni bir güncelleme kalmayana kadar bu işlem devam eder.

2.3.2. Algoritma Sözde Kodu(Pseudocode)

Dijkstra yöntemini daha iyi ele almak için sözde kodunu aşağıya sıralayalım.

- Başlangıç düğümünün mesafe değerini sıfıra ayarlayın.
- Diğer tüm mesafeleri sonsuz bir değere ayarlayın.
- Başlangıç noktasını geçici düğümler listesine ekleyin.
- Geçici düğümlerin listesi boş olduğu sürece:
  - Geçici düğümlerden kaynaktan en düşük mesafeye sahip olan bir düğümü değerlendirme düğümü olarak seçin.
  - Her değerlendirmede en düşük mesafeyi koruyarak değerlendirme düğümünün komşularının başlangıç düğümüne olan mesafesi güncellenir.
  - Sonra bu komşuları geçici düğüm listesine ekleyin.
  - Sonra da değerlendirme düğümünü kalıcı listeye taşıyın.

2.4. AÇIK ARTIRMA(AUCTION) ALGORİTMASI

2.4.1. Algoritmanın Açıklaması

Açık Artırma algoritmasına bu isim Naif Açık Artırma(Naive Auction) algoritması ile yakından ilişkili olduğu için verilmiştir. Açık Artırma algoritması 1979 yılında klasik atama problemleri için Dimitri P. Bertsekas tarafından önerilen bir algoritma ile ortaya çıkmıştır. Bu algoritma başlangıç noktasında başlayan tek bir yolu geliştirerek hedefe ulaşmaya çalışır. Başlangıç düğümünden hedefe kadar her yinelemede yol, yeni bir düğüm eklenerek genişletilir veya yol dediğimiz düğüm listesinin en son eklenen düğümünü silerek daraltılır. Eğer ki zincire son eklenen düğüm hedef düğüm ise algoritma çalışmasını sonlandırır. [8]

Algoritmanın nasıl çalıştığının genel olarak bir izlenimini elde etmek için bir örnek üzerinden gidilebilir. Bir hedefe ulaşmaya çalışan, grafik benzeri bir labirentte hareket eden bir fareyi düşünün. Fare, labirentin yollarını ileri geri giderek ve geri dönerek tarar. Fare bir düğümden her geri dönüşünde bu geri döndüğü yolu kaydeder. Gelecekte bu düğümü tekrar gözden geçirmesi gerekirse bu bilgiyi kullanacaktır. Düğümün arzu edilebilirlik ölçüsü diğer düğümlerinkinden üstün gelirse bu yoldan ikinci kez geçerek yoluna devam edebilir. Algoritma, basit veri yapılarını kullanarak bu arama işlemini etkili bir şekilde taklit eder.[7]

Algoritma adım adım kendine eklemeler yaparak final düğüme erişmeye çalışır. Herhangi bir *n* numaralı düğümde ise başlangıçtan *n* adımına kadar bağlantıları tek tek çıkararak ya da

ekleyerek gelmiştir. Algoritma kısaca iki temel mantık üzerine çalışmaktadır: Silme ve ekleme. Mevcut duruma göre bir "eğer" mantık operatörü yürütülür. Bu mantık sonucunda eğer karar silme ise, algoritma çözüm yoluna eklediği son düğümü kaldırır. Eğer mantık operatörünün sonucunda ekleme çıkar ise mantığa göre seçtiği komşu düğümü çözüm yoluna ekler. Algoritmanın çalışma mantığının temel bir veriyapısı vardır. O da küçük harf ile gösterilen *p* harfidir. Bu *p*, topolojideki düğüm sayısı uzunluğundaki bir tam sayı dizisi olarak düşünülebilir. Başlangıç noktasından itibaren son düğüme kadar numara verip, bu numaraların tam sayı dizisindeki karşılığı ise o düğümün *p* vektöründeki değeri olur. Örnek olarak p5 gibi bir değer olabilir. Açık Artırma mantığının en temel noktası burasıdır. Mevcut işleme konu olan düğümün *x*-1 numaralı düğüm olduğunu varsayalım. Potansiyel X numaralı düğüm ise 3 tane olsun. Bu 3 düğüm *x*-1 numaralı düğüme bağlantısı olan 3 komşu düğüm oluyor. Her bir potansiyel hamle için şu hesaplanır. Cost(*X-1 , X*) + *pX* (vektördeki değeri). Bu iki değerin toplanma işi gidilme ihtimali olan tüm düğümler için yapılır. Bu sonuçlardan en küçüğü alınır. Eğer ki bu değerlerden en küçüğü(*cX+px*) son mevcut düğüm olan *X-1* in vektördeki değerinden büyük ise hiç bir düğüme hamle yapılmaz. Aksine *X-1* numaralı düğüm de çözüm kümesinden çıkarılır. Eğer çalışmaya yeni başlanmışsa ve *X-1* düğümü aslında başlangıç düğümü ise çözüm kümesinden silinmez. Diğer tüm durumlarda silinir(contradiction). Öte yandan eğer bu minimum toplama değeri(*cX+pX*) *X-1* numaralı düğümün vektördeki değerinden küçük ise bu *X* düğümü çözüm kümesine katılır (extension). Dikkat edilmesi gereken bir nokta da şudur. Sadece silinme durumunda *X-1* numaralı düğümün vektördeki değeri güncellenir. Yeni değer ise *cX+pX* değeri olur. Yani son düğüm silinip geri çekilirken, o düğümünden sonraki gidilecek düğümün maliyeteti çekilen düğümün vektörüne eşitlenir[9][10]

2.4.2. Algoritma Sözde Kodu

- Başlangıç düğümü ile bitiş düğümü aynı ise çalışmayı durdur.
- CurrentNode <- başlangıç düğümü
- solution <- CurrentNode
- Pvector <- düğüm sayısı büyüklüğünde dizi ve başlangıçta tüm elemanlar 0
- nextLink <- iki düğüm arasındaki bağlantıyı simgeleyen bir değişken.
- NextLink değerinin hedef ucu hedef düğüm olana kadar şu işlemler tekrarlanır;
  - linksCurrentNode <- Mevcut düğümden çıkan bağlantıların listesini ata.
  - LinksCurrentNode listesindeki tüm düğümler için şu kontrolü uygula
    - vectorValue <- bu bağlantının vardığı düğümün Pvector değeri
    - linkValue <- bu bağlantının bağlantı maliyeti

- topValue <- [linkValue + vectorValue]
- nextLink <- topValue değerini sağlayan bağlantı

◦ currentVectorValue <- currentNode değişkeninin p vektöründeki değeri
◦ IF currentVectorValue < topValue
then: contradictPath
else : extendPath
◦ contradictPath -> çözüm kümesinden son düğümü kaldır
◦ extendPath -> çözüm kümesine nextLink değerinin vardığı düğümü ekle [8]

2.5. İKİLİ TIRMANMA(DUAL ASCENT) ALGORİTMASI

2.5.1. Algoritmanın Açıklaması

İkili Tırmanma yaklaşımını kullanarak ortaya çıkan algoritmalar arasından dağıtık sürüm çeşidini 1984 yılında Ricard T. Wong ortaya atmıştır.[11] Bu tezde Richard T. Wong'un makalesinden referans alarak bir sonraki bölümde açıklanacak olan uygulamanın geliştirilmesine başlanmıştır.

İkili Tırmanma, bir primal-dual algoritma örneğidir [12]. Doğrusal bir program formülasyonunun duali üzerinden çalışmak olarak yorumlanabilir[13].

İkili Tırmanma algoritması kaynak düğümden başlayarak iteratif bir şekilde tekrar ederek çalışır. Algoritmanın çalışmasını özetlerken kullanılan birkaç veri yapısından bahsetmek gerekir. Öncelikle "*rootComponent*" ve "*graph*" adı verilen ve topolojideki tüm düğümlerin bir alt kümesi olarak düşünülebilecek iki düğüm listesi vardır. Her bir iterasyonda bu "*graph*" düğüm listesini genişletiyoruz. Eğer ki bir iterasyonda hedef düğüm bu "graph" düğüm listesine dahil olursa algoritma başlangıç ile hedef düğüm arasında, içinde en kısa yolu da içeren bir yol kümesini bulmuş olur. Yani kaynağı kaynak düğüm olan ve düğümlerinin biri de hedef düğüm olan bir ağaç yapısı şeklinde düğüm ağacı ortaya çıkar.

İkinci olarak indirgenmiş maliyet listesinin tutulmasıdır. İndirgenmiş maliyet listesi ilk değer olarak topolojideki tüm bağlantıların maliyetlerinin tutulduğu bir listedir. Algoritmanın çalışması sırasındaki her bir iterasyonda öncelikle "*rootComponent*" isimli düğüm listesi "*graph*" düğüm listesinin o anlık durumundan yola çıkarak bulunur. Ardından "*rootComponent*" düğüm listesinde bulunan düğümlere göre bağlantı maliyetlerinde güncelleme yapılır. Yani, bu "*rootComponent*" listesinin içine giren düğümlere göre indirgenmiş maliyet listesi güncellenecektir.

Kısaca algoritmanın çalışma mantığı şu üç hususta toparlanabilir. Bir değişken olan ve "*graph*" ismi verilen düğüm listesi olsun. Bu ilk başta sıfır olacaktır. Bu boş "*graph*" listesinden ilk olarak bir "*rootComponent*" alt listesi oluşacaktır ve bu dizinin tek elemanı ise başlangıç düğümü olacaktır. Buna ilk adım denilebilir. Sonrasında bu "*rootComponent*" alt listesinden yola çıkarak uygun bir iki düğüm arası bağlantı seçilip bulunacaktır. Buna "*a*" ismini verildiğini varsayalım. Bu "*a*" bağlantısı ile bağlantı maliyet tablosundan yola çıkılarak "indirgenmiş maliyet" tablosu güncellenecektir. Bu iki işleme de adım 2 denilebilir. Daha sonra da bu "*a*" bağlantısı kalıcı olarak çözüm kümesi olan "*graph*" listesine eklenecektir. Buna da adım 3 diyelim. Hedef düğüm "*graph*" dediğimiz kalıcı çözüm listesine eklenene kadar ikinci ve üçüncü adım tekrar edilecektir.

Bu işlemlerin ardından ise sonuç ortaya çıkacaktır. Bu sonuç dediğimiz veri yapısı ise başlangıçtan hedefe olan en kısa yolu içeren bir ağaç(tree) yapısıdır. Algoritmanın ikinci kısmı ise bu ağaç veri yapısından yola çıkılarak kaynak-hedef arasındaki net yolu ortaya çıkarmaya çalışmaktadır. Genel olarak ikinci kısmın çalışma şekli incelendiğinde iki konu ortaya çıkmaktadır. Devamında bağlantı olmayan yani bu ağaca bağlı olan ve kendisine bu bağdan başka bağlantısı olmayan düğümleri kaldırıyoruz. Bunlar çıkmaz sokak gibi düşünülebilir. İkinci ve daha önemlisi olarak ise ağaç veri yapısı içerisinde daire(cycle) şeklinde döngü oluşturacak tüm alt düğüm grupları saptanır. Bunlar da temizlenerek çalışma sonlandırılır.

2.5.2. Algoritma Sözde Kodu

Algoritma ile ilgili olarak genel bir anlayış kazandırmak için genel hatlarıyla sözde kodunu şöyle verebiliriz.

- "rootComponent" ve "graph" <- düğüm dizisi //step 1
- "graph" kalıcı listesinden bir kerelik "rootComponent" geçici listesini oluştur.
- Hedef düğüm "graph" dizisine girene kadar şu adımları tekrar et:
  - "rootComponent" listesinden yola çıkarak en uygun düğümü bul. Bu bağlantı hedef düğümü "rootComponent" dizisi içerisinde olan bağlantılardan seçilir.
  - Sonra bu seçilen bağlantıyı kullanarak indirgenmiş maliyet tablosunda güncelleme yap. Yani her bir bağlantının hedef düğümü "rootComponent" listesinde ise; o bağlantının indirgenmiş maliyet tablosundaki değerini en başta seçilen en düşük bağlantı maliyetine sahip bağlantının(a bağlantısı) değeri kadar eksiltilerek güncelle.[14]
  - Sonrasında bu "a" düğümünü "graph" zincirine ekle. Daha sonrasında ise "rootComponent" dizisi temizle ve "graph" zincirinden kopyalayarak tekrar hesapla.

- “graph” çözüm kümesi içerisindeki hem kaynak tarafına hem de hedef tarafına bağlantısı olmayan düğümleri temizle. Bunlara karar vermek için bazı veri yapıları indirgenmiş maliyet güncellemesi yapılırken tutulur.
- Tüm bu “graph” listesindeki düğümler için halka(cycle) şeklinde döngü üretebilecek alt düğüm gruplarını sapta ve çözümle.

3. MALZEME VE YÖNTEM

3.1. FLOODLIGHT

Floodlight Açık YTA Denetleyicisi Apache lisanslı, Java tabanlı Openflow standartlarını uygulayan bir sanal ağ kontrolör katmanıdır. Big Switch Networks'ten birkaç mühendisi de içeren bir topluluk geliştiricisi tarafından desteklenmektedir. Floodlight denetleyicisi, tüm ağ kurallarının korunmasından ve ağ trafiğinin nasıl idare edilebileceği konusunda gerekli bilgilerin verilmesinden sorumludur. Bu denetleyici, kullanıcıların ihtiyaçlarına daha iyi uyum sağlamak ve kullanıcıların ağlar üzerinden daha iyi kontrol sahibi olmalarını sağlamak için değiştirilebilir.

Openflow, Open Networking Foundation tarafından yönetilen açık bir standarttır. Anahtar yoluyla bir uzaktan kumandanın bir protokolü iyi tanımlanmış bir "yönlendirme talimatı seti" ile ağ cihazlarının davranışını değiştirebileceğini belirtir. Floodlight, artan sayıda anahtar, yönlendirici, sanal anahtar ve Openflow standardını destekleyen erişim noktalarıyla çalışmak üzere tasarlanmıştır.

3.2. MININET

Mininet, sanal bilgisayarlar(virtual machine), anahtarlar(switch), denetleyiciler(controller) ve bağlantılar(links) oluşturan bir ağ emülatörüdür. Mininet uygulaması standart Linux çekirdeği üzerinde çalışmaktadır. Mininet anahtarları, YTA'lar için son derece toleranslı ve esnetilebilen bir yönlendirme sağlayan OpenFlow'u destekler.

Mininet geliştirme, öğrenme, prototip oluşturma, test etme ve hata ayıklamayı(debugging) destekler. Birden fazla eşzamanlı geliştiricinin aynı topolojide bağımsız olarak çalışmasına izin verir.

Fiziksel bir ağ olmadan karmaşık topoloji testlerini mümkün kılar. Belirli topolojileri destekler ve bir dizi parametrik topoloji içerir. Topoloji oluşturmak ve test etmek için basit ve genişletilebilir bir Python API ve ayrıca otomatik oluşturma yeteneğini sağlar.

Mininet, sistem geliştirme ve hata ayıklama durumunda deneme yapmanın kolay bir yolunu sağlar. Mininet ağları, standart Unix / Linux ağ uygulamalarının yanı sıra gerçek Linux çekirdeğini ve ağ yığınını içeren gerçek kodu çalıştırır [15].

3.3. UYGULAMA

3.3.1. Kurulum

3.3.1.1. Manuel İşletim Sistemi Kurulumu

Bu Floodlight ve Mininet uygulamalarının üstüne inşa edilecek uygulama için ilk adım olarak; işletim sisteminin uygun hale getirilmesi gerekmektedir. Çünkü Mininet uygulaması Linux kernel'ları üzerinde çalışmaktadır. Bu sebeple Floodlight + Mininet ikilisini bir arada senkronize halde çalıştırmak için bir Linux dağıtımı işletim sistemine sahip olmak gereklidir. Bunun birden fazla yolu mevcuttur.[16]

Ubuntu 18.04 sürümünün Floodlight ve Mininet ile birlikte uyumlu çalıştığı bu uygulamayı geliştirme esnasında görülmüştür. Bu sebeple Ubuntu 18.04 sürümünün yüklü olduğu bir bilgisayar ile çalışılması tavsiye olunur. Daha eski sürümlere verilen işletim sistemi desteği yeterli gelmeyebileceğinden dolayı sürüm seçiminde dikkatli olunmalıdır.

Ubuntu sürüm 18.04'ü kurmak için gerekli adresten[16] iso uzantılı dosya indirilerek bilgisayara kaydedilir. Bir tane ücretsiz iso dosyasını Cd'ye yazan uygulama kurulumu vasıtasıyla kurulum cd'si hazırlanabilir. Free Iso Burner programı kullanılabilir.

Ubuntu sürüm 18.04'ü bilgisayarınıza kurduğunuz takdirde sırası ile şu kurulumları yapmak gerekecektir.

- Java 8(Minör sürüm fark etmiyor ama en günceli tavsiye edilir)
- Intellij IDEA(Eclipse kurulumunun alternatifidir)
- Intellij içerisinden(File→New Project from Version Control →GIT) menüsü kullanılarak projenin kaynak kodu elde edilmelidir. IDE'nin çıkaracağı dialog kutusuna "https://github.com/floodlight/floodlight" adresi girilir ve Floodlight kaynak kodu çekilir. Master ismi verilen son sürümünün çekilmesi gerekmektedir.

• Eğer Ubuntu bilgisayarda GIT kurulumu yok ise Ubuntu işletim sisteminin sağladığı paket yöneticisinden kolayca kurulabilir.

apt-get install git

• Ant kurulumu yapılabilir. Apache'nin adresinden [17] son stabil sürümünün .tar.gz kurulum paketi indirilebilir. Eğer Intellij IDEA Ultimate kurarak başladıysanız bu adımı atlayabilirsiniz. IDEA içerisinde ant kurulumu mevcuttur.

• Java projelerinde bir standart olan Maven'in de kurulumu gerekmektedir. Intellij IDEA Ultimate kurulumu yaptıysanız IDE olarak, Intellij içerisinde kurulu geliyor. Bu adımı atlayabilirsiniz.

• Mininet kurulumu yapılmalıdır. http://mininet.org/download/ adresine gidilerek indirilmesi sağlanılabilir. 2.2.1 versiyonu şu an için güncel olup sorunsuz çalışmaktadır.

• Intellij IDEA içerisinden ant build toolundan "+" işaretine tıklanarak bir ant build oluşturulup ant build çalıştırılması Floodlight'ı build edecek olup, geliştirmeye hazır hale getirecektir.

3.3.1.2.Hazır Sanal Makine Dosyası Kurulumu

Floodlight platformunun internet sitesinde yer alan bağlantı linkinden indirme gerçekleştirilebilinir. Link şu şekildedir: http://opennetlinux.org/binaries/floodlight-vm.zip

Bu Floodlight VM içerisinde Floodlight v1 sürümü, Eclipse IDE, mininet, open vSwitch, wireshark ve w/openflow kurulmuş vaziyette bulunmaktadır. VM içerisinde kullanılan işletim sisteminde ön tanımlı olarak gelen kullanıcının şifresi de Floodlight'tır.

Bu imajda Floodlight'ın eski bir sürümü çalışmaktadır. Bununla beraber Mininet'in de bir versiyonu kurulmuştu. Sabit, 10gb'lık bir imaj. Yani bunu çalıştırdığınızda sanal işletim sistemi 10gb oluyor ve artırılması mümkün değil. Virtualbox'un 2019 Ekim ayı başındaki en son sürümü ile denendiğinde mümkün değildi. Bu kurulumda bilgisayarda hafıza sorunu sebebiyle bazı güncellemeler alınamıyor. Bu sebeple sıfırdan kurulum gerçekleştirilmesi tavsiye olunur.

Floodlight projesini Eclipse IDE içerisinde açmak için eclipse simgesine tıklamanız yeterlidir. Bu Sanal Makine(Virtual Machine, VM) içerisinde tüm kurulumlar hazır gelmektedir. Fakat

artık Floodlight master branch üzerinden daha yeni bir sürümde ilerlemektedir. Özellikle web ara yüzü çok daha anlaşılır olduğu için yeni sürümü tercih edilmiştir. Floodlight son sürüm Java 1.8 de çalıştığı için https://www.java.com/tr/download/ adresinden Java 1.8 veya sonrası bir sürüm indirilmelidir. Ardından ise sıralanmış olan kodlar sırası ile çalıştırılır. Oracle şirketi tarafından JDK kullanıcı şartları değiştirildiği için Ubuntu üzerinden shell prompt üzerinden direkt komutlarla indirilmesi artık mümkün değildir.

Öncelikle Floodlight'ın sitesinden Ubuntu'nun 14.04 nolu versiyonunun bir imajı indirilmiştir. Bu ortamda çalışma sürdürülmüştür. Yaşanılan bellek sorunu ve buna bağlı olarak güncelleme sorunlarının ardından bilgisayara en baştan Ubuntu'nun Ekim 2019 başındaki en son versiyonu kurulmuştur. Bilgisayarın tüm kapasitesi işletim sistemine ayrılabildiği için bellek sorunu yaşanmamıştır.

- git clone git://github.com/floodlight/floodlight.git
- cd floodlight/
- ant
- java -jar target/floodlight.jar

Bu komutların ardından Floodlight derlenmeye hazır haldedir. Denetleyici olarak Floodlight'ın bir topoloji üzerinde çalışması gerekmektedir. O sebeple öncelikle bir topolojiyi Mininet komutları ile çalıştırmak için aşağıdaki komutlar girilmelidir.

"sudo mn - -custom miniTopo.py --controller=remote, ip=10.0.2.15 port=6653"

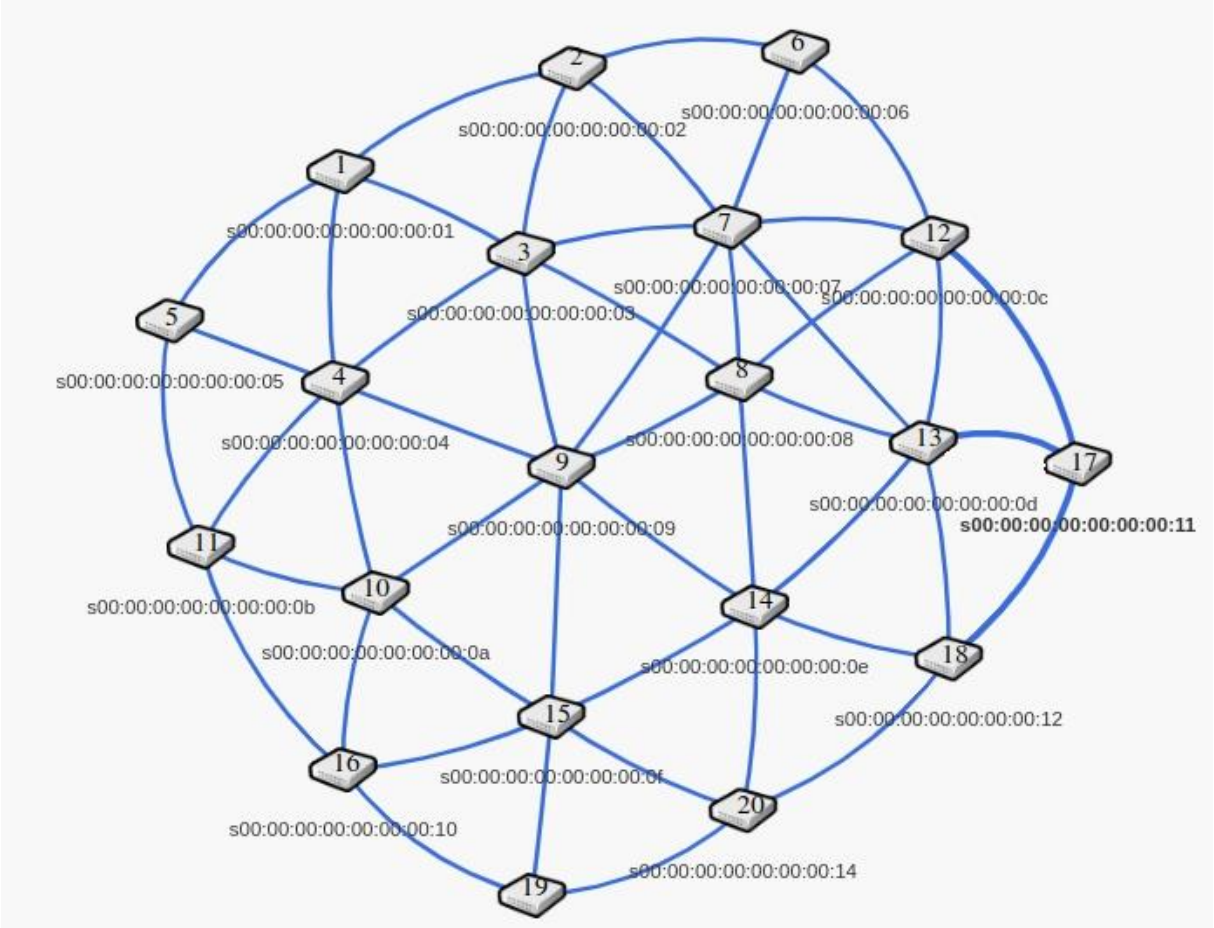
Bu verilen IP/Port bilgileri Floodlight projesinin ön tanımlı olarak ayağa kalktığı IP/Port bilgileridir. Bu komutun ardından Floodlight projesi çalıştırıldığında bu topolojinin denetleyicisi olarak çalışmaya başlayacaktır. "Controller" isimli parametreyi de "remote" değerine ayarlayarak Mininet uygulamasını Floodlight uygulamasını dinler hale getiririz.

Ayrıca, "miniTopo.py" parametresi de Mininet uygulamasına geçirilmiştir. Bunun anlamı da Mininet ayağa kalkarken bu dosya içerisinde belirtilen topoloji ile ayağa kalkar.

3.3.2. Topoloji Oluşturma ve Python Kodu

Bu çalışma kapsamında; küçük topolojilerde yapılan basit testlerle Floodlight denetleyicisinin Mininet topolojileriyle uyumlu bir şekilde çalışmaya başladığı görülmüştür. Bunun ardından geliştirmelere başlanmıştır. Geliştirme süresince geliştirmenin doğruluğundan emin olmak için iki farklı topoloji kullanılmıştır. Daha sonra da bu topolojiler biraz daha özelleştirilerek test çıktı kümelerinde de kullanılmıştır.

3.3.2.1. Tek Alt Ağ İçeren Topoloji



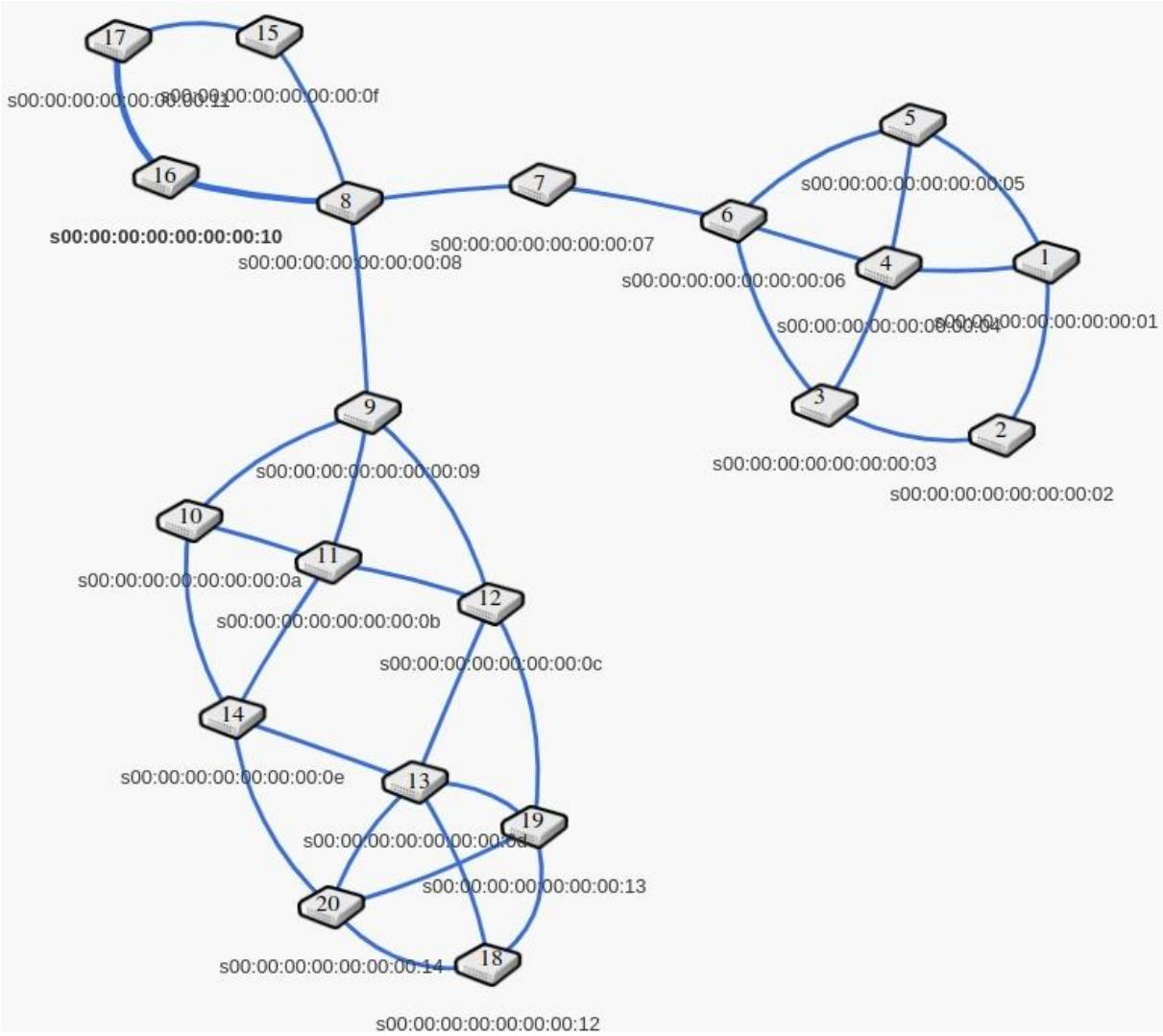
Şekil 3.1: Tek Alt Ağ Topolojisi[18]

Şekil 3.1'de verilen topoloji tek alt ağdan oluşmaktadır. Yani birbirine bağlı iki ayrı topoloji yoktur. Bu topolojinin oluşturulması için gereken Python kodu şu şekildedir.[18]

```
from mininet.topo import Topo
class MyTopo( Topo ):
    def __init__( self ):
        Topo.__init__( self )
        Switch1 = self.addSwitch('s1')
        Switch2 = self.addSwitch('s2')
        Switch3 = self.addSwitch('s3')
        Switch4 = self.addSwitch('s4')
        Switch5 = self.addSwitch('s5')
        Switch6 = self.addSwitch('s6')
        Switch7 = self.addSwitch('s7')
        Switch8 = self.addSwitch('s8')
        Switch9 = self.addSwitch('s9')
        Switch10 = self.addSwitch('s10')
        Switch11 = self.addSwitch('s11')
        Switch12 = self.addSwitch('s12')
        Switch13 = self.addSwitch('s13')
        Switch14 = self.addSwitch('s14')
        Switch15 = self.addSwitch('s15')
        Switch16 = self.addSwitch('s16')
        Switch17 = self.addSwitch('s17')
        Switch18 = self.addSwitch('s18')
        Switch19 = self.addSwitch('s19')
        Switch20 = self.addSwitch('s20')
        self.addLink( Switch1, Switch2 ) #
        self.addLink( Switch1, Switch3 )
        self.addLink( Switch1, Switch4 )
        self.addLink( Switch1, Switch5 )
        self.addLink( Switch2, Switch3 )
        self.addLink( Switch2, Switch6 )
        self.addLink( Switch2, Switch7 )
        self.addLink( Switch3, Switch4 )
        self.addLink( Switch3, Switch7 )
        self.addLink( Switch3, Switch8 )
        self.addLink( Switch3, Switch9 )
        self.addLink( Switch4, Switch5 )
        self.addLink( Switch4, Switch9 )
        self.addLink( Switch4, Switch10 )
        self.addLink( Switch4, Switch11 )
        self.addLink( Switch5, Switch11 )
        self.addLink( Switch6, Switch7 )
        self.addLink( Switch6, Switch12 )
        self.addLink( Switch7, Switch8 )
        self.addLink( Switch7, Switch9 )
        self.addLink( Switch7, Switch12 )
        self.addLink( Switch7, Switch13 )
        self.addLink( Switch8, Switch9 )
        self.addLink( Switch8, Switch12 )
        self.addLink( Switch8, Switch13 )
        self.addLink( Switch8, Switch14 )
        self.addLink( Switch9, Switch10 )
        self.addLink( Switch9, Switch14 )
        self.addLink( Switch9, Switch15 )
        self.addLink( Switch10, Switch11 )
        self.addLink( Switch10, Switch15 )
        self.addLink( Switch10, Switch16 )
        self.addLink( Switch11, Switch16 )
        self.addLink( Switch12, Switch13 )
        self.addLink( Switch12, Switch17 )
        self.addLink( Switch13, Switch14 )
        self.addLink( Switch13, Switch17 )
        self.addLink( Switch13, Switch18 )
        self.addLink( Switch14, Switch15 )
        self.addLink( Switch14, Switch18 )
        self.addLink( Switch14, Switch20 )
        self.addLink( Switch15, Switch16 )
        self.addLink( Switch15, Switch19 )
        self.addLink( Switch15, Switch20 )
        self.addLink( Switch16, Switch19 )
        self.addLink( Switch17, Switch18 )
        self.addLink( Switch18, Switch20 )
        self.addLink( Switch19, Switch20 )
topos = { 'mytopo': ( lambda: MyTopo() ) }
```

Şekil 3.1: Tek alt Ağ topolojisi Python Kodu

3.3.2.2.Birden Fazla Alt Ağ İçeren Topoloji



Şekil 3.2: Çoklu Alt Ağ Topolojisi

Şekil 3.2'de verilen topoloji birden fazla ağdan oluşmaktadır. Yani birbirine bağlı üç ayrı topoloji vardır. Bu topolojinin oluşturulması için gereken Python kodu şu şekildedir.

```
from mininet.topo import Topo

class MyTopo(Topo):

    def __init__(self):

        Topo.__init__(self)

        ######################################## switchler


        s1 = self.addSwitch('s1')
        s2 = self.addSwitch('s2')
        s3 = self.addSwitch('s3')
        s4 = self.addSwitch('s4')
        s5 = self.addSwitch('s5')
        s6 = self.addSwitch('s6')
        s7 = self.addSwitch('s7')
        s8 = self.addSwitch('s8')
        s9 = self.addSwitch('s9')
        s10 = self.addSwitch('s10')
        s11 = self.addSwitch('s11')
        s12 = self.addSwitch('s12')
        s13= self.addSwitch('s13')
        s14 = self.addSwitch('s14')
        s15 = self.addSwitch('s15')
        s16 = self.addSwitch('s16')
        s17 = self.addSwitch('s17')
        s18 = self.addSwitch('s18')
        s19 = self.addSwitch('s19')
        s20 = self.addSwitch('s20')

        self.addLink(s1 , s2)
        self.addLink(s1 , s4)
        self.addLink(s1 , s5)
        self.addLink(s2 , s3)
        self.addLink(s3 , s4)
        self.addLink(s3 , s6)
        self.addLink(s4 , s5)
        self.addLink(s5 , s6)
        self.addLink(s4 , s6)
        self.addLink(s6 , s7)
        self.addLink(s7 , s8)
        self.addLink(s8 , s9)
        self.addLink(s9 , s10)
        self.addLink(s9 , s11)
        self.addLink(s9 , s12)
        self.addLink(s10 , s11)
        self.addLink(s10 , s14)
        self.addLink(s11 , s14)
        self.addLink(s11 , s12)
        self.addLink(s12 , s13)
        self.addLink(s13 , s14)
        self.addLink(s8 , s15)
        self.addLink(s8 , s16)
        self.addLink(s15 , s17)
        self.addLink(s16 , s17)
        self.addLink(s12 , s19)
        self.addLink(s14 , s20)
        self.addLink(s18 , s19)
        self.addLink(s18 , s20)
        self.addLink(s19 , s20)
        self.addLink(s13 , s18)
        self.addLink(s13 , s19)
        self.addLink(s13 , s20)

        #self.addLink(leftHost1 , leftSwitch , bw = 10)

topos = { 'mytopo' : ( lambda : MyTopo() )  }
```

Şekil 3.3: Çoklu Alt Ağ Topolojisi Python Kodu

3.3.2.3.Topoloji Kullanımları

Oluşturulan topolojilerde her bir yönlendirme algoritması ayrı ayrı çalıştırılmıştır. Her birinin sonucu ayrı ayrı tutulup kıyaslanmıştır. Bu bölümün ilerleyen kısımlarında topoloji ve algoritma bazında ayrı ayrı sonuçlar irdelenecektir.

Tüm bu değişen girdiler sonucunda elde edilen veriler ayrı ayrı toplanmıştır. Bu veriler formatlı bir şekilde metin dosyalarına yazılmıştır. Her bir metin dosyası bir topoloji üzerinde algoritmalardan birinin denenmesi anlamına gelmektedir. Bu metin dosyaları için ise bir katman daha hazırlanmıştır. Bu formatlı metin dosyalarını okuyarak bir görsel hazırlayan katman projeye eklenmiştir.

Bu görselde topoloji içerisinde yer alan tüm düğümler görsele yerleştirilmiştir. Ardından algoritmaların sonuç olarak verdiği en düşük maliyetli yollar farklı renklerde görsele dökülmüştür.

3.3.3. Proje Gelişimi

İlk olarak Floodlight projesinin YTA'lar üzerinde çalışan bir denetleyici olduğu belirtilmişti. Bu denetleyici tüm ağı denetlemekle görevlidir. Görevlerinden biri ise ağ içerisindeki noktalar arasındaki en az maliyetli yolları düzenli olarak hesaplayıp bilmektir. Floodlight projesinin bu sorunu nasıl çözdüğü ilk incelenen konudur. Bunun sonucu olarak Topology modülünün içindeki TopologyInstance.java sınıfında yer alan Dijkstra fonksiyonunun bu işten sorumlu olduğu anlaşılmıştır. Bu fonksiyonun girdisi network içindeki tüm düğümler ve bu düğümlere ait maliyet bilgileridir. Ek olarak kaynak ve hedef düğümler de fonksiyonun girdilerindendir. Projenin bu kısmının çıktısı ise en kısa yol ve bunlara ait maliyetlerdir.

Burası saptandıktan sonra Auction.java ve DualAscent.java isimli iki Java sınıfı oluşturulmuştur. Bu Java sınıflarının içerisinde Dijkstra algoritmasını hesaplayan fonksiyonun benzeri oluşturulmuştur. Burada benzerden kasıt, aynı tip ve sayıda parametreler alıp aynı veri yapısı tipinde cevap geri dönecek şekilde yazılmış olmasıdır.

Burada geliştirmeler yapılırken her bir algoritma için ayrı bir sınıf yazılmıştır. Bu her bir sınıf içerisinde ise birden fazla "private" fonksiyon olup sadece bir tane "public" fonksiyon olmasına özen gösterilmiştir. "public" olan fonksiyonlar burada en kısa yol hesaplamasını yapan ana kod parçalarıdır. Bu fonksiyon da yazılan her bir sınıf için aynı tür ve sayıda parametreler alıp aynı

veri yapısını geri döndürmektedir. Bu da kolaylıkla birbirleri yerine kullanılabilirliği sağlamaktadır.

3.3.3.1.Açık Artırma Algoritması(Auction)

Açık Artırma algoritmasının çalışma mantığı önceki bölümlerde detaylıca verilmişti. Asıl mantık her bir düğümün değerini bir vektör üzerinde tutmak ve bunun üzerinden sürekli bir kıyaslama yapmaktır. Kıyas sonucunda ya ilgili düğümü çözüm kümesine ekleriz ve çözüm kümemiz büyür, ya da çözüm kümemize eklediğimiz son düğüm çözüm kümesinden çıkartılır. Bu adım hedef düğüm çözüm kümesine eklenene kadar devam eder.

```
DatapathId currentNode = from;
int currentInd = mapOfNodesKeepsIndex.get(from.getLong());
while (true) {
    int longToFindRoute = Integer.MAX_VALUE;
    Link nexLink = null;
    int nexLinkCost = 0;
    Set<Link> linksForGivenNode = links.get(currentNode);
    for (Link link : linksForGivenNode) {
        int local_calculatedNextNodeValue;
        int local_calculatedWeightToNextNode;
        if (link.getSrc() != currentNode) continue;
        local_calculatedNextNodeValue =
                    Math.toIntExact(pNodeVector[mapOfNodesKeepsIndex.get(link.getDst().getLong())]);
        local_calculatedWeightToNextNode =  getLinkCost(linkCost,link);
        int aggregatedInt = local_calculatedNextNodeValue+local_calculatedWeightToNextNode;
        if (nexLink == null ||checkIfLinkHasLessCost(longToFindRoute,nexLink, aggregatedInt,link)) {
            longToFindRoute = local_calculatedNextNodeValue + local_calculatedWeightToNextNode;
            nexLink = link;
            nexLinkCost = getLinkCost(linkCost,link);
        }
    }
    if (pNodeVector[currentInd] < longToFindRoute) {
        //contractPath
        pNodeVector[currentInd] = longToFindRoute;
        DatapathId getLastFrom = getLastFrom(currentNode, nexthoplinks);
        if (getLastFrom != null && !getLastFrom.equals(from)) { // do not remove the starting point
            currentNode = getLastFrom;
            nexthoplinks.remove(currentNode);
            currentInd = mapOfNodesKeepsIndex.get(currentNode.getLong());
        }
    } else {
        //extend Path
        nexthoplinks.put(currentNode, nexLink);
        cost.put(currentNode, nexLinkCost);
        if (nexLink.getDst() == toNode) {
            break;
        }
        currentNode = nexLink.getDst();//it could be either previous node or extended node
        currentInd = mapOfNodesKeepsIndex.get(currentNode.getLong());
    }
}
BroadcastTree ret = new BroadcastTree(nexthoplinks, cost);
return ret;
```

Şekil 3.4: Açık Artırma Algoritması Kodu

3.3.3.2.İkili Tırmanma Algoritması(Dual Ascent)

İkili Tırmanma algoritmasının da çalışma mantığı genel olarak daha önceki başlıklarda verilmişti. Kısaca özetlemek gerekirse; başlangıç noktasından başlayarak sıra ile en uygun

düğümler sırası ile çözüm kümesine eklenir. Her bir ekleme işlemi esnasında en uygun düğüm bulunduktan sonra, çözüm kümesine eklenmeden önce çözüm kümesi içerisindeki tüm bağlantı maliyetleri uygun şekliyle güncellenir. Hedef düğüm çözüm kümesine eklendiğinde algoritma hedefine ulaşmış olmaktadır. Ardından ise, çözüm kümesi içerisinde çevrim oluşturan düğüm kümesi varsa bunlar temizlenir ve algoritma çalışmasını durdurur.[14]

```
public BroadcastTree findPath(Map<DatapathId, Set<Link>> links, DatapathId src, DatapathId dst,
                              Map<Link, Integer> linkCost, Map<Link, Integer> linkBandWith) {
    this.linkCost = linkCost;
    reducedCost = new HashMap<Link, Integer>(this.linkCost);
    this.src = src;
    this.dst = dst;
    requiredVertices.add(this.dst.getLong());
    gp = new Graph(links, false);
    this.links = links;
    this.siwCount = links.size();
    connections = initWithFalse(links.size());
    this.linkBandWith = linkBandWith;
    int index = 0;
    for (DatapathId node : links.keySet()) {
        connections[index][index] = true;
        nodeIndexes.put(node.getLong(), index);
        index++;
    }
    HashSet<Long> tempGroup = eveluateRoot(gp); // birinci adim baslangic
    while (tempGroup != null) {
        Link tempLink = findMinLink(gp, tempGroup);
        editCosts(tempGroup, tempLink);
        addLinkToSolution(gp, tempLink);
        tempGroup = eveluateRoot(gp);
    }
    setSolution(gp); // resolve cycles
    HashMap<DatapathId, Link> convertedSolution = convertSolutionForBroadCastTree(treeHash);
    HashMap<DatapathId, Integer> convertedCost = convertCostForBroadCasTree(treeHash);
    BroadcastTree ret = new BroadcastTree(convertedSolution, convertedCost);
    return ret;
}
```

Şekil 3.5: İkili Tırmanma Algoritması Kodu

3.3.3.3.Test Süreci

Bu tez çalışması içerisinde birden fazla algoritmanın ve algoritmalarla birlikte birden fazla kriterin kıyaslanması amaçlanmıştır. Bu sebeple öncelikle sıfırdan yazılan Açık Artırma ve İkili Tırmanma algoritmaları test senaryolarında düşünülmüştür. Ardından, test sonuç kümelerini zenginleştirmek amacıyla Bellman-Ford algoritması da uygulamaya entegre edilmiştir.[19] Projenin orjinalinde ise Dijkstra algoritması olduğu için bu üç algoritma Disjktra algoritmasıyla beraber test edilmiştir.

Floodlight projesi yapması gerektiği gibi tüm düğümler arasında ayrı ayrı en az maliyetli yolu hesaplamak için algoritmayı çalıştırmaktadır. Geliştirme ve test süresince test verisi üzerinde kodun nasıl çalıştığı adım adım incelenmiştir. Bu test verileri de belirli noktalar arasında önceden çıkarılıp hazır edilmiştir. O sebeple yeni yazılan kodlar önceden belirlenmiş noktalar

arasında çalıştırılmıştır sadece. Bunu yapmanın yolu da kaynak ve hedef düğümlerini filtreleyerek önceden belirlenmiş düğümler arasında çalışmasını sağlamaktır.

İkili Tırmanma, Açık Artırma, Dijkstra ve Bellman Ford algoritmaları bu çalışma kapsamında ürettikleri sonuçlar itibarıyla kıyaslanmıştır. Kıyaslama sonuçlarını zenginleştirmek adına algoritmaların yalın hallerinin yanı sıra iki farklı normalizasyon yöntemi ile zenginleştirilmiş hallerinin de sonuçları eklenmiştir. Bunlar sonuçlar bölümünde görülebilir.

3.3.4. Algoritmaların Geliştirilmesi ve Uygunluğun Arıttırılması

Auction ve Dual Ascent yaklaşımlı algoritmaların çalışması ile ilgili detaylar 2.4 ve 2.5 numaralı başlıklarda verilmişti. Bu geliştirilmiş olan iki ayrı algoritmanın bir ortak noktası mevcuttur. O da, bir sonraki adıma karar verirken bir sonraki muhtemel bağlantıların sadece bağlantı maliyetlerini göz önüne almaktır. Maliyetlerin hesaplanması ve yorumlanması farklı olsa da dikkate alınan veriler sadece bağlantı maliyetleridir. Verilerin nasıl yorumlandığı kısmı Auction ve Dual Ascent yaklaşımlı algoritmaların işleme mantığına göre değişmektedir ama hangi verilerin dikkate alındığı ise kodlanan tüm algoritmalar için aynıdır.

Kodlanan bu algoritmaların geliştirilerek YTA'ların sunduğu hizmet kalitesini arttırdığı gözlemlenmiştir. Bu kalite artırma işi kodlanan algoritmaların ne olduğundan bağımsız olarak hesaplama yaparken göz önüne alınan verilerin artırılmasına dayanmaktadır. Bu tezin amaçlarından birisi olarak, sadece bağlantı maliyetleri değil, ikinci bir parametrenin de göz önüne alınması sağlanmaktadır. Bu ikincil veri ise o bağlantının sahip olduğu o anlık müsait bant genişliğidir. Bu işlemin sonucu olarak ilgili bağlantının sahip olduğu bant genişliği tercih edilebilirliğine etki ediyor olacaktır. Bu iki ayrı veri kümesi normalize edilerek bir araya getirilmektedir.

3.3.5. Normalizasyon

Bu ikincil parametrenin eklenmesinden önce, ön tanımlı olarak sadece bağlantı maliyetleri vardı. Bunun anlaşılmasını kolaylaştırmak için iki nokta arasındaki fiziki uzaklığa benzetilebilir. Bu sadece bir benzetmedir ve YTA'lar açısından farklılık gösterebilir.

Floodlight projesi içerisinde iki nokta arasındaki en düşük maliyetli yolu hesaplamak için ayrı bir fonksiyon vardır. Bu fonksiyon içerisinde bağlantı maliyeti olarak çeşitli veriler kullanılabilir. Bu projenin kurulumunda bağlantı maliyeti olarak linklerin sahip olduğu ve

Floodlight tarafından sağlanan gecikme değeri kullanılmıştır. Bu Floodlight projesinde "latency" olarak geçmektedir. İşte bu veri ikincil bir başka veri ile normalize edilerek kullanılacaktır.

Bunun üzerine biz nasıl karar verme mekanizmasını geliştirebiliriz diye araştırdığımızda şu veriye ulaştık. Bir bilgisayar ağındaki gecikme değeri dışındaki diğer durum değerlerini de okuyup bunu karar verme mekanizmasına enjekte edersek performansı geliştirebiliriz. Burada iki ayrı türden ve birimden veriyi bir araya getirip anlamlı bir şekilde yorumlamak ihtiyacı doğdu. Bunun sonucunda ise bant genişliği değerini normalize ederek gecikme değeriyle birlikte kullanmaya karar verdik.

3.3.5.1.Min-Max Ölçekleme(Rescaling) Yöntemi İle Normalizasyon

Normalizasyon formülü aşağıdaki gibi tanımlanmıştır. Amaç en küçük ve en büyük değerlerin oranlarına göre lineer bir şekilde tüm aralığı 0 ile 1 değeri arasında getirmektir. Aşağıdaki formülde (3.1) belirtilmiştir.

$$X' = \frac{\mathrm{X} - X_{min}}{X_{max} - X_{min}} \tag{3.1}$$

Bu yöntemi kullanarak normalize etmeye çalıştığımızda olan şey tam olarak şudur. Elimizde bulundurduğumuz bir dizi bant genişliği verisini 0 ile 1 sayısı arasına çekmektir. Bunun yolu da bu dizi içerisindeki bir elemanı(X) o dizinin en büyük ve en küçük elemanı arasındaki bir aralığa yerleştirmektir. Yukarıdaki formülde de görüleceği üzere normalize etmek istediğimiz değerden o dizinin en küçük elemanını çıkartıyoruz. Daha sonrasında ise bu sayıyı o dizinin en büyük elemanı ile en küçük elemanının sayısal değerinin farkına bölüyoruz. Buradan anlaşılacağı üzere eğer ki dizinin en küçük elemanının normalize edilmiş değerine erişmeye çalışıyorsak sonucumuz 0 olacaktır. Çünkü bu payda hesabının pay kısmı sıfır olacaktır. Bu da sonucu sıfıra götürecektir. Aynı şekilde dizinin en büyük değerinin normalize edilmiş değeri de 1 olacaktır. Formülden görüldüğü üzere en büyük değer ile en küçük değerin farkını yine kendisine bölmüş olacağız. Bu da bizi 1 sayısına götürür.

Bu normalizasyon yönteminin uygulama mantığını şu şekilde basitçe açıklayabiliriz. Algoritmalar orijinal haliyle bir düğümden sonra seçilecek yola karar verirken bağlantı maliyetine bakarak karar verirler. Bunlar için M_1 ve M_2 değişkenlerini tanımlamış olalım ve değerleri sıra ile 60 ve 64 olsun. Bağlantı maliyeti daha düşük olacağı için M_1 bağlantısı tercih

edilir normalde. Ama buradaki karar verme yetisini güçlendirmek için biz o ilgili bağlantıların o anlık müsait bant genişliğini de hesaba katmak istedik. Bant genişliğinin yüksek olmasının o ilgili bağlantının tercih edilebilirliğini artırdığı düşüncesine geldik. Bant genişliği değeri için ise B_1 ve B_2 değişkenlerini tanımlamış olalım. Bunların değeri de sırası ile 20 birim ve 24 birim. Sistemdeki en düşük bant genişliği 16 birim ve en yüksek değer ise 26 birim olsun. Bunların ışığında ortaya çıkacak hesaplama şöyle olacaktır. X1, B_1 değeri için ağırlık katsayısı anlamına gelecektir. Aynı şekilde de X2, B_2 değeri için ağırlıklı ortalama değeri olacaktır.

$$X1 = \frac{20 - 16}{26 - 16}$$

$$X2 = \frac{24 - 16}{26 - 16}$$

Bu eşitlikten X1 0,4 çıkacaktır, X2 ise 0,8 çıkacaktır. Bunun anlamı ise şudur. İkinci bağlantının sahip olduğu 24 birimlik bant genişliği değeri; ilk bağlantının da bant genişliğinin 16 olmasından dolayı iki kat daha tercih edilebilir anlamına gelmektedir. Bağlantı maliyetleri 60 ve 64 olmasına rağmen ikinci bağlantı algoritma tarafından seçilecektir.

Bunun açıklamasını da şöyle yapabiliriz. Bağlantı maliyeti tercih edilebilirlikle ters orantılı olup bant genişliği ise tercih edilebilirlikle doğru orantılıdır. 60 x 0,8 değeri 64 x 0,4 değerinden daha yüksek çıkacağı için algoritma tarafından tercih edilmeyecektir. Bant genişliğinin ikinci bağlantıda daha yüksek olması algoritmanın kararını değiştirmiş oldu.

Bu örnek sadece bant genişliği değerinin normalize edilerek bağlantı maliyeti hesabına dâhil edilmesi üzerine kurulmuştur. Bu uygulamanın da sonuç veri setinden yola çıkarak verimli bir şekilde çalıştığı gözlemlenmiştir. Öte yandan, her iki değer de yani hem bant genişliği hem de bağlantı maliyetleri normalize edilerek de hesaplanabilir. Bu örnek konunun anlaşılması için basit bir şekilde verilmiş olup, uygulama esnasında yapılan testlere göre uygulamada her iki yöntem de tercih edilebilir[20].

3.3.5.2.Standart Skorlama(Standart Scoring) Yöntemi İle Normalizasyon

Standart skorlama çok yararlı bir istatistiktir, çünkü normal dağılımımızda meydana gelen bir skorun olasılığını hesaplamamıza izin verir. Ayrıca Z-Skor Normalizasyon (Z-Score Normalization) olarak da adlandırılır[21]. Ayrıca farklı normal dağılımlardan iki puanı karşılaştırmamızı sağlar. Standart puan bunu normal bir dağılımdaki puanları standart bir

dağılım haline getiren z-skorlarına dönüştürerek (diğer bir deyişle standartlaştırma) yapar[22]. Basitçe, her bir değerin ortalamadan olan uzaklığının standart sapmaya olan oranı ile bulunur. Aşağıdaki formülde (3.2) belirtilmiştir.

$$X' = \frac{X-\mu}{\sigma} \tag{3.2}$$

Bu yöntemde, ortalama değer ve standart sapma değerleri dikkate alınmıştır. Sistemde kullanılan standart sapmaya istinaden, standart puan da denir. Bu yöntemde ise yapılan işlemler biraz daha farklıdır. Öncelikle bu bant genişliği dizisinden yola çıkarak bu diziye ait iki sayısal değeri önceden hesaplamamız gereklidir.

Bunlardan ilk olanı aritmetik ortalamadır(Arithmethic Mean). Bunun yolu ise dizideki tüm elemanların değerinin toplamını yine aynı dizinin eleman sayısına bölmektir.

Standart sapma ise biraz daha farklı hesaplanmaktadır. Standart sapma, dizideki elemanların dizinin aritmetik ortalamasına olan uzaklığının karelerinin toplamının, dizinin eleman sayısının bir eksiğine bölünmesiyle elde edilen sayının karekköüdür. Biraz açmanız gerekiyorsa, standart sapmayı hesaplamak için; sayıların aritmetik ortalaması hesaplanır. Her sayı aritmetik ortalamadan farklıdır. Bulunan farklılıkların her birinin karesi hesaplanır. Farkların kareleri toplanır. Elde edilen toplam, serinin eleman sayısının bir eksiğine olarak bölünür. Bulunan sayının karekökü alınır. Standart sapma küçükse, ortalamaya yakın bir haritada dağılmışlardır. Tersine, standart sapma büyükse, bunlar serinin ortalamasından uzaktır.

Bu yöntemin kullanılması esnasında bazı değişikliklere gidilmiştir. Normalize edilecek değer aritmetik ortalamadan küçük ise sonuç negatif çıkacaktır. Bu değerleri bağlantı maliyet değerleriyle beraber kullanıp bir sonuç ürettiğimiz için değerlerin negatif olmaması gerekiyor. O sebeple çıkan sonuca 2 ekleyerek tüm sonuçları 2 sayı ötelemiş olduk. Sonucun -2 ve daha küçük olduğu durumlarda ise -1.99 kabul ettik. Çıkan sonuç Bant genişliği ile bağlantı maliyetleri o ilgili bağlantının tercih edilebilirliği üzerinde ters etkiye sahip olduğu için bant genişliğinin normalize edilmiş değeri çarpım şeklinde değil bölme şeklinde hesaba katılır.

Floodlight'ın topolojiye dair sağladığı bilgiler ışığında karar verme anındaki girdilerimizi zenginleştirerek topoloji üzerindeki trafiği daha doğru bir şekilde yönlendirmiş olduk. Yol

maliyetinin yanı sıra gecikme bilgisinin de kullanılması sistemin daha akıllı karar vermesine yol açacaktır. Bu da hizmet kalitesinde artışa sebep olacaktır.

Bu normalleştirme yönteminin nasıl çalıştığını şu örnek ile açıklayabiliriz. Tüm bağlantıların bant genişliğini biliyoruz. Tüm bu bağlantıların bant genişliği değerinden yola çıkarak ortalama değeri ve standart sapmayı yukarıdaki tanımlara göre hesaplayabiliriz. Yine *L1* ve *L2* bağlantılarımız olsun. Bunların maliyetleri ise sırası ile *M1* ve *M2* olsun. *B1* ve *B2* ise bant genişliği değerleri olsun. Bu *B* değişkenlerinin değerleri sırası ile yine 58 ve 64 olsun. Ortalama değer 60, standart sapma da 62 olduğu durumda hesaplar şöyle yapılabilir. *X1* ve *X2* yine normalleşmiş değerlerimiz olacak şekilde;

$$X1 = \frac{58 - 60}{62} + 2$$

$$X2 = \frac{64 - 60}{62} + 2$$

hesaplamasına ulaşılır. X1 için 1,96, X2 değeri için ise 2,06 değeri elde edilir.

Sonuç olarak L1 linkinin tercih edilmesi hesaplanırken M1 maliyet değeri 1,96'ya bölünür. L2 linkinin tercih edilebilirliği hesaplanırken de M2 maliyet değeri 2,06'ya bölünür. Bu da bant genişliği yükseldikçe hesaplanan link bağlantı maliyeti küçülür anlamına gelir. Bu durum o bağlantının tercih edilebilirliğini artıracaktır. Normalleştirme yapılmasının amacı da bu şekildedir[20].

3.3.6. Diğer Çalışmalar

Bu tez kapsamında 2.4 başlığında detayı verilen Açık Artırma yaklaşımlı algoritmanın ve 2.5 başlığında detayı verilen İkili Tırmanma yaklaşımlı algoritmanın kodlaması en baştan yapılmıştır. Bu tezin yazılma zamanında bu algoritmaların Floodlight için herhangi yazılmış kodu yoktu. Bu algoritmaların kodlaması bittikten sonra test sonuçlarını diğer çalışmalar ile kıyaslayarak oluşturduğu değeri görmek amacıyla başka çalışmalar ile kıyaslar yapılmıştır. 4.1 ve 4.2 başlıklarında detaylı olarak da sonuçlar görülebilir. Özetle Açık Artırma ve İkili Tırmanma algoritmaları başka bir çalışmadan alınan Bellman-Ford algoritması ile de kıyaslanmıştır[19].

Bu çalışma kapsamında geliştirilen İkili Tırmanma ve Açık Artırma algoritmaları yıllardır popüler olan ve zaman içinde dinamiklerinin geliştirilmesine fırsat bulan Bellman-Ford algoritmasından bazı topoloji şartlarında daha iyi sonuç vermeyi başarabilmiştir.

Bu çalışmanın kapsamında yeni kodlanan algoritmaların popüler ve iyice bilindik hale gelmesi durumunda hem teoride hem de kodlama pratiğinde daha iyi noktaya geleceği düşünülmektedir. Bu durumda algoritmaların çok daha iyi sonuçlar üreteceğini söyleyebiliriz. İşte bu anlamda bu çalışma diğer çalışmalardan daha farklı anlam ve önem taşımaktadır.

3.3.7. Yazılım Geliştirmede Kullanılan Teknolojiler ve Ürünler

Yazılım üretilirken güncel yazılım üretme yardımcı mekanizmaları kullanılmıştır. Bunlar şöyle gruplandırılabilir.

3.3.7.1.Java

Floodlight projesi Java tabanlı bir uygulamadır. Kullanılan Java sürümü söz konusu olduğunda genellikle Floodlight'ın iki sürümünden bahsedilmektedir. Bunlar son sürüm olan "master" ve Floodlight v1.2 ve altı olarak ayrılmaktadır. Hazır VM Dosyası başlığında belirtilen VM imaj dosyası içerisinde gelen sürüm Floodlight v1.2 olup Java 7 kullanmaktadır. Floodlight'ın "master" adı verilen sürümü yani son sürümü ise Java 8 kullanmaktadır.

Bu proje kapsamında Floodlight'ın üreticileri tarafından verilen ve kolay kurulum sağladığı için tercih edilen şekliyle geliştirmeye başlanmıştır. Bu esnada Java 7(Java 1.7) kullanılmıştır. Ama bu sürüm eski olduğu ve daha günceline gerek duyulduğu için Java sürüm 8'e güncellenmiştir. Proje içerisinde Java 8 ile gelen birçok özellik kullanılmakta olup, bu yeni özelliklerin sayesinde kod kalitesi artmış olup, 20-25 satır ile yazılacak kod blokları 3-4 satıra düşürülmüştür.

3.3.7.2.Intellij IDEA

Intellij IDEA Java'da bilgisayar yazılımı geliştirmek için kullanılan bir tümleşik geliştirme ortamıdır (IDE). JetBrains (resmi adı ile Intellij) şirketi tarafından geliştirilmektedir. Hem Apache 2 lisansı altında yayınlanan topluluk sürümü, hem de sahipli ticari sürümü bulunmaktadır. Her ikisi de ticari amaçlı geliştirmelerde kullanılabilmektedir.

Java geliştiricileri arasında sıklıkla tercih edilen bu IDE'nin en beğenilen IDE olduğu ileri sürülmektedir. Bu proje kapsamında kullanım kolaylığı, esnekliği, hızı, kod geliştirmeye verdiği destek sebebiyle tercih edilmiştir.

3.3.7.3.Ubuntu 18.04

Ubuntu bir bilgisayar işletim sistemidir. Diğer alternatiflerinin birçoğundan farklı olarak ücretsizdir. Ubuntu, tüm dünyada milyonlarca insan tarafından herhangi bir cihazda ücretsiz ve açık yazılım çalıştırmasına müsaade eden önemli bir proje olan Linux'un işletim sistemidir. Yazılım kodunu düzenlemenize, istediğiniz kadar kopya yüklemenize veya programı istediğiniz kadar dağıtmanıza müsaade eder. Kullanmak istediğiniz Ubuntu lisans programı için ödeme yapmanız gerekmez. Ubuntu'yu indirdiğinizde herhangi bir ücret ödemezsiniz ve istediğiniz kadar kullanabilirsiniz ve bu süre boyunca hiçbir şey ödemeniz gerekmez. Bu nedenle, Linux müşterilerine ücretsiz ve serbestçe kullanılabilir bir masaüstü işletim sistemi sunar.[23][24]

Ubuntu'nun teze başlama tarihindeki en güncel versiyonu indirme[23] adresinden indirilip kurulmuştur. Tezin başlangıç zamanındaki türü 18.04 versiyonudur.[23]

4. BULGULAR

4.1. BULUNAN YOL KARŞILAŞTIRMALARI

Bu başlıkta algoritmaların çıktı olarak ürettiği en kısa yol bilgileri kıyaslanacaktır. Bu başlık altında verilecek sonuç şöyle özetlenebilir. Algoritmalara girdi değeri olarak düğümler arası bağlantı maliyetleri ve bağlantıların bant genişliği değerleri programcı tarafından verilmiştir. Verilen girdiler ışığında farklı algoritmaların kıyaslandığını göreceksiniz. Bununla beraber iki farklı topolojinin ve farklı normalleştirme tekniklerinin de farklılaştığı sonuç kümelerinde görülebilecektir. Bu iki farklı topoloji ve normalleştirme teknikleri önceki başlıklarda detaylıca açıklanmıştır. Bu kıstasları bir araya getirdiğimizde altı ayrı senaryo ortaya çıkmaktadır.

Bu testler esnasında tüm algoritmalar aynı girdiler ile çalıştırılmıştır. Yani kıyaslama sonuçlarının sağlıklı olması için aynı girdiler verilmiştir. Bu aynı girdilerin ışığında algoritmaların çıktıları kıyaslanmıştır. Ayrıca tutarlılık açısından her bir senaryo beş sefer tekrarlanmıştır. Sonuçlar ise bu denemelerin ortalamaları alınarak verilmiştir.

Sonuçların değerlendirilmesindeki önemli husus ise şudur. Sayılar genel olarak algoritmaların ürettiği varyasyonları artırmak adına geniş aralıklardan seçilmiştir. O sebeple bir çözüm kümesinin toplam maliyet değerinden daha çok birbirleriyle olan oranı ön plandadır. 4.2 başlığında verilmiş olan “Iperf” komutunun sonuçlarında da aynı durum söz konusudur.

Tüm sonuçların detayları aşağıdaki detay başlıklarında verilmiştir. Her senaryo beş sefer çalıştırıp ortalaması alındığı için ve farklı kriterler ayrı ayrı uygulandığı için sonuçların tutarlı olduğunu söyleyebiliriz. Eğer bir senaryo için en iyi sonucu iki ayrı algoritma tarafından önerilmişse, o iki algoritmanın ikisi için de en iyi sonucu verdi şeklinde değerlendirme yapılmıştır. Bu gözle baktığımızda Dijkstra algoritmasının toplamda 4 kere en iyi sonucu verdiğini görebiliriz. Diğer algoritmalar yani Bellman Ford, Açık Artırma ve İkili Tırmanma algoritmaları ise aynı sayıda en iyi sonucu önermişlerdir. Bu sayı da 2’dir. Bu veriler analiz edildiğinde şu görülmüştür. Algoritmaların herhangi bir normalizasyon yöntemi ile birlikte çalıştığında yalın hallerine göre farklı performans gösterdiği saptanmıştır.

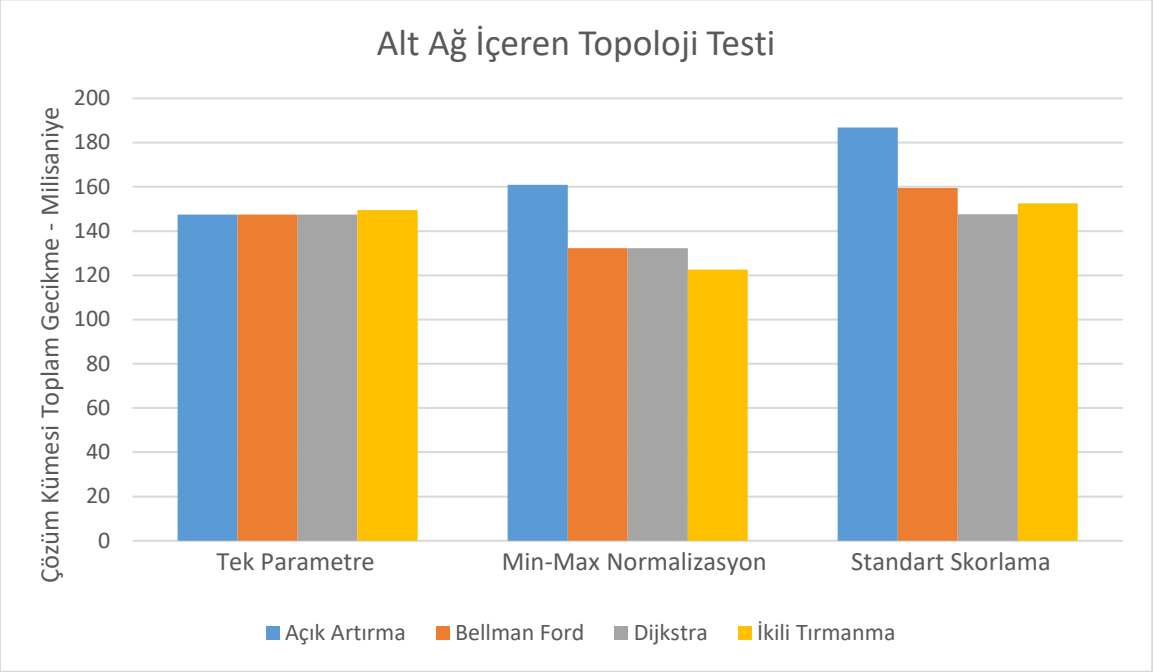
Dijkstra algoritması normalizasyon yöntemlerinden herhangi biriyle çalıştığı durumda diğer algoritmalardan bir adım önde değerler üretmiştir. İkili Tırmanma ve Açık Artırma

algoritmaları daha az bilinen algoritmalar olmasına rağmen Bellman Ford algoritması ise aynı sayıda en iyi sonucu öneren algoritmalar olabilmişlerdir.

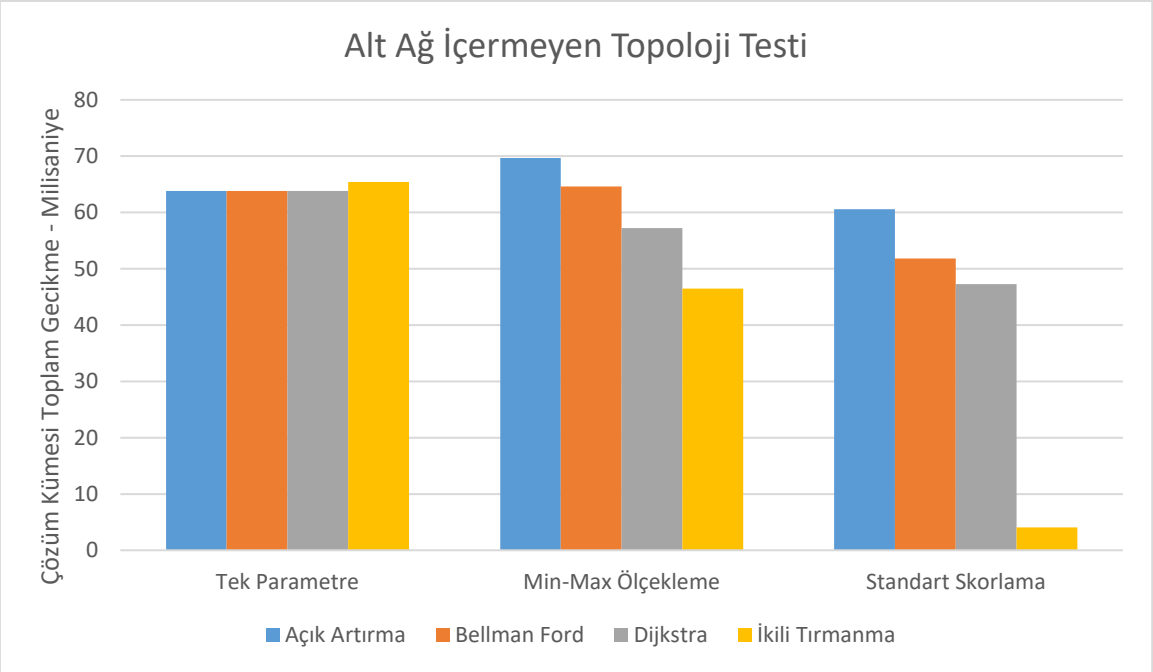
Sonuçların analizlerini yaptıktan sonra verilmesi gereken bir detay daha mevcuttur. Dijkstra ve Bellman Ford algoritmaları bilinen algoritmalar oldukları için zaman içerisinde hem programcılar tarafından hem de teorik olarak bu işle ilgilenen insanlar tarafından daha çok optimize edildiği düşünülmektedir. Öte yandan İkili Tırmanma ve Açık Artırma algoritmaları çok fazla optimize edilme fırsatına sahip olmadıklarını düşünülmektedir. Çünkü bu çalışmada kodlanan algoritmalar İkili Tırmanma ve Açık Artırma yaklaşımlı algoritmalardır. Yani ortada tek bir İkili Tırmanma ve Açık Artırma algoritması yok. Aynı temel kuralları uygulayarak bu algoritmaların daha iyi sürümleri yazılabilir. Bu şekilde optimize edildiği durumda sonuçların değişebileceği düşünülmektedir.

Aşağıdaki Şekil 4.1'de verilen çubuk grafiğinde 3 ayrı senaryo için elde edilen sonuçlar görülebilir. Gerekli detaylar ilgili senaryoların başlıklarında verilmiştir. Özetle; alt ağ içeren bir topolojide koşulan testlerin sonucunda, algoritmaların ürettiği çıktılara ait toplam maliyet bilgileri verilmiştir. Senaryolarda değişenler sırasıyla tek parametre ile çalışması, Min-Max Ölçekleme metodunu kullanması ve Standart Skorlama kullanmasıdır.

Ayrıca, Şekil 4.2'de verilen çubuk grafiğinde 3 ayrı senaryo için elde edilen sonuçlar görülebilir. Gerekli detaylar ilgili senaryoların başlıklarında verilmiştir. Özetle; alt ağ içermeyen bir topolojide koşulan testlerin sonucunda, algoritmaların ürettiği çıktılara ait toplam maliyet bilgileri verilmiştir. Senaryolarda değişenler sırasıyla tek parametre ile çalışması, Min-Max Ölçekleme metodunu kullanması ve Standart Skorlama kullanmasıdır.



Şekil 4.1: Alt Ağ İçeren Topolojilerde Gerçekleştirilen Testler



Şekil 4.2: Alt Ağ İçermeyen Topolojilerde Gerçekleştirilen Testler

4.1.1. Alt Ağ İçeren Topoloji Testleri

4.1.1.1.Algoritmaların Tek Parametreli Çalıştırılması Testi

Bu senaryoda topoloji olarak Şekil 3.2'de verilen ve alt ağ içeren topoloji kullanılmıştır. Algoritmalar sadece bağlantı maliyeti olacak şekilde tek parametre girdisi ile çalıştırılmıştır. Sonuçlar milisaniye cinsinden olup toplam gecikmeyi göstermektedir.

Bu senaryoda en iyi sonuç üç algoritma tarafından önerilmiştir. Bunlar Dijkstra, Bellman Ford ve Açık Artırma algoritmalarıdır. Üretilen çıktıdaki toplam maliyet değeri 147,4'tür. İkinci en iyi sonucu yani en kötü sonucu ise 149,5 değeri ile İkili Tırmanma algoritması üretmiştir. Tüm senaryoları bir araya getirerek bir yorum ortaya çıkartmak için hangi senaryoda hangi algoritmalar en iyi sonucu vermiş bilgisi kaydediliyor. Bu senaryo için Dijkstra, Bellman Ford ve İkili Tırmanma algoritmaları en iyi sonucu üreten algoritmalar olarak belirlenmiştir.

4.1.1.2. Min-Max Ölçekleme Yöntemi Kullanılarak Bant Genişliği Değerinin Hesaba Katılması Testi

Bu senaryoda topoloji olarak Şekil 3.2'de verilen ve alt ağ içeren topoloji kullanılmıştır. Algoritmalar bağlantı maliyetinin yanında bant genişliği değerinin de hesaba katılması ile çalışacaktır. Bu senaryo için normalizasyon yöntemi olarak Min-Max Ölçekleme kullanılmıştır. Sonuçlar milisaniye cinsinden olup toplam gecikmeyi göstermektedir.

Bu senaryo için ise en iyi sonucu ilk senaryonun tam aksine ikili Tırmanma algoritması 122,6 değeri ile birlikte üretmiştir. 160,8 değeri ile en az başarılı sonuç Açık Artırma algoritmasından önerilmiştir. İkinci en iyi sonuç 132,2 değeriyle Dijkstra'dır. Bellman Ford algoritması ise ürettiği sonuç itibariyle toplam maliyet bilgisi 132,3'tür. Bu da bu algoritmayı en iyi üçüncü sonuç yapar.

4.1.1.3.Standart Skorlama Yöntemi Kullanılarak Bant Genişliği Değerinin Hesaba Katılması Testi

Bu senaryoda topoloji olarak Şekil 3.2'de verilen ve alt ağ içeren topoloji kullanılmıştır. Algoritmalar bağlantı maliyetinin yanında bant genişliği değerinin de hesaba katılması ile çalışacaktır. Bu senaryo için normalizasyon yöntemi olarak "standart skorlama" kullanılmıştır. Sonuçlar milisaniye cinsinden olup toplam gecikmeyi göstermektedir.

Bu senaryo için ise en iyi sonucu 147,6 değeri ile birlikte Dijkstra üretmiştir. 186,8 değeri ile en az başarılı sonuç Açık Artırma algoritmasından önerilmiştir. İkinci en iyi sonuç 152,5 değeriyle İkili Tırmanma algoritmasıdır. Bellman Ford algoritması ise ürettiği sonuç itibariyle toplam maliyet bilgisi 159,4'tür. Bu da bu algoritmayı en iyi üçüncü sonuç yapar.

4.1.2. Alt Ağ İçermeyen Topoloji Testleri

4.1.2.1. Algoritmaların Tek Parametreli Çalıştırılması Testi

Bu senaryo ise birinci senaryo ile topoloji dışında aynı özelliklere sahiptir. Bu durum için herhangi bir alt ağ içermeyen ve detayı Şekil 3.1'de verilen topoloji kullanılmıştır. Algoritmalar sadece bağlantı maliyeti olacak şekilde tek parametre girdisi ile çalıştırılmıştır. Sonuçlar milisaniye cinsinden olup toplam gecikmeyi göstermektedir.

Bu senaryoda en iyi sonuç üç algoritma tarafından önerilmiştir. Bunlar Dijkstra, Bellman Ford ve Açık Artırma algoritmalarıdır. Üretilen çıktıdaki toplam maliyet değeri 63,8'dır. İkinci en iyi sonucu ama aynı zamanda en az başarılı sonucu ise 65,4 değeri ile İkili Tırmanma algoritması üretmiştir. Bu senaryo için Dijkstra, Bellman Ford ve İkili Tırmanma algoritmaları en iyi sonucu üreten algoritmalar olarak belirlenmiştir.

4.1.2.2. Min-Max Ölçekleme Yöntemi Kullanılarak Bant Genişliği Değerinin Hesaba Katılması Testi

Bu senaryoda topoloji olarak Şekil 3.1'de verilen ve alt ağ içermeyen topoloji kullanılmıştır. Algoritmalar bağlantı maliyetinin yanında bant genişliği değerinin de hesaba katılması ile çalışacaktır. Bu senaryo için normalizasyon yöntemi olarak Min-Max Ölçekleme kullanılmıştır. Sonuçlar milisaniye cinsinden olup toplam gecikmeyi göstermektedir.

Bu senaryo için ise en iyi sonucu 46,5 değeri ile birlikte İkili Tırmanma algoritması üretmiştir. 69,7 değeri ile en az başarılı sonuç Açık Artırma algoritmasından önerilmiştir. İkinci en iyi sonuç 57,2 değeriyle Dijkstra algoritmasıdır. Bellman Ford algoritması ise ürettiği sonuç itibariyle toplam maliyet bilgisi 64,6'dır. Bu da bu algoritmayı en iyi üçüncü sonuç yapar.

4.1.2.3.Standart Skorlama Yöntemi Kullanılarak Bant Genişliği Değerinin Hesaba Katılması Testi

Bu senaryoda topoloji olarak Şekil 3.1'de verilen ve alt ağ içermeyen topoloji kullanılmıştır. Algoritmalar bağlantı maliyetinin yanında bant genişliği değerinin de hesaba katılması ile çalışacaktır. Bu senaryo için normalizasyon yöntemi olarak "standart skorlama" kullanılmıştır. Sonuçlar milisaniye cinsinden olup toplam gecikmeyi göstermektedir.

Bu senaryo için en iyi sonuç İkili Tırmanma algoritmasından gelmiştir. Üretilen yolların ortalama maliyet değeri 40,5'tir. İkinci en iyi sonuç ise 47,3 değeri ile Dijkstra algoritması tarafından önerilmiştir. Üçüncü en iyi sonuç ise 51,8 değeri ile Bellman Ford algoritmasıdır. Döndüncü en iyi sonucumuz ise 60,6 değeri ile Açık artırma algoritmasıdır.

4.2. AĞ YOĞUNLUK SONUÇLARI

Bu çalışma kapsamında da algoritmalar Floodlight projesi içerisinde geliştirilmiştir. Daha önce anlatıldığı üzere Mininet uygulaması kontrolör olarak Floodlight'ı kullanır. Bunun sonucu olarak Mininet'e ait "Iperf" komutu kullanılmıştır. Mininet kontrolör olarak Floodlight'ı kullandığı için bu "Iperf" komutunu kullanarak algoritmalarımızı test edebildik.

"Iperf" komutunu özetlemek gerekirse; komutta belirtilen süre içerisinde ve düğümler arasında trafik oluşturmaya yarar. Genel olarak performans değerlendirilmesi yapılırken kullanılan bir komuttur.[25] Biz de testlerimizce TCP trafiği oluşturmak için "Iperf" komutunu kullanacağız.

Ayrıca unutulmamalıdır ki bu testlerin sonunda çıkan sayılar donanım bağımlıdır. Bilgisayar kaynakları farklı olan bir makinede çalıştıklarında tamamen farklı sayılar üretebilirler. Bu sebeple sayıların direkt olarak kendisinden daha çok birbirleri ile olan oranları önemlidir. Tüm testler aynı şartlarda yapıldığı için birbirleri ile olan oranları bize anlamlı veriler verecektir.

Yine bu ağ yoğunluk testlerimizi yaparken de farklı kıstasları bir araya getirdiğimizde sekiz ayrı senaryo ortaya çıkmaktadır. Ağ yoğunluk testleri sadece Şekil 3.2'de verilen ve alt ağ içeren topoloji üzerinde koşulmuştur. Tüm senaryolar aynı topolojide çalıştığı için topoloji kıstası bu senaryolarda mevcut değildir.

Tüm testler 15 saniye TCP trafiği üretilerek gerçekleştirilmiştir. Anlık sonuçların ekrana yazılma sıklığı 1 saniye olarak belirlenmiştir. Tüm sonuçlara ait ekran görüntüleri verilmiştir.

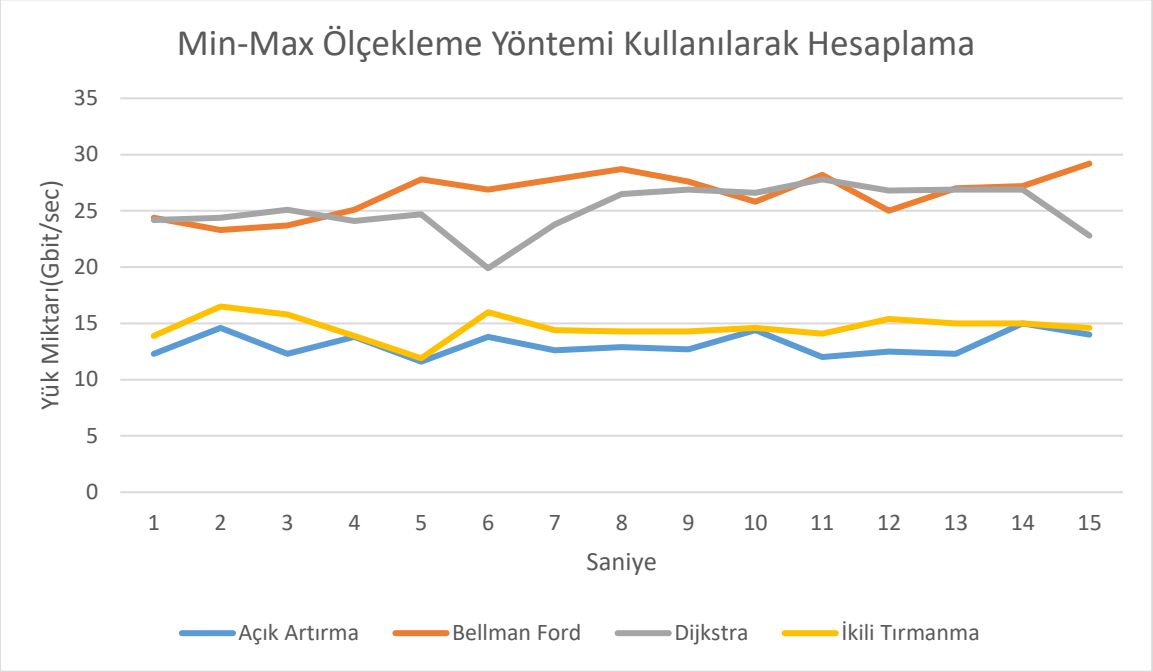
Bu ekran görüntülerinde en alttaki satır 15 saniyenin sonundaki ortalama değer olup diğer üstteki satırlar ise her bir saniye orataya çıkan anlık sonuçlardır.

Her bir senaryo için gerekli detaylar ilgili senaryolara ait başlıklar altında verilmiştir. Daha önce de belirtildiği gibi her bir sonuç için değerlerin kendisine değil, oranlarına ve büyük veya küçük olduğuna bakılmalıdır. Çünkü sayıların kendisi donanım ve işlemcinin o anlık yüküne bağlı olarak değişebilmektedir. Tüm bu bilgilerin ışığında Dijkstra ve Bellman Ford algoritmalarının her iki normalizasyon metodu kullanılan sürümlerinde de birbirine yakın sonuçlar verdiği gözlemlenmiştir. Bu değerler diğer iki algoritma yani İkili Tırmanma ve Açık Artırma algoritmalarının ürettiği sonuçların ortalamasından yüksektir. Dijkstra ve Bellman Ford algoritmalarının ortalama sonuçları saniyede 25 Gigabit seviyelerindeyken öte yandan İkili Tırmanma ve Açık Artırma algoritmaları için bu sonuç saniyede 15 Gigabit ortalamalarında gezmektedir.

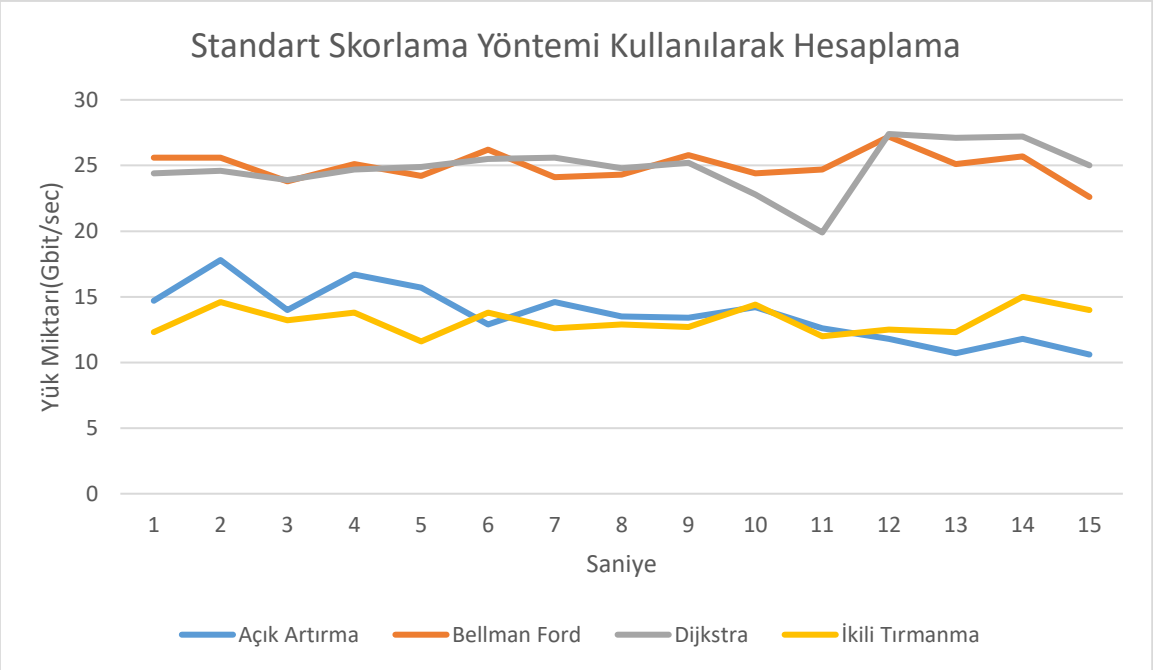
Sonuç olarak, 4.1 başlığında da belirtildiği gibi Dijsktra ve Bellman-Ford algoritmaları daha bilnen algoritmalar olduğu için zaman içinde daha çok optimize edilmiştir. Öte yandan İkili Tırmanma ve Açık Artırma algoritmalarının bu çalışmada incelenen yaklaşımları ise optimizasyona daha çok açıktır. Bundan dolayı adı daha çok bilinen Dijkstra ve Bellman Ford algoritmalarının kısmen daha iyi sonuç verdiği gözlemlenmiştir.

Şekil 4.3’de Min-Max Ölçekleme yöntemi kullanılarak 15 saniye boyunca Açık Artırma, Bellman Ford, Dijsktra ve İkili Tırmanma algoritmalarının yük testindeki gösterdikleri performanslar grafiğe dökülmüştür.

Ayrıca, şekil 4.4’de de Standart Skorlama yöntemi kullanılarak 15 saniye boyunca Açık Artırma, Bellman Ford, Dijsktra ve İkili Tırmanma algoritmalarının yük testindeki gösterdikleri performanslar grafiğe dökülmüştür. Grafikte sağlanan verilerin detaylarına ilgili senaryo başlığı altından ulaşılabilir.



Şekil 4.3: Min-Max Ölçekleme Yöntemi Sonuçları

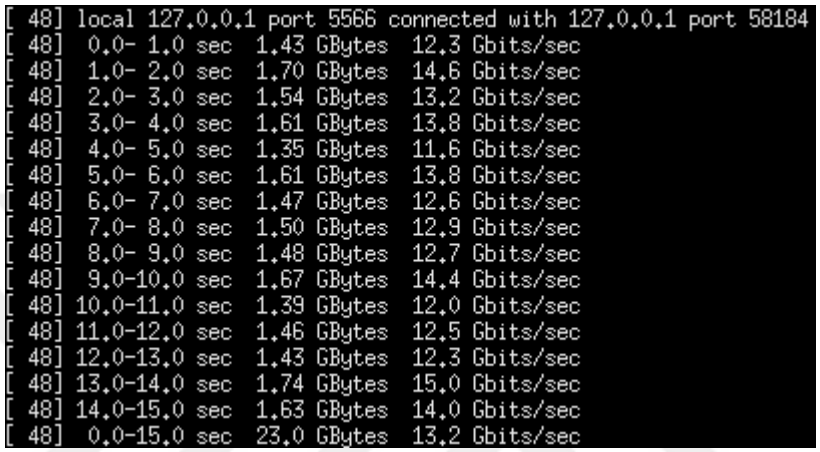


Şekil 4.4: Standart Skorlama Yöntemi Sonuçları

4.2.1. Min-Max Ölçekleme Testleri

4.2.1.1.Açık Artırma Testi

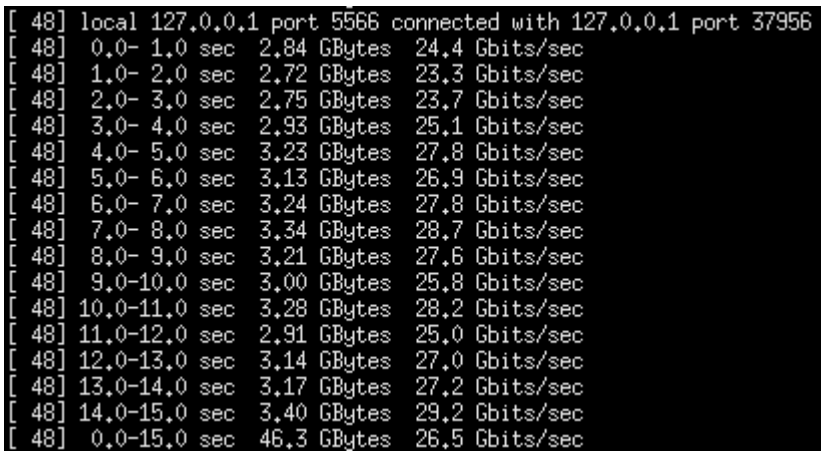
Bu senaryoda performans testi yapılırken algoritma olarak Açık Artırma kullanılmıştır. Ayrıca bu senaryo için normalizasyon yöntemi olarak Min-Max Ölçekleme kullanılmıştır. Ekran görüntüsünden de görüleceği üzere saniyede 13.2 Gigabitlik aktarım hızı ortaya çıkmış.



Şekil 4.5: Açık Artırma Algoritması Performans Sonucu

4.2.1.2.Bellman-Ford Testi

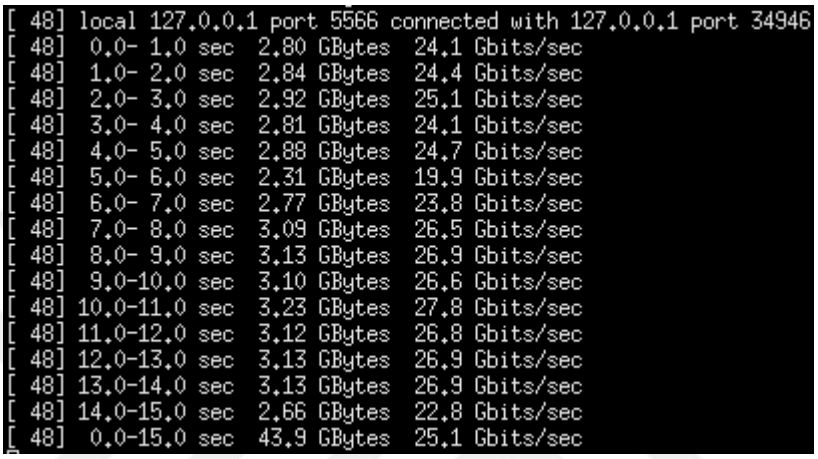
Bu senaryoda performans testi yapılırken algoritma olarak Bellman-Ford kullanılmıştır. Ayrıca bu senaryo için normalizasyon yöntemi olarak Min-Max Ölçekleme kullanılmıştır. Ekran görüntüsünden de görüleceği üzere saniyede 26.5 Gigabitlik aktarım hızı ortaya çıkmış.



Şekil 4.6: Bellman Ford Algoritması Performans Sonucu

4.2.1.3.Dijkstra Testi

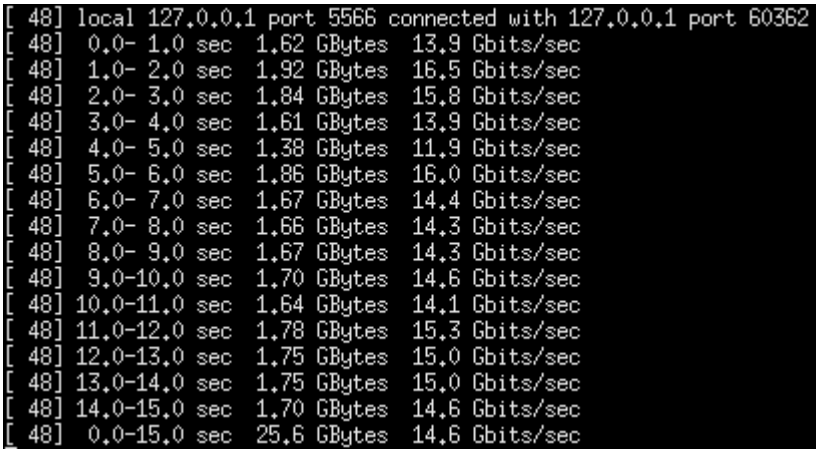
Bu senaryoda performans testi yapılırken algoritma olarak Dijkstra kullanılmıştır. Ayrıca bu senaryo için normalizasyon yöntemi olarak Min-Max Ölçekleme kullanılmıştır. Ekran görüntüsünden de görüleceği üzere saniyede 25.1 Gigabitlik aktarım hızı ortaya çıkmış.



Şekil 4.7: Dijkstra Algoritması Performans Sonucu

4.2.2. İkili Tırmanma Testi

Bu senaryoda performans testi yapılırken algoritma olarak İkili Tırmanma kullanılmıştır. Ayrıca bu senaryo için normalizasyon yöntemi olarak Min-Max Ölçekleme kullanılmıştır. Şekil 4.8'deki ekran görüntüsünden de görüleceği üzere saniyede 14.6 Gigabitlik aktarım hızı ortaya çıkmış.

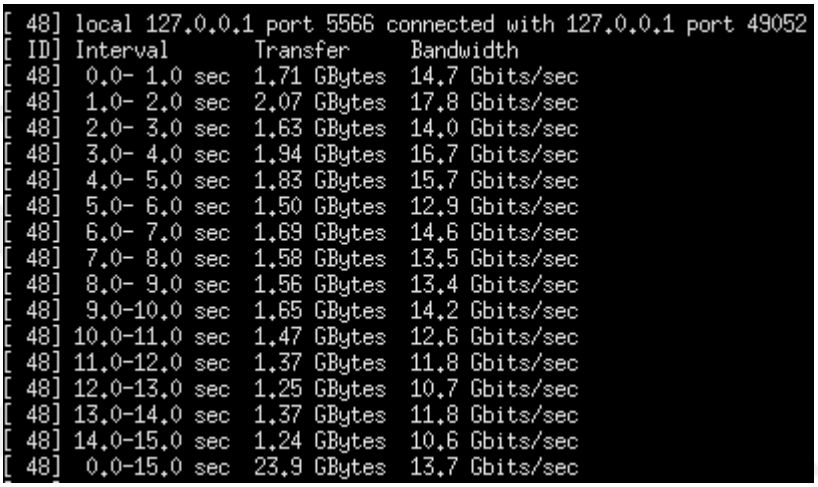


Şekil 4.8: İkili Tırmanma Algoritması Performans Sonucu

4.2.3. Standart Skorlama Testleri

4.2.3.1.Açık Artırma Testi

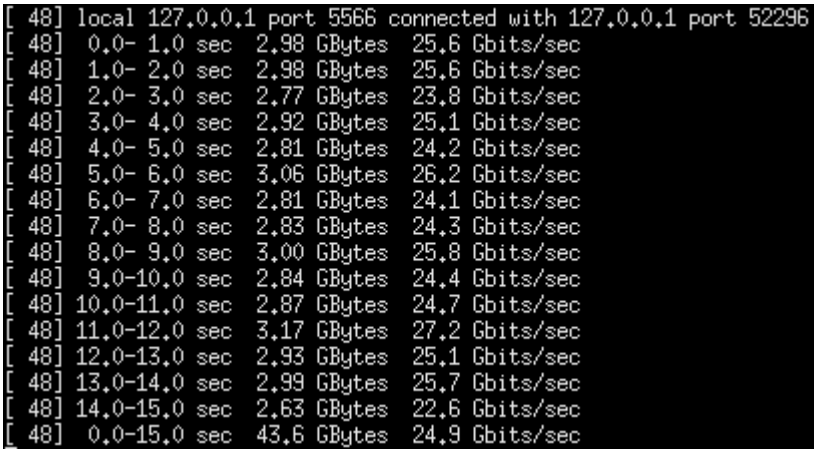
Bu senaryoda performans testi yapılırken algoritma olarak Açık Artırma kullanılmıştır. Ayrıca bu senaryo için normalizasyon yöntemi olarak standart skorlama kullanılmıştır. Şekil 4.9'deki ekran görüntüsünden de görüleceği üzere saniyede 13.7 Gigabitlik aktarım hızı ortaya çıkmış.



Şekil 4.9: Açık Artırma Algoritması Performans Sonucu

4.2.3.2.Bellman-Ford Testi

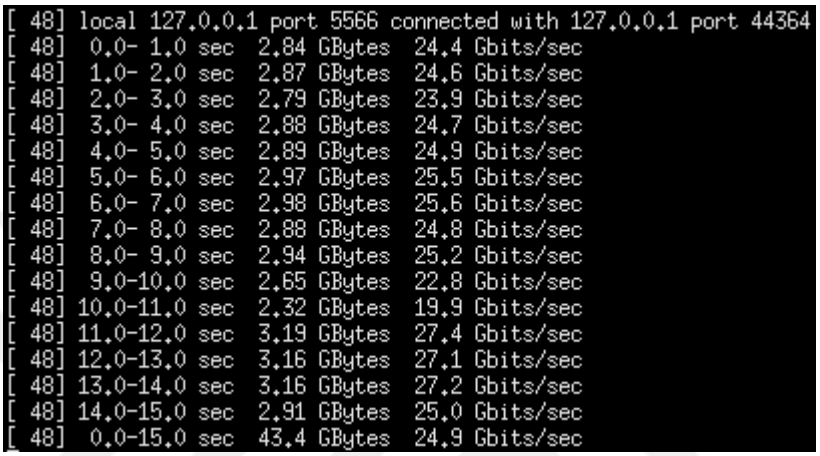
Bu senaryoda performans testi yapılırken algoritma olarak Bellman Ford kullanılmıştır. Ayrıca bu senaryo için normalizasyon yöntemi olarak standart skorlama kullanılmıştır. Şekil 4.10'da görüleceği üzere saniyede 24.9 Gigabitlik aktarım hızı ortaya çıkmış.



Şekil 4.10: Bellman Ford Algoritması Performans Sonucu

4.2.3.3.Dijkstra Testi

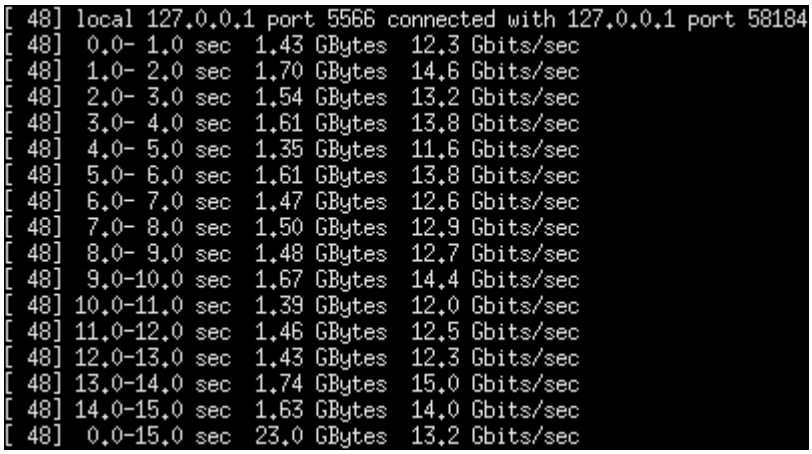
Bu senaryoda performans testi yapılırken algoritma olarak Dijkstra kullanılmıştır. Ayrıca bu senaryo için normalizasyon yöntemi olarak standart skorlama kullanılmıştır. Şekil 4.11'den de görüleceği üzere saniyede 24.9 Gigabitlik aktarım hızı ortaya çıkmış.



Şekil 4.11: Dijkstra Algoritması Performans Sonucu

4.2.4. İkili Tırmanma Testi

Bu senaryoda performans testi yapılırken algoritma olarak İkili Tırmanma kullanılmıştır. Ayrıca bu senaryo için normalizasyon yöntemi olarak standart skorlama kullanılmıştır. Şekil 4.12'den de görüleceği üzere saniyede 13,2 Gigabitlik aktarım hızı ortaya çıkmış.



Şekil 4.12: Dijkstra Algoritması Performans Sonucu

5. TARTIŞMA VE SONUÇ

YTA'lar günümüzde gittikçe önemini artırmaktadır. Artık bilgisayarlar birbirine bağlı şekilde her gün daha çok hayatımıza girmektedirler. Bunlar birbirine bağlı bilgisayarlar olduğundan bilgisayar ağlarının da önemi aynı şekilde artmaktadır. Bunun sonucu olarak bu ağlardkai verilerin işlenmesi her geçen gün daha önemli hale gelmektedir. YTA'lar da tam bu noktada öne çıkmaktadır. Hem büyüyen verinin boyutuyla başa çıkmak noktasında hem de sunduğu kolaylıklar ve fonksiyon setinden dolayı çok işlevsel bir modeldir. Bu çalışma ile birlikte YTA'ların önemli bir kontrolör uygulama projesi olan Floodlight'ı iyileştirmek için yapılmıştır. Floodlight projesi normalde Dijsktra algoritmasını kullanmaktadır. Biz bu çalışma ile birlikte İkili Tırmanma ve Açık Artırma algoritmalarını Floodlight projesinin içinde geliştirerek katkıda bulunduk. Bir katkı da tüm bu algoritmaların karar verme mekanizmasını zenginleştirerek yapılmıştır. Algoritmalar ana karar verme aşamasında sadece tek parametre ile çalışırken buna ikinci parametreyi ekleyerek zenginleştirme yapılmıştır. Bununla birlikte zenginleştirme esnasında veriler normalleştirilirken iki ayrı normalleştirme yöntemi kullanılmıştır. Bunlar Min-Max Ölçekleme ve standart skorlama yöntemleridir. Bunların önemli bir noktası sadece yeni geliştirilen İkili Tırmanma ve Açık Artırma algoritmalarına uygulanmamış, halihazırda kullanılan Dijsktra algoritmasına da entegre edilmiş olmasıdır. Sonuç ve kıyas kümesini daha da zenginleştirmek için başka bir projeden Floodlight için yazılmış olan Bellman Ford algoritmasını da kullanılmıştır. Böylece zengin bir veri seti oluşturulmuştur. Toplam 14 senaryoda 28 kıyaslama yapılmıştır.

Algoritmaların özelinde kritik yapacak olursak şunlar söylenebilir. Hem 4.1 numaralı başlıkta hem de 4.2 numaralı başlıklarda kıyaslar gerekli detaylarla verilmiştir. Buralarda da özetle söylendiği gibi algoritmalardan bazı çıkarımlarda bulunma fırsatına sahibiz. İlk olarak söylemek gerekirse daha çok bilinen algoritmaların daha başarılı olduğu gözlemlenmiştir. Yani testlerin birçoğunda Dijsktra algoritması daha başarılı görünse de Bellman Ford algoritması da Dijsktra algoritmasına çok yakın ve iyi bir performans sergilemiştir. Öte yandan yeni kodlamış olan İkili Tırmanma yaklaşımlı ve Açık Artırma yaklaşımlı iki algoritmanın başarılı olduğu ve en iyi sonucu verdiği senaryolar olmakla birlikte genel istatistiksel verilere bakıldığında biraz daha az iyi maliyete sahip çıktılar ürettiği görülmüştür.

Biraz daha bu konunun üzerinde durulması önem arz etmektedir. Zaman içinde ismi daha çok bilinen algoritmalar daha çok kişinin dikkatini ve ilgisini çekmiş olması normaldir. Bu ilgi ve dikkatin sonucu olarak insanlar zaman içinde bu algoritmaları daha çok optimize etmişlerdir. Basit bir Google araştırması ile birlikte bu Dijsktra ve Bellman Ford algoritmalarının onlarca farklı uygulamasını neredeyse tüm yazılım dillerinde yazılmış sürümünü bulabilirsiniz. Testlerde Dijkstra ve Bellman Ford algoritmasının başarılarını bundan kaynaklı olduğu düşünülmektedir.

Öte yandan İkili Tırmanma ve Açık Artırma yaklaşımlı iki algoritmanın ilk sürümü yazılmıştır. Burada özellikle algoritmaların çalışma zamanı noktasında programcıya dayalı olarak iyileştirmeye açık noktaları bulunabilir. Bu kodlama konusunun da dışında teorik olarak bu algoritmalar zaman içinde optimize edilirse bu çalışmanın sonucunda ortaya çıkan tablonun değişebileceği düşünülmektedir.

Eğer bu yapılan çalışma daha da ileri götürülecekse bunun en önemli ayağı bu testler için özelleştirilmiş bilgisayarlar kullanılabilir. Özelleştirilmiş bilgisayarlardan kastımız daha güçlü kaynaklara sahip bilgisayarlardır. Böylece daha fazla düğüm sayısına sahip topolojiler üzerinde testler koşulabilir. Bunun sonucu olarak daha detaylı sonuçlar alınabilir. Testlerin daha detaylı olması her zaman bu tarz çalışmalar için önem arz etmektedir. Ortaya çıkan performansın ölçülmesi yapılan çalışmayı ortaya koyma açısından çok önemlidir. Tüm algoritmaların testleri için şartlar aynı tutulmuştur. Fakat testler şahsi bilgisayarlarda koşulduğu için tüm testlerde performans bir kısıt olmuştur.

Bu tez kapsamında yazılan kodlara https://github.com/tunahan/floodlight-tunahan adresindenki repository'den erişim sağlanabilir. Bu projenin daha ileri götürülmesi durumunda bu adresten kodlar çekilerek geliştirmesine başlanabilir. Çalışmanın durumuna göre bu repository gizli olabilir. Böyle bir durumda CV kısmındaki mail adresine mail attığınız taktirde paylaşım sağlanacaktır.

KAYNAKLAR

[1] Öner, D., 2004, Bilgisayar Ağları, Papatya Yayıncılık, Istanbul.

[2] Bonaventure, O., 2011, Computer Networking: Principles, Protocols and Practice, The Saylor Foundation.

[3] Bird, D. Harwood, M., Network+ Training Guide

[4] Akbaş, M.F. ve Karaarslan, E. ve Güngör, C.,2016, A Preliminary Survey on the Security of Software-Defined Networks, International Journal Of Applied Mathematics, Electronics and Computers, 4 (Special Issue), 184-189.

[5] Goransson, P, Black, C., 2014, Software Defined Network A Comprehensive Approach, Morgan Kaufmann

[6] Wang K. Floodlight documentation. Project floodlight; docs.projectfloodlight.org/display/floodlightcontroller.

[7] Bertsekas, D.(1998) Network Optimization: Continuous and Discrete Models

[8] Bertsekas, D.(1991) Auction Algorithms for Network Flow Problems: A Tutorial Introduction

[9] Bertsekas, D.(1991) An Auction Algorithms for Shortest Paths

[10]Becker, R. Fickert, M. , Karrenbauer, A. A Novel Dual Ascent Algorithm for Solving the Min-Cost Flow Problem

[11]Wong, R.T. (1984). A dual ascent approach for steiner tree problems on a directed graph. Mathe- matical Programming, 28:271–287.

[12]Goemans, M.X. and Williamson, D. (1996). The primal-dual method for approximation algorithms and its application to network design problems. In Approximation algorithms for NP-hard problems, pages 144–191. PWS Publishing Co., Boston, MA, USA.

[13] M. Santos, L.M.A Drummond, E. Uchoa Distributed dual ascent algorithm for Steiner problem in Networks, (2007), SBRC 2007, pp. 381-392

[14] Github, https://github.com/mouton5000/DSTAlgoEvaluation

[15] Mininet, http://mininet.org/overview/

[16]Floodlight,https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/pages/1343544/Installation+Guide#InstallationGuide-Linux>

[17] Apache, https://ant.apache.org/bindownload.cgi

[18] Ölmez, S. Yazılım Tanımlı Ağlarda Genetik Algoritma Tabanlı Multimetrik Yönlendirme. Yüksek Lisans Tezi, İstanbul Üniversitesi, 2019.

[19] Özbek, U., Yiltas-Kaplan, D., Zengin, A., Ölmez, S. “Traffic analysis of a software defined network, 1st International Congress on Engineering Technologies (EngiTek 2020), 16-18 June 2020, Irbid, Jordan, Accepted.

[20] Charu C., 2015, Aggarwal, Data Mining: The Textbook, Springer International Publishing Switzerland.

[21] Saranya, C., Manikandan, G.(2013) A Study on Normalization Techniques for Privacy Preserving Data Mining

[22] Standard Score, https://statistics.laerd.com/statistical-guides/standard-score.php

[23] Ubuntu, https://ubuntu.com/

[24] Mediaclick, https://www.mediaclick.com.tr/blog/ubuntu-nedir

[25] How to use iperf over mininet, http://csie.nqu.edu.tw/smallko/sdn/iperf_mininet.htm

EKLER

ÖZGEÇMİŞ

Kişisel Bilgiler	
Adı Soyadı	Hilmi Tunahan İLHAN
Doğum Yeri	Ankara
Doğum Tarihi	30.12.1991
Uyruğu	☑ T.C. ❒ Diğer:
Telefon	+90 536 515 99 06
E-Posta Adresi	Tunahann40@gmail.com
Web Adresi	



Eğitim Bilgileri	
Lisans	
Üniversite	Kocaeli Üniversitesi
Fakülte	Mühendislik
Bölümü	Bilgisayar Mühendisliği
Mezuniyet Yılı	01.07.2013

Yüksek Lisans	
Üniversite	İstanbul Üniversitesi-Cerrahpaşa
Enstitü Adı	Lisansüstü Eğitim Enstitüsü
Anabilim Dalı	Bilgisayar Mühendisliği Anabilim Dalı
Programı	Bilgisayar Mühendisliği

Yayınlar
Ilhan, H. T., Yiltas-Kaplan, D., “Changing Routing Module in Floodlight”, Proc. of the 2nd International Conference on Electrical, Communication and Computer Engineering (ICECCE), June 12-13, Istanbul, Turkey, Accepted.