KARADENİZ TEKNİK ÜNİVERSİTESİ
MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

YAZILIM TANIMLI AĞLARIN ANALİZİ İÇİN BİR EMÜLASYON ARACININ GELİŞTİRİLMESİ

BİTİRME PROJESİ

Oğuzhan ÖZBEK
Ümit GÖRÜR

2018-2019 BAHAR DÖNEMİ

KARADENİZ TEKNİK ÜNİVERSİTESİ
MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

YAZILIM TANIMLI AĞLARIN ANALİZİ İÇİN BİR EMÜLASYON ARACININ GELİŞTİRİLMESİ

BİTİRME PROJESİ

Oğuzhan ÖZBEK
Ümit GÖRÜR

2018-2019 BAHAR DÖNEMİ

IEEE Etik Kuralları
IEEE Code of Ethics

Mesleğime karşı şahsi sorumluluğumu kabul ederek, hizmet ettiğim toplumlara ve üyelerine en yüksek etik ve mesleki davranışta bulunmaya söz verdiğimi ve aşağıdaki etik kurallarını kabul ettiğimi ifade ederim:

1. Kamu güvenliği, sağlığı ve refahı ile uyumlu kararlar vermenin sorumluluğunu kabul etmek ve kamu veya çevreyi tehdit edebilecek faktörleri derhal açıklamak;
2. Mümkün olabilecek çıkar çatışması, ister gerçekten var olması isterse sadece algı olması, durumlarından kaçınmak. Çıkar çatışması olması durumunda, etkilenen taraflara durumu bildirmek;
3. Mevcut verilere dayalı tahminlerde ve fikir beyan etmelerde gerçekçi ve dürüst olmak;
4. Her türlü rüşveti reddetmek;
5. Mütenasip uygulamalarını ve muhtemel sonuçlarını gözeterek teknoloji anlayışını geliştirmek;
6. Teknik yeterliliklerimizi sürdürmek ve geliştirmek, yeterli eğitim veya tecrübe olması veya işin zorluk sınırları ifade edilmesi durumunda ancak başkaları için teknolojik sorumlulukları üstlenmek;
7. Teknik bir çalışma hakkında yansız bir eleştiri için uğraşmak, eleştiriyi kabul etmek ve eleştiriyi yapmak; hatları kabul etmek ve düzeltmek; diğer katkı sunanların emeklerini ifade etmek;
8. Bütün kişilere adilane davranmak; ırk, din, cinsiyet, yaş, milliyet, cinsi tercih, cinsiyet kimliği, veya cinsiyet ifadesi üzerinden ayırımcılık yapma durumuna girişmemek;
9. Yanlış veya kötü amaçlı eylemler sonucu kimsenin yaralanması, mülklerinin zarar görmesi, itibarlarının veya istihdamlarının zedelenmesi durumlarının oluşmasından kaçınmak;
10. Meslektaşlara ve yardımcı personele mesleki gelişimlerinde yardımcı olmak ve onları desteklemek.

IEEE Yönetim Kurulu tarafından Ağustos 1990’da onaylanmıştır.

ÖNSÖZ

Yazılım Tanımlı Ağlar için Emülasyon Aracının Geliştirilmesi adlı bu projede, yeni bir teknoloji olan Software Defined Network için geliştirilmiş olan Mininet uygulamasının daha kullanışlı hale getirilmesi amacıyla projeye yeni özellikler eklenmesi amaçlanmıştır.
Bu kapsamda projeye yeni özellikler eklenmiş ve proje daha kullanışlı hale getirilmiştir.

Tez çalışmamızda; planlamada, araştırmada, oluşumunda ve yürütülmesinde ilgi ve desteğini esirgemeyen, engin bilgi ve tecrübelerinden yararlandığımız, yönlendirme ve bilgilendirmeleriyle çalışmamızı bilimsel temeller ışığında şekillendiren sayın Dr. Öğr. Üyesi Selçuk CEVHER hocamıza ve bu günlere gelmemize vesile olan ailelerimize sonsuz teşekkürlerimizi sunarız.

Oğuzhan ÖZBEK
Ümit GÖRÜR
Trabzon 2019

İÇİNDEKİLER

ÖZET

Yazılım tanımlı ağlar teknolojisi, geleneksel ağ yapısını değiştiren ve bu alanda bir çığır açan yeni bir teknolojidir. Bu yeni teknolojinin geliştirilmesi amacıyla emülasyon araçları vardır. Bu proje, yazılım tanımlı ağ teknolojisini kullanan bir emülasyon aracının daha kullanışlı bir hale gelmesi amacıyla tasarlanmıştır. Bu kapsamda 2018-2019 Güz döneminde, tasarım projesi olarak bu emülasyon aracı incelenmiş olup yazılım tanımlı ağ teknolojisi araştırılıp kavranmıştır ve emülasyon aracına yazılım tanımlı ağın analiz edilmesi için bazı özellikler eklenmesi kararlaştırılmıştır.

Bu projede, Yazılım Tanımlı Ağ(YTA) teknolojisi kullanan Linux tabanlı bir emülasyon aracının daha kullanışlı bir hale getirilmesi için emülasyon aracına yeni özellikler eklemek amaçlanmıştır. Bu kapsamda 2018-2019 Güz döneminde tasarımı yapılmış olan "Yazılım Tanımlı Ağlar için Emülasyon Aracının Geliştirilmesi " adlı projede eklenmesi belirlenen özellikler emülasyon aracına eklenmiştir. Bu eklenen özellikler sırasıyla aşağıda gösterilmektedir: Ping Atma Arayüzü, Adım Adım Koşma Arayüzü, Switchler Üzerindeki Tablo ve İstatistikler, Link Üzerinden Geçen Mesaj Bilgisi, Log Screen Ekranı özellikleri emüslasyon aracına eklenmiş olup emülasyon aracı yazılım tanımlı ağların analizini daha anlaşılır ve daha kolay bir hale getirmiştir.

1. GENEL BİLGİLER

1.1. Giriş

Geleneksel ağ mimarisi; router, switch ve birçok ara internet aygıtları ile bu cihazlara tanımlanmış olan protokollerden oluşmaktadır. Her yeni eklenen protokolün avantajları olduğu gibi dezavantajları da vardır. Yeni eklenen protokol, sorunları çözerken diğer yandan karmaşık ve yönetilmesi daha zor ağ yapıları oluşmasına neden olur. Geleneksel ağ mimarisinde performans, güvenlik, yönlendirme gibi işlemler için mevcut alt yapılara ara ağ cihazları eklenir. Bu demek oluyor ki geleneksel ağ mimarisinde her problem için yeni bir protokol, o protokolü gerçekleştirecek ara ağ cihazı ve bundan kaynaklı olarak karmaşıklaşan ağ kontrolü olur. Anlaşılacağı üzere yapılandırmanın manuel olarak yapılması gerekir. Yapılmak istenen yeniliklerin donanım bazında, donanım uygulaması olarak yapılması gerekmektedir. Yazılım Tanımlı Ağ (YTA) teknolojisi, ağ ve yönetim sistemlerinin basitleştirilmesini sağlayan yeni bir ağ yaklaşımıdır. Geleneksel ağ mimarisinin altyapısında tümleşik olan veri ve kontrol katmanlarının ayrılmasını öneren yaklaşımdır. Bu şekilde merkezi bir sistemden yönetilebilen bir kontrol katmanıyla; dinamik, yüksek performanslı, verimli ve geliştirilebilir bir iletişim altyapısı kurabilmesi sağlanmaktadır.

1.2.Yazılım Tanımlı Ağ(YTA) nedir?

Yazılım tanımlı ağ (YTA), donanım ve yazılımı ayıran bir ağ kavramıdır. Yani ağın kontrolü, gerçek veri iletişimi gerçekleştiren donanımdan ayrıdır. Bir YTA'nın iki merkezi bileşeni: kontrol düzlemi ve veri düzlemidir. Yazılım tanımlı ağlar veya kısaca YTA, bir ağdaki yazılımı ve donanımı birbirinden ayırır. Ağ, Ağ Denetleyicisi ile sözde Kontrol Düzlemi (KD) tarafından kontrol edilir. Veri paketlerini iletmek, YTA'da Veri Düzlemi (VD) olarak belirtilen ağ donanımının sorumluluğudur. Veri Düzlemi, Kontrol Düzleminin talimatlarını uygular. Bunlar, örneğin veri paketlerini yönlendirme kuralları olabilir.
Ağ Denetleyicisi genelde merkezileştirilir ve yönlendiriciler veya anahtarlar gibi çeşitli ağ bileşenlerini denetleyebilir ve yönetebilir. Ağ işletim sistemi, kontrol ve yönetim için gereken tüm bilgileri işler ve saklar. Kontrol Düzleminin Ağ Kontrol Cihazı ile yaptığı merkezi görevler şunlardır:

- Farklı ağ bileşenlerinin yönetimi
- Donanımın konfigürasyonu
- Ağ güvenliği konfigürasyonu ilgili güvenlik özellikleri
- Ağ bileşenlerine erişim yönetimi
- Donanım tarafından veri iletme kontrolü
- Veri paketlerini istenilen yere yönlendirmek için yönlendirme tekniklerini oluşturma

Veri Düzleminin, İletme Donanımı ile birlikte merkezi görevleri şunlardır:

- Veri paketlerinin iletilmesi
- Ağ Denetleyicisinden gelen kontrol bilgisinin alınması

Yazılım tanımlı ağın temel bir avantajı, tüm ağın yönetilebilirliğinin çok esnek olmasıdır.Tescilli üretici protokolleri üzerinde izole edilmiş çözümler veya bağımlılıklar yoktur ve yeni bileşenler büyük çaba harcamadan ağa eklenebilir. Yeni donanım bileşenlerinin görevlerini gerçekleştirmesi için gereken tüm bilgiler, otomatik olarak Kontrol Düzleminden alınır. Bireysel cihazlar için ayrı konfigürasyonlar ortadan kaldırılmış ve değişiklikler şebekeden otomatik olarak dağıtılmıştır. Komple ağ topolojisini bilerek, kontrolör ağ üzerinden çok verimli güzergahlar sağlayabilir ve iletim bağlantılarını en iyi şekilde kullanabilir.

Ağlar gittikçe artan taleplere maruz kalmaktadır. Onlar giderek daha büyük ve karmaşıklaşıyor ve aynı zamanda dinamik değişiklikler ve sanallaştırma dereceleri artıyor. Kendi istihbaratlarıyla donatılmış ve büyük oranda özerk çalışan geleneksel ağ bileşenleri artık bu gerekliliklere uygun değildir. Geleneksel ağlarda yapılan değişiklikler, birçok bireysel cihazdaki konfigürasyonların değiştirilmesi ve ayarlanması gerektiğinden, çoğunlukla çok fazla manuel çaba gerektirir. Çoğu zaman, aygıtların farklı işletim sistemleri veya yazılım sürümleri de vardır; bu da özelleştirme ve karmaşıklık maliyetlerini daha da artırır. Aynı zamanda, ağdaki hatalara duyarlılık artmaktadır. Yazılım tanımlı ağ, dağıtılmış istihbarat kavramından ve farklı işletim sistemlerinin kullanımından uzaklaşıyor. YTA'da, ağın istihbarat merkezi bir örneğe taşınır ve tek cihazların veya işletim sistemlerinin konfigürasyonu gereksizdir. Konseptin amacı, istihbarat ve esnekliği arttırırken, şebeke bakım ve yönetimini azaltmaktır. Buna ek olarak, donanım gerçek görevine, veri iletişine konsantre olabilir ve kontrol ve yönetim işlevlerinden kurtulur.Yazılım tanımlı ağ, ağ yapılarının sanallaştırılmasında büyük avantajlar sunar. Belirli özelliklerin artık özel donanımla çalışması gerekmez; ancak Ağ Denetleyicisi aracılığıyla diğer aygıtlara dinamik olarak atanabilir. Ağdaki değişikliklerden, gerçek zamanlı olarak neredeyse gerçeklenebilir. Güvenlik politikalarının yönetimi için YTA kavramı da avantajlıdır. Güvenlik politikaları, tüm ağda merkezi olarak tanımlanabilir ve uygulanabilir. İzole edilmiş çözümler veya farklı güvenlik yapılandırmalarının çoğaltılmasından kaçınılabilir. Daha fazla verimlilik, yazılım tanımlı ağ ve ağ sanallaştırması kombinasyonundan gelir. Daha fazla soyutlama katmanı, YTA'nın avantajlarını daha da geliştirir. Yukarıda açıklanan YTA'nın avantajları, büyük sağlayıcılar ve bulut operatörleri için uygulamayı özellikle ilginç kılmaktadır. Ancak Kontrol ve Veri Düzlemini birbirinden ayırarak daha küçük ağlar kazanabilir.

Yazılım tanımlı bir ağda, kontrol ve veri düzlemi arasında net bir ayrım vardır. Veri Penceresi, veri paketlerinin gerçekte iletilmesinden sorumlu olan donanım bileşenlerini içerir. Cihazlar Katman 3 veya Katman 2 üzerinde çalışır ve veri düzeyinde yönlendiriciler veya anahtar görevlerini yerine getirirler. Talimatlarınızı ve bilginizi Kontrol Düzleminden alacaksınız. Yazılım tanımlı ağ, ayrıca kontrol seviyesinin çalışması için donanım gerektirir. Kural olarak, bunun için, Kontrol Düzleminin donanımından açıkça ayrılmış özel sunucular kurulur. Data Plane donanımı paket yönlendirme için optimize edilmişken, kontrol seviyesi sunucuları, ağın zekası için işleme gücü ve depolama alanı sağlar. Uygulamaya bağlı olarak, Kontrol Düzlemi ve Veri Düzlemi cihazları, açık, standartlaştırılmış veya üreticiye özgü bir protokol aracılığıyla iletişim kurar. Genellikle, örneğin OpenFlow, yazılım tanımlı bir ağda bir protokol olarak bulunabilir. Open Networking Foundation tarafından yönetilir ve geliştirilir. Denetim düzlemine idari erişim, Rol Tabanlı Erişim Denetimi Protokolü (RTED) gibi diğer protokollerle işlenir. Yüksek bir güvenlik standardı sunar ve rollere dayalı olarak çalışır.[1]

1.3. OpenFlow Protokolü nedir?

OpenFlow, yenilikçi ağ protokollerini ve yeni yöntemlerin kullanımdaki ağlarda geliştirilmesini sağlayan bir açık standarttır. Yazılım tabanlı ağ fikri, uzun süredir araştırmacıların kafasını kurcalayan bir problem olup buradaki temel problemler, günümüzdeki ağ bileşenlerinin, Üç bileşenli yapısının birbirinden ayrılması problemi olarak belirtilebilir. Bu üç bileşen aşağıdaki gibidir:

- Yönetim Birimi (Management Plane)
- İletim Birimi (Forwarding Plane)

- Kontrol Birimi (Control)

Bu katmanların birbirinden ayrılması için ilk kullanılan yöntemlerden biri NetFPGA isimli düşük bütçeli bir PCI karttır. Bu ürün programlanabilir bir FPGA bileşeni içeren bir üründür; ancak yine de bu cihazların gelişime açık yönleri oldukça sınırlıdır.

OpenFlow standardı tüm bu gelişim içinde, tüm eksikleri karşılayacak bir çözüm olarak doğmuştur. Yazılım tabanlı bir ağ olarak OpenFlow aynı zamanda sistem yöneticilerine ağdaki topoloji ve paket filtreleme değişikliklerinde yönetim imkânı sunan bir yetenek kazandırmaktadır.

OpenFlow içerisindeki en önemli bileşenler: denetleyici (controller) ve OpenFlow ağ anahtarlarıdır. Klasik olarak ağ anahtarı ve yönlendiricilerde paket iletimi (Data Path) ve yüksek seviye yönlendirme karar mekanizması (Control Path) aynı cihazda bulunur. OpenFlow ise, Data Path bileşenini yine fiziksel ortamda bırakarak Control Path bileşenini bir sunucuya taşımıştır.[2]

Ağ anahtarı yapısı iki bölüme ayrılabilir, Veri katı ve Kontrol katı. Veri katı, ağ anahtarına gelen ve yönlendirilecek flow'ların tutulduğu kısımdır. Bu flow'ların işlenmesi kontrol katının görevidir. Günümüzde bir şasi içinde bulunan bu yapı, üreticileri için dışarıdan müdahaleye izin vermeyen bir şekildedir.

Bu iki katman ile ilgili olarak daha esnek bir yapıya geçmek için OpenFlow switchler tasarlanmıştır. Böylece kullanılagelen ağ anahtarlarında büyük değişiklikler yapılmış ve veri katmanı ile kontrol katmanı birbirinden ayrılmıştır. Bu da veri katmanının güvenli bir bağlantı üzerinden dışarıdan kontrol edilebilmesini getirmiştir.

OpenFlow anahtarının ana fikri, Kontrol katmanını ağ anahtarının dışına almaktır. İki tip OpenFlow ağ anahtarı mevcuttur. Birincisi donanım bazlı olup ticari ağ anahtarlarıdır ve TCAM ile Flow tablosu ve OpenFlow protokolünü kullanırlar. Diğer tip ağ anahtarları ise yazılım tabanlı olup OpenFlow ağ anahtarı fonksiyonlarını bir adet UNIX/Linux işletim sistemi kullanarak yerine getirirler. OpenFlow ağ anahtarı en az üç adet bileşenden oluşmaktadır. Bunlar sırasıyla şunlardır:

Flow tablosu (FlowTable); anahtara bir flow'u nasıl işleyeceğini belirten flow tablosu bileşeni içeren veri bütünü.

Güvenli bağlantı; anahtarın kullandığı komutlara ait paketlerin controller arasında haberleşmesinde kullanılır.

OpenFlow protokolü; anahtarla ve controller'la, açık ve standart bir iletişim olanağı sunan yöntem. OpenFlow ağ anahtarındaki flow table girdileri üzerinde kontrol imkânı sağlamaktadır.

Akış

Akış tanım olarak; aynı kaynak IP'si, hedef IP'si, kaynak portu, hedef portu, protokol demet öğelerini paylaşan ve ayarlanabilen bir D zaman aşımı süresiyle ayrılmış tek yönlü paketler serisidir.

Akış tablosu

Akış tabloları; ağ anahtarları, yönlendiriciler, firewalls gibi günümüzdeki ağ bileşenlerinin çoğunda akış bileşenlerinin saklandığı bir yapıdır. Ağ anahtarlarında ve yönlendirici cihazların içinde veri–satıhı içinde tutulmaktadır[2]

OpenFlow ağ anahtarlarında bulunan akış tabloları 3 adet temel bileşenden oluşmaktadır.

- Akış'ı tanımlayan paketin başlığı: Paketin içerisinde kontrol edilecek bilgilerin bulunduğu kısımdır. Eğer belirtilmemiş bir parametre var ise her değer kabul edilir. Gelen trafik akış tablosu içindeki satırlarla yukardan aşağıya kontrol edilir.
- Paketin nasıl işleneceğini belirten bölüm (action)
- Akış paketlerinin, boyutu, izleme numarası ve zaman aşımı bilgilerinin bulunduğu istatistik veri bölümü.

Akış tablosu içeriğinde akış'ları tanımlamak için iki adet tablo bulunmaktadır. İlk tablo lineer bir tablo olup akış'ları tanımlayan karakterler içermektedir.
Bu tablodaki kayıt girdileri kayıt içindeki bir akış için sadece bazı karakterleri tanımlar (MAC adresi, IP adresi ve Port gibi). Bu ilk tablo da 100 flow kaydı içerir. OpenFlow ağ anahtarı bu tablo içindeki başlık alanlarıyla, gelen akış'ın başlık alanlarını karşılaştırır.

İkinci tablo ise, tam bir karşılaştırma tablosudur ve HASH algoritması kullanarak bu kaydı girdilerini saklar ve içlerinde karşılaştırma ve arama yapabilir. Boyutu ise 131072 adet tam karşılaştırmalı flow girdisi kadardır. Tam karşılaştırmalı girdi ise bir flow'un tüm olası içeriğini tanımlar. Bunlar aşağıda sıralanmıştır:

- Giriş portu
- Kaynak / hedef MAC adresleri
- Ethernet protokolü
- Kaynak/hedef IP adresi
- Ağ protokolü
- Kaynak/hedef portu

Akış tablosu içerisinde iki adet zaman parametresi mevcuttur:

- Idle Timeout : O satıra eşleşme olmadığı durumda satırın akış tablosundan silineceği süredir. Akış tablosundan silinen trafik için en baştan tüm adımlar aynı şekilde gerçekleşir.
- Hard Timeout: Satırın eşleşme olsun olmasın akış tablosundan silineceği süredir.

1.3.1 OpenFlow Mesaj Türleri

OFPT_HELLO	= 0	SİMETRİK MESAJ
OFPT_ERROR	= 1	SİMETRİK MESAJ
OFPT_ECHO_REQUEST	= 2	SİMETRİK MESAJ
OFPT_ECHO_REPLY	= 3	SİMETRİK MESAJ
OFPT_EXPERIMENTER	= 4	SİMETRİK MESAJ
OFPT_FEATURES_REQUEST	= 5	CONTROLLER-SWİTCH MESAJ
OFPT_FEATURES_REPLY	= 6	CONTROLLER-SWİTCH MESAJ
OFPT_GET_CONFIG_REQUEST	= 7	CONTROLLER-SWİTCH MESAJ
OFPT_GET_CONFIG_REPLY	= 8	CONTROLLER-SWİTCH MESAJ
OFPT_SET_CONFIG	= 9	CONTROLLER-SWİTCH MESAJ
OFPT_PAKET_IN	= 10	ASENKRON MESAJ
OFPT_FLOW_REMOVED	= 11	ASENKRON MESAJ
OFPT_PORT_STATUS	= 12	ASENKRON MESAJ
OFPT_PACKET_OUT	= 13	CONTROLLER-SWİTCH MESAJ
OFPT_FLOW_MOD	= 14	CONTROLLER-SWİTCH MESAJ
OFPT_GROUP_MOD	= 15	CONTROLLER-SWİTCH MESAJ
OFPT_PORT_MOD	= 16	CONTROLLER-SWİTCH MESAJ
OFPT_TABLE_MOD	= 17	CONTROLLER-SWİTCH MESAJ
OFPT_MULTIPART_REQUEST	= 18	CONTROLLER-SWİTCH MESAJ
OFPT_FEATURES_REPLY	= 19	CONTROLLER-SWİTCH MESAJ
OFPT_BARRIER_REQUEST	= 20	CONTROLLER-SWİTCH MESAJ
OFPT_FEATURES_REPLY	= 21	CONTROLLER-SWİTCH MESAJ
OFPT_QUEUE_GET_CONFIG_REQUEST	= 22	CONTROLLER-SWİTCH MESAJ
OFPT_QUEUE_GET_CONFIG_REPLY	= 23	CONTROLLER-SWİTCH MESAJ
OFPT_ROLE_REQUEST	= 24	CONTROLLER-SWİTCH MESAJ
OFPT_ROLE_REPLY	= 25	CONTROLLER-SWİTCH MESAJ
OFPT_GET_ASYNC_REQUEST	= 26	CONTROLLER-SWİTCH MESAJ
OFPT_GET_ASYNC_REPLY	= 27	CONTROLLER-SWİTCH MESAJ
OFPT_SET_ASYNC	= 28	CONTROLLER-SWİTCH MESAJ
OFPT_METER_MOD	= 29	CONTROLLER-SWİTCH MESAJ

Şekil 1-3: OpenFlow Mesaj Türleri

OpenFlow mesajları 3'e ayrılır.Bunlar: Contollerdan cihaza giden mesajlar,cihazdan controller'a giden mesajlar,simetrik mesajlar.[3]

- Controllerdan Cihaza Giden Mesajlar

Features: Kontrolörün cihazın yeterlilikleri hakkında bilgi almasını sağlayan mesajlardır. Kontrolör kendisine bağlanmak isteyen her cihazdan bu bilgiyi ister.
Configuration:Kontrolörün cihaza yapılandırma bildirimi yaptığı mesajlardır
Modify-State: Kontrolörün cihazın akış tablosuna ve port durumlarına müdahale etmesini sağlayan mesajlardır.
Read-State: Kontrolörün cihazdan istatistiksel veri çekmesini veya port bilgilerini hakkında bilgi almasını sağlayan mesajlardır.
Packet-Out: Cihazdan gelen bilinmeyen trafiğe ait alınacak aksiyonların söylendiği mesajlardır.

- Cihazdan controller'a Giden Mesajlar

Paket-In : Cihazın akış tablosunda olmayan trafiğe ait aksiyonu öğrenmek için trafik bilgilerini kontrolöre ilettiği mesajlardır.

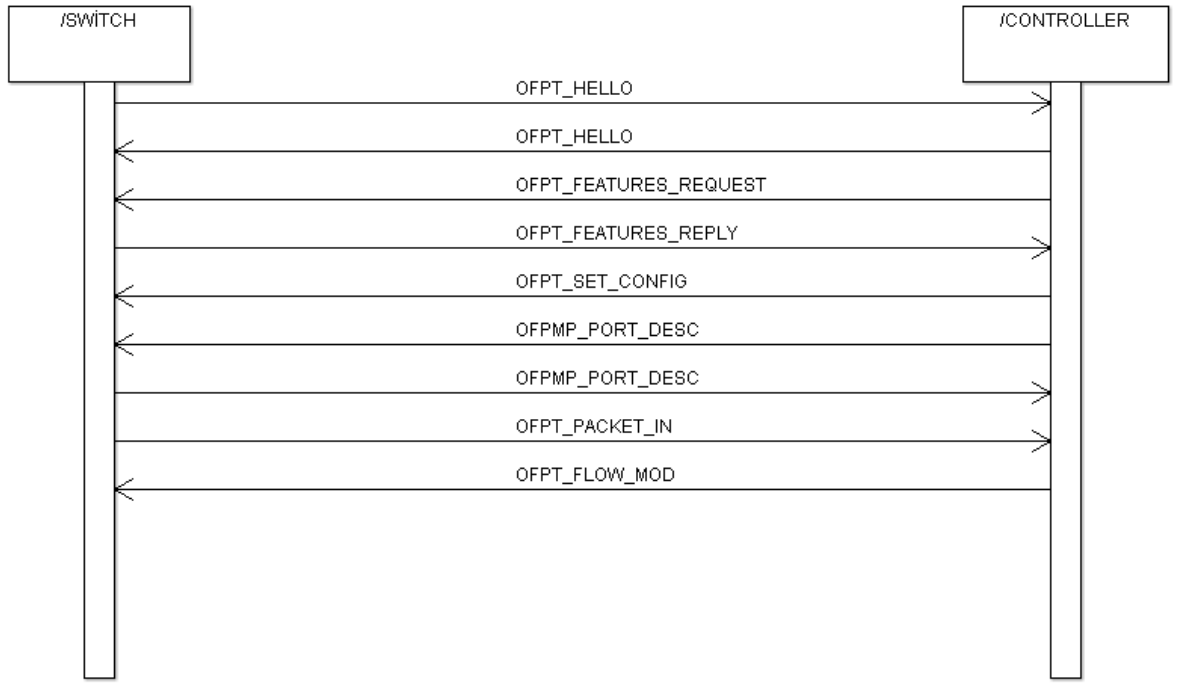
Flow Removed/Expiration: Akış tablosu içerisinde artık kullanılmayan veya süresi geçmiş akışların durumlarını kontrolöre bildiren mesajlardır.

Port-Status: Cihazın port durumlarında olan değişiklikleri kontrolöre ilettiği mesajlardır.
Error: Cihaz üzerinde meydana gelen problemleri kontrolöre iletmek için kullandığı mesajlardır.

- Simetrik Mesajlar

Hello: Cihazın kontrolörle ilk iletişime geçtiği mesajdır.
Echo: Kontrolör ile cihaz arasında sürekli gönderilip alınan kontrol mesajlarıdır. Echo Request ve Echo Reply mesajları bu mesaja örnektir.

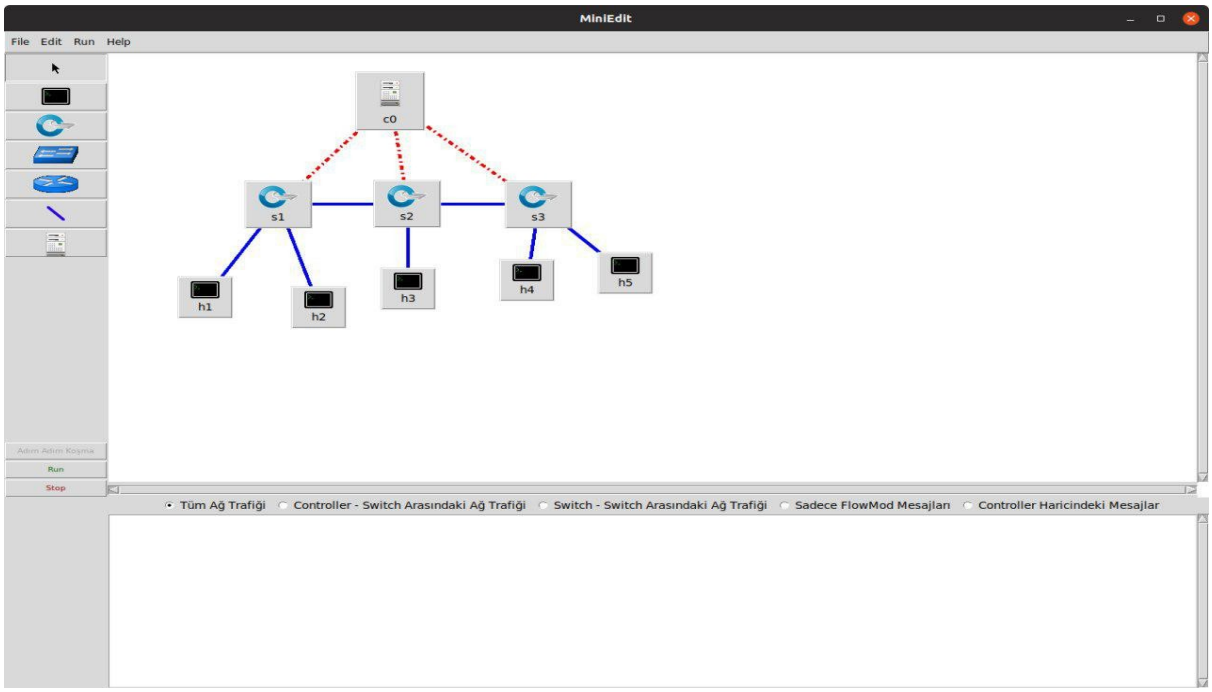
- Flow_Mod: Bu mesajlara ek olarak projede sıkça karşılaşılan controller-switch arası mesajlaşma türü olan *"Flow_Mod"* mesajları, controllerdan switch'e gönderilen bir mesajdır. Cihazdan ping atmak istendiğinde switch hangi link üzerinden mesaj gideceğini bilemez ve controllar'a sorar. Controller hangi link üzerinden mesajın gideceğini switch'e iletir. Controller'ın switch'e gönderdiği bu mesaja *Flow_Mod* mesajı denir.



Şekil 1-11: Controller-Switch Mesajları Sequence Diyagramı

1.4 Scapy Nedir?

Scapy, güçlü bir etkileşimli paket işleme programıdır. Çok sayıda protokolün paketlerini oluşturabilir veya kodlarını çözebilir, onları kabloya gönderebilir, onları yakalayabilir, istekleri ve yanıtları eşleştirebilir ve çok daha fazlasını yapabilir.[5]
NOT: Belirlenen özellikleri eklerken scapy kütüphanelerinden yararlanılmıştır. Scapy, bu projede ağdaki paketleri yakalamak ve parse etmek için kullanılmıştır. Scapy içerisinde bulunan “*”sniff()”* fonksiyonu ile paket yakalama işlemi yapılmıştır.

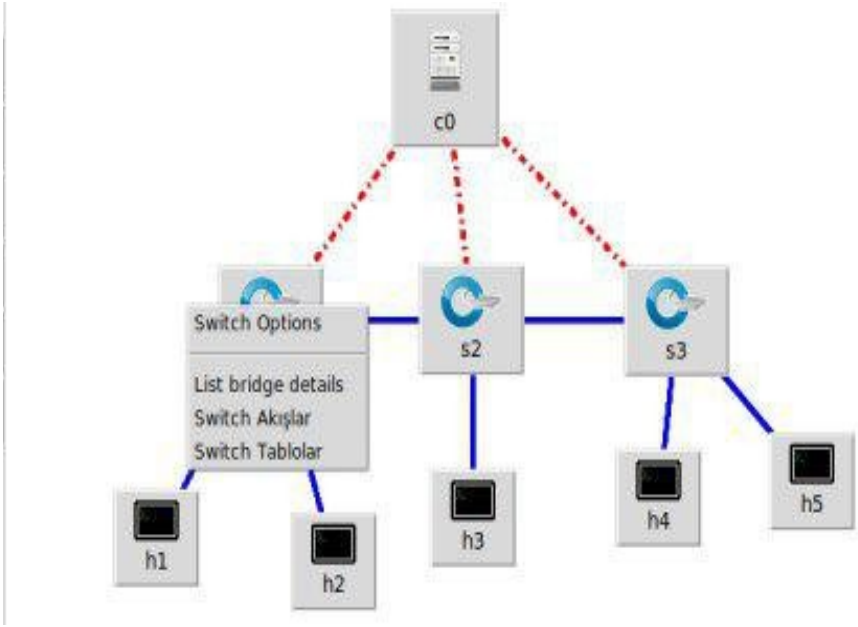


Şekil 1-12: Mininette Oluşturulmuş Bir Topoloji

2. YAPILAN ÇALIŞMALAR

2.1 Switch Üzerinde Tablo ve İstatistiklerin Gösterilmesi

Switch üzerinde tabloların gösterilmesi işlemi uygulama koşmaya başladıktan sonra komut satırı ekranına *“dpctl dump-tables”* komutunun girilmesi ile görülebilmektir. Switch üzerindeki kurallar ise komut satırı ekranına *“dpctl dump-flows”* komutunun gönderilmesiyle görülebilmektedir. Oluşturulan arayüzde ise bu işlemler daha kolay bir şekilde görülebilmektedir. Örneğin uygulamada bir adet controller, iki adet switch, iki adet host olduğu varsayılsın. Topoloji koşulmaya başlatıldığında hangi switch üzerinde hangi akışların olduğunu ve kaçar adet kullanıldığı görülebilmekte ayrıca tablolara da switch üzerine gelip arayüz açıldığında istatistikler rahat bir şekilde görülebilmektedir.



Şekil 1-1 Switch Arayüzü

```
 cookie=0x0
 duration=6.625s
 table=0
 n_packets=0
 n_bytes=0
 idle_timeout=60
 priority=1
arp
in_port="s1-eth2"
vlan_tci=0x0000/0x1fff
dl_src=66:54:aa:ef:9f:61
dl_dst=72:2f:5a:bf:2b:2f
arp_spa=10.0.0.2
arp_tpa=10.0.0.1
arp_op=2 actions=output:"s1-eth1"
```

Şekil 1-2 Switch Akışlar

```
OFPST_TABLE reply (OF1.3) (xid=0x2):
  table 0:
    active=7, lookup=112, matched=112
  table 1:
    active=0, lookup=0, matched=0
  tables 2...253: ditto
```

Şekil 1-3 Switch Tabloları

Opcode	ARP Message Type
1	ARP Request
2	ARP Reply
3	RARP Request
4	RARP Reply
5	DRARP Request
6	DRARP Reply
7	DRARP Error
8	InARP Request
9	InARP Reply

Şekil 1-4 Arp Mesaj türü[8]

Şekil 1-1 de h1 cihazından h2 cihazına ping gönderildiğinde, Şekil 1-1 ‘de gösterilen switch akışlar özelliğini girildiğinde şekil 1-2 deki gibi bir örnek akış tablosu görülmektedir. Bu akış tablosunda;
cookie: Akış filtrelemek için kontroller tarafından kullanılabilir akış istatistik, akış değişikliği.
Duration: Tabloda kaç saniyedir bulunduğunu gösteren değişken.
table: Akışın hangi tabloyu kullanıdığı bilgisi.
n_packets: Akış üzerinden kaç adet paketin geçtiği bilgisi.
n_bytes: Akış üzerinden kaç bytelık veri gönderildiği bilgisi.
idle_timeout: Bu akışın ne kadar süre tabloda bulunacağı bilgisi.
Priority: Akışın önceliğini gösteren değişken.
Arp: Gönderilen mesajın arp mesajı oldunu belirtir.
in_port: Mesajın hangi link üzerinden gönderiliği bilgisi
vlan_tci: Sanal Lan maskesi
dl_src: Mesajı gönderen cihazın mac adresi bilgisi.
dl_dst: Mesajın gideceği cihazın mac adresi bilgisi
arp_spa: Mesajın gönderildiği cihazın ip adresi bilgisi.
arp_spa: Mesajın gideceği cihazın ip adresi bilgisi.
arp_op: Hangi arp mesajının gönderildiğini gösteren değer.
actions=output: Mesaj gönderilen cihazın hangi portta olduğunu döndüren değişken.

Şekil 1-1 de h1 cihazından h2 cihazına ping gönderildiğinde, Şekil 1-2 ‘de gösterilen switch tablolar özelliğini girildiğinde şekil 1-3 deki gibi bir örnek tablo görülmektedir.
Bu tablo, mesajın hangi tabloda olduğunu ve bu tabloda kaç tane aktif tablo olduğu bilgisini içermektedir. Bu tabloda, table 0 da veri olduğu görülür. Diğer tabloların da boş olduğu görülmektedir.

Switch üzerinde tablo ve istatistiklerin oluşturulması arayüzünde akışlar için bir adet ve tablolar için bir adet class kullanılmıştır. Bu classlardan ilki “*FlowsOnSwitch”*, tablolar içinde “*TablesOnSwitch”* tir.

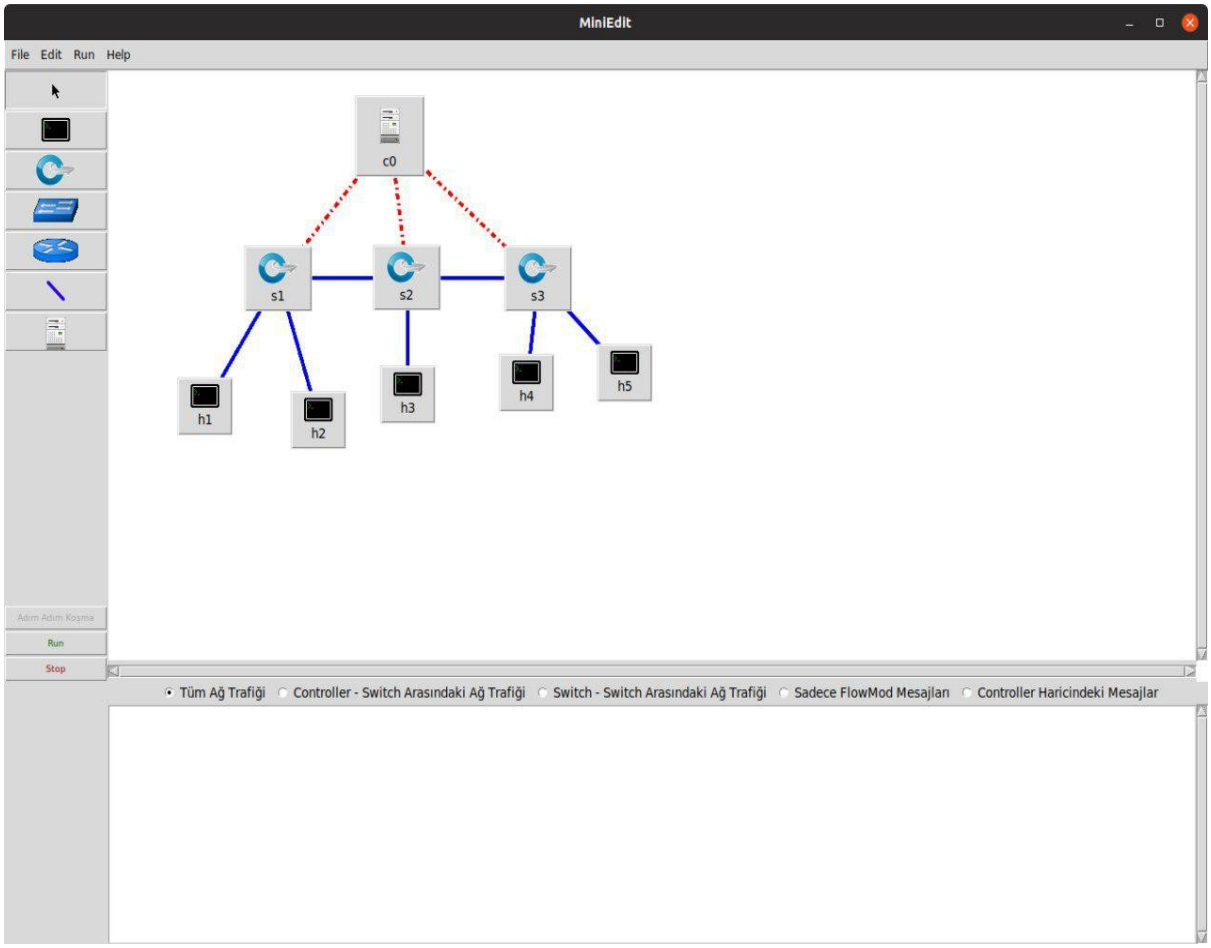
Birinci class olan “*FlowsOnSwitch”* classında üç adet fonksiyon vardır. Bu fonksiyonlardan ilki, switch’e bağlı bir arayüz oluşturmayı sağlayan fonksiyondur. İkinci fonksiyon ise bu arayüzde controllerın “*Administer Openflow Switches”* [7] panelinden yararlanarak switchlere ulaşılmayı sağlayan fonksiyonu içerir. “*ovs-ofctl O OpenFlow13 dump-flows %s”* komutundan yararlanarak akışlara ulaşılmaktadır. %s ise hangi switch’in akış tablosu olduğunu belirtir. %s kısmına “*self.switch.name”* ile ulaşılmaktadır. Üçüncü fonksiyon ise arayüzden çıkmayı sağlayan komutları içerir.

Diğer class ise switchler üzerindeki tablolara erişimi sağlayan “*TablesOnSwitch” classıdır*. Bu classta üç adet fonksiyon vardır. Bu fonksiyonlardan ilki switch’ e bağlı bir arayüz oluşturmayı sağlar. İkinci fonksiyon ise arayüzde controllerın *“Administer Openflow Switches”* panelinden yararlanarak switchlere ulaşılmasını sağlamaktadır. “*ovs-ofctl O OpenFlow13 dump-tables %s”* komutundan yararlanarak tablolara ulaşılmaktadır. %s ise hangi switch’in tablosu olduğunu belirtir. %s kısmına “*self.switch.name”* ile ulaşılmaktadır. Üçüncü fonksiyon arayüzden çıkmayı sağlayan komutaları içerir.

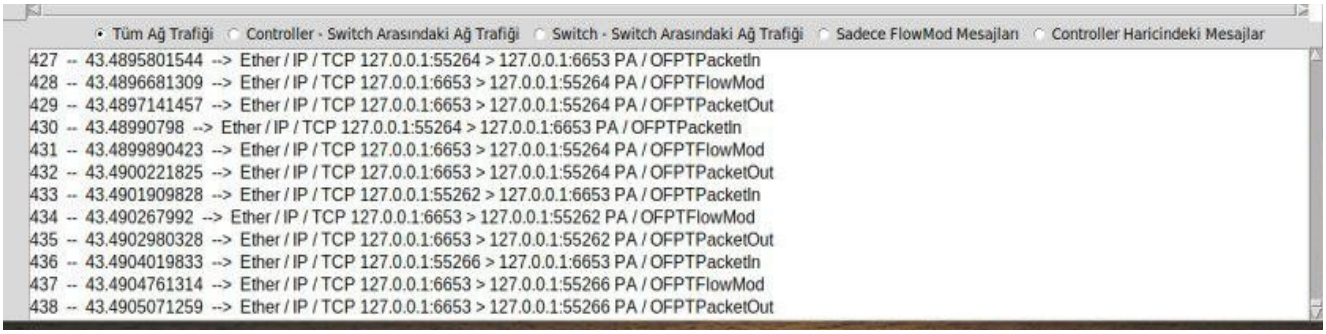
Diğer bir fonksiyonlar da miniedit class’ı içerisine yazılmış olan “*flowsOnSwitchDetail”ve “tablesOnSwitchDetail”* fonksiyonlarıdır. Bu fonksiyonlar ise oluşturulan “*FlowsOnSwitch” ve “TablesOnSwitch”* classlarının Miniedit arayüzünde görülebilmesini sağlayan komutları içerir.

2.2 Log Screen Ekranı

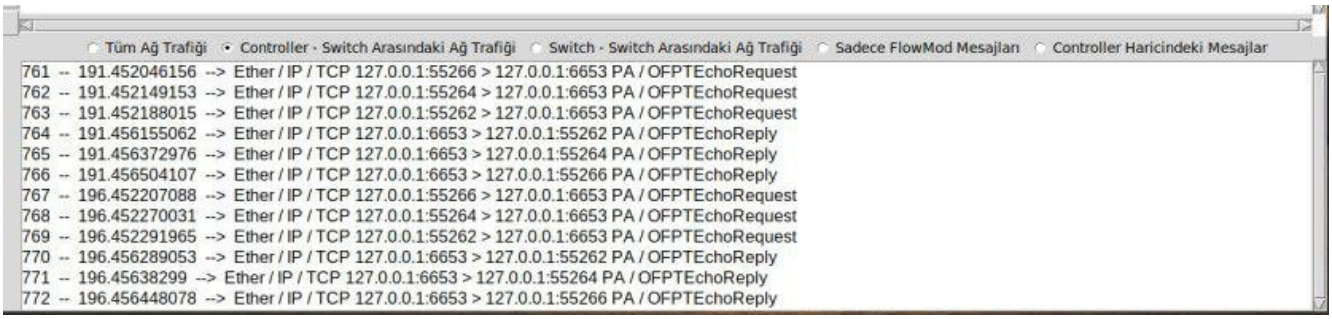
Mininet arayüzünde, oluşturulan topoloji koşulmaya başlandığında ağdaki mesajlaşmaları görebilmek için *"Wireshark"* veya benzeri programlara ihtiyaç duyulmaktadır. Bu da kullanıcı açısından çok verimsiz, ağdaki mesajlaşmaları ve içeriklerini kontrol etmek oldukça zordur. Oluşturulmuş olan *"Log Screen Arayüzü"* 'nde ise ağdaki tüm mesajlaşmaları, controller-switch, switch-switch, sadece Flow_Mod mesajlarını ve içeriklerini ayrı ayrı göstermeye olanak sağlar; ayrıca diğer programlara göre çok daha hızlı görebilmeye olanak sağlamaktadır. Örneğin uygulamada bir adet controller, iki adet switch, iki adet host olduğu varsayılsın (Şekil: 2-0). Topoloji, Run komutuyla başlatıldığında sniff fonksiyonu çalışmaya başlar. Sniff fonksiyonundan gelen paketler parse edilir yani ayrıştırılır. Bu ayrıştırma sonucunda bir thread yardımıyla parse edilen pakatler ekrana yazdırılır. Radyo butonları sayesinde kullanıcı, *"Log Screen Ekranı"*'nda tüm mesajlaşmaları(şekil:2.1), controller-switch(şekil: 2.2), switch-switch(şekil: 2.3), log screen(şekil: 2.4) mesajlaşmalarını ve içeriklerini gerçek zamanlı olarak görebilmektedir. Aynı zamanda ping atılmışsa, bu ping'in hangi güzergahtan gittiği arayüzde görülmektedir.



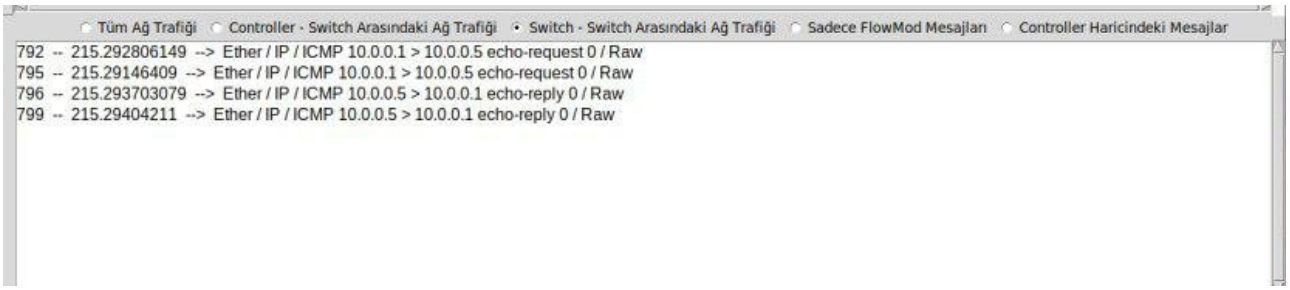
Şeki 2-0 Oluşturulan topoloji



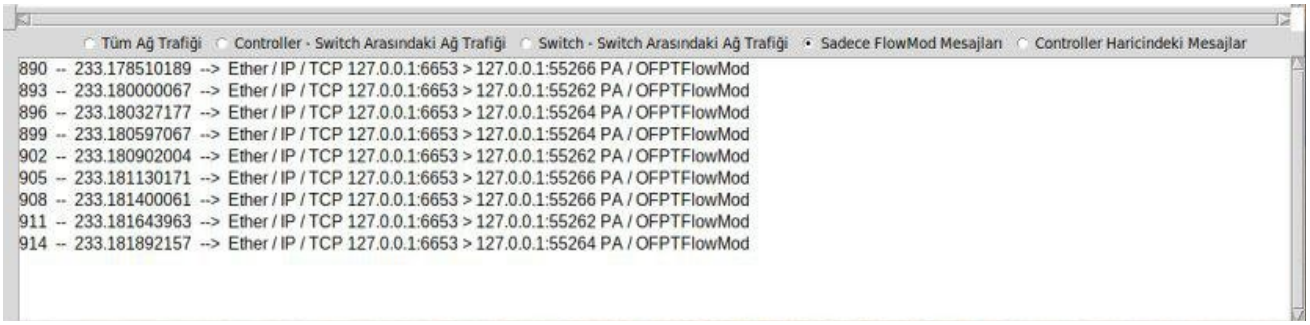
Şekil 2-1 Tüm Ağ trafiği mesajlaşmaları



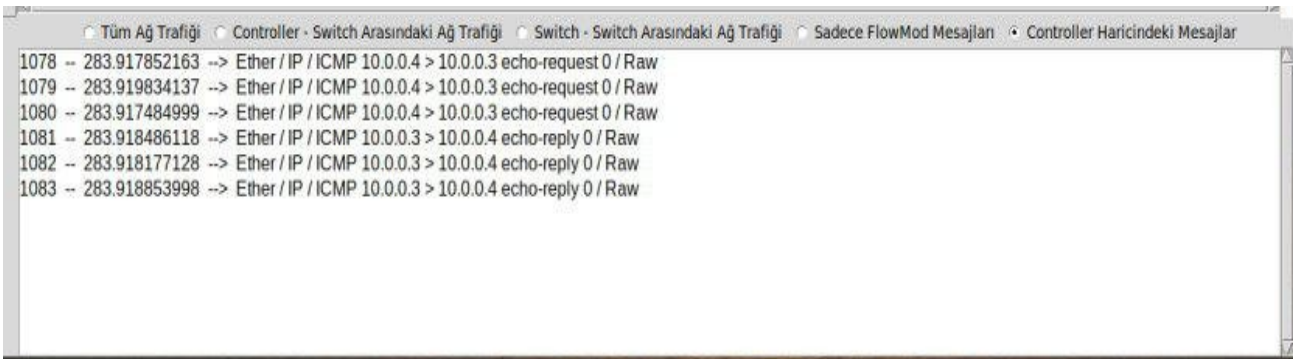
Şekil 2-2 Controller-Switch arası mesajlaşmalar



Şekil 2-3 Switch-Switch Arası mesajlaşmalar



Şekil 2-4 Flow_Mod mesajları



Şekil 2-5 Controller Haricindeki Mesajlaşmalar

MiniEdit

```
###[ Ethernet ]###
  dst       = 00:00:00:00:00:00
  src       = 00:00:00:00:00:00
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0xc0
     len       = 60
     id        = 44344
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = tcp
     chksum    = 0x8ec1
     src       = 127.0.0.1
     dst       = 127.0.0.1
     \options   \
###[ TCP ]###
        sport     = 41602
        dport     = 6653
        seq       = 3209993524
        ack       = 3364193960
        dataofs   = 8
        reserved  = 0
        flags     = PA
        window    = 86
        chksum    = 0xfe30
        urgptr    = 0
        options   = [('NOP', None), ('NOP', None), ('Timesta
mp', (2723941126, 2723936702))]
###[ OFPT_ECHO_REQUEST ]###
           version   = OpenFlow 1.3
           type      = OFPT_ECHO_REQUEST
           len       = 8
           xid       = 0
```

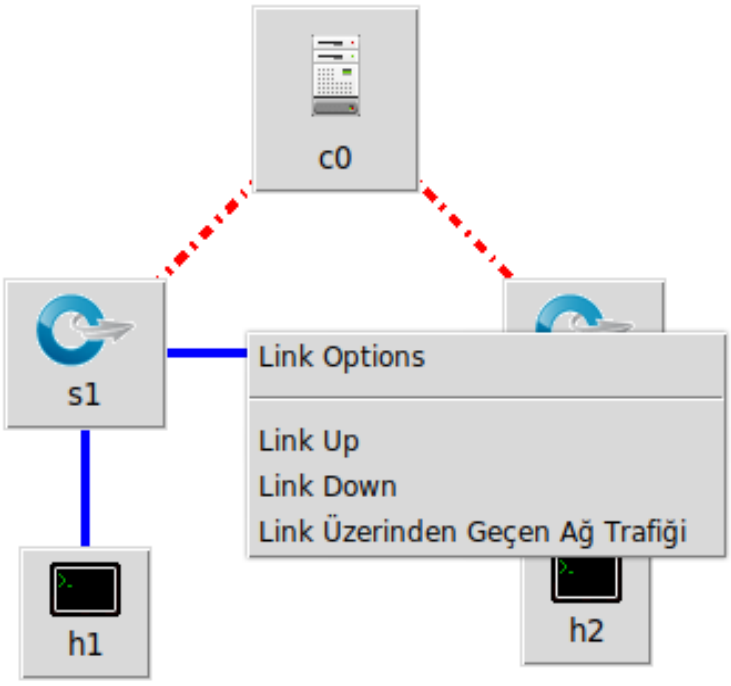
Şekil 2-6 Mesajların İçeriği

“*Log Screen Ekranı”* arayüzünü oluştururken iki class tanımlanmıştır. Bu classlardan ilki “*SniffingOnAllNetwork”, ikincisi ise “MyThreadingClass”* tır. Birinci classta sekiz adet fonksiyon yazılmıştır. İlk fonksiyon miniedit guı’e bağlı bir arayüz oluşturmayı ve threadlerin oluşturulmasını sağlayan fonksiyondur. İkinci fonksiyon, sniffleme işlemi için uygun parametrelerin verildiği ve snifflemenin başlatıldığı fonksiyondur. Üçüncü fonksiyon, linkleri dinleme işleminin yapıldığı fonksiyondur. Dördüncü fonksiyon, tüm ağ trafiğindeki mesajlaşmaların parse edildiği yani ayrıştırıldığı fonksiyondur. Beşinci fonksiyon, gönderilecek olan ping’in hangi güzergah üzerinden gideceğini belirleyen fonksiyondur. Bu fonksiyonda ikinci class olan, thread oluşturmayı sağlayan classtan yaralanarak bir thread oluşturulmuştur. Altıncı fonksiyonda belirlenen ping güzergahı oluşturulan thread yardımıyla ping hangi güzergahtan gidiyorsa o güzergahın renginin değişmesini sağlayan fonksiyonu tanımlamaktadır. Yedinci fonksiyon, oluşturulan threadi durdurmayı sağlayan threadi başlatır. Son olarak sekizinci fonksiyon ise threadin çalışıp çalışmadığını kontrol eden fonksiyondur.

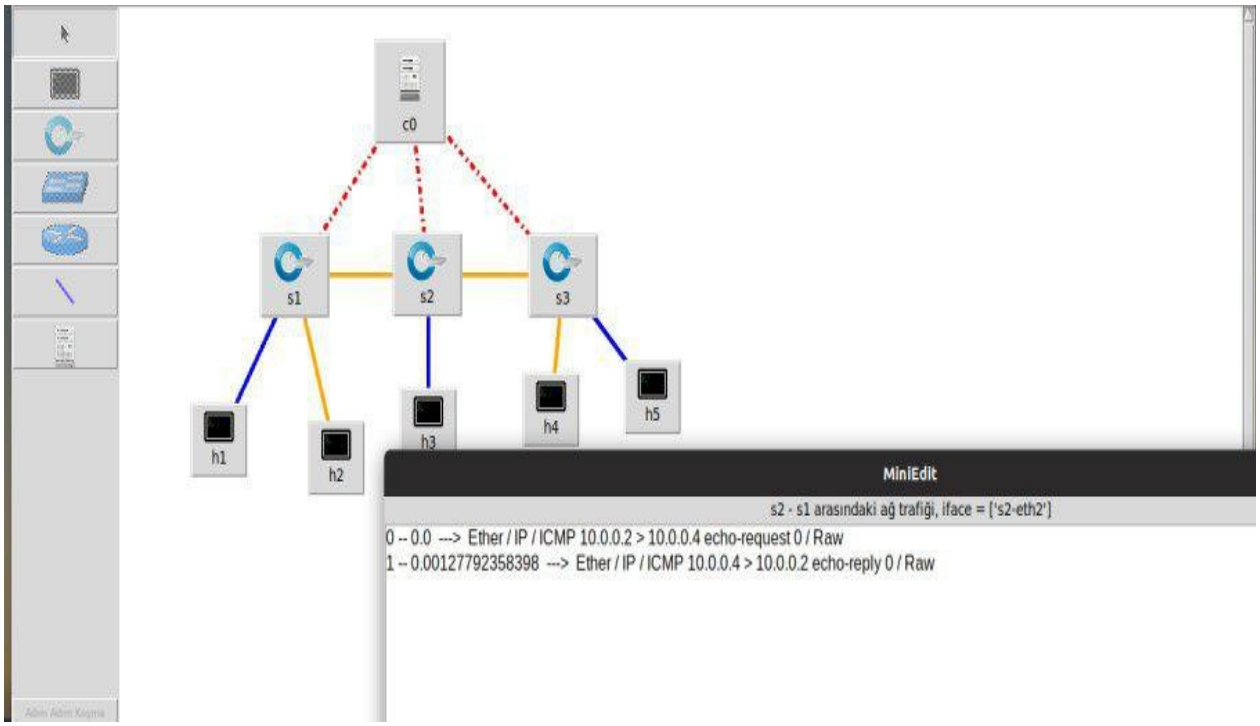
Miniedit classı içine beş adet fonksiyon tanımlanmıştır. Bu fonksiyonlardan ilki istenen mesajlaşmaların görülmesine olanak sağlayan radyo butonlarının tanımlandığı fonksiyondur. İkinci fonksiyon, radyo butonlarından seçilmiş olan mesajlaşma türünün diğer bir mesaj türüne tıklandığından listenin temizlenmesini sağlayan komutları içeren fonksiyondur. Üçüncü fonksiyon, GUI üzerinde bir list box oluşturmayı sağlayan fonksiyondur. Dördüncü fonksiyon, radyo butonlarından seçilen mesaj türüne göre bu mesajların listeye atılmasını ve içeriklerinin görülmesini sağlayan komutların bulunuğu fonksiyonu içerir. Beşinci ve son fonksiyon, paket detaylarının text box’a yazıldığı fonksiyondur.

2.3 Linkler Üzerinden Geçen Mesaj Bilgisi

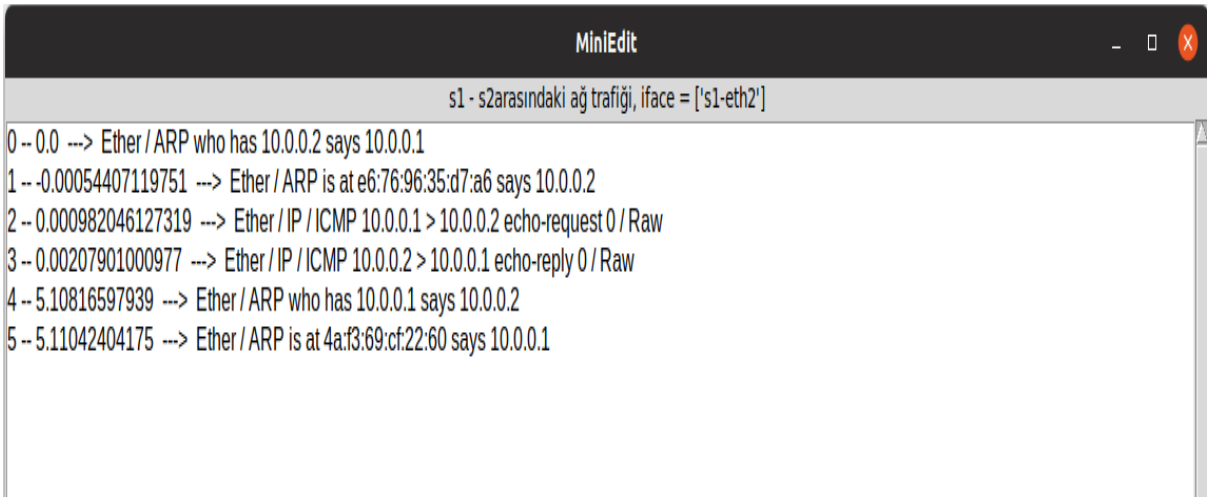
Miniedit arayüzünde, hangi link üzerinden hangi mesajların geçtiği bilgisini gösteren bir arayüz yoktur. Hangi link üzerinden hangi mesajın geçtiğini görmek oldukça zor bir işlem gerektirir. Oluşturulan arayüz sayesinde tıklanan linkin adı, link üzerinden hangi mesajların geçtiği ve bu mesajların içeriği kolaylıkla görülebilmektedir. Örneğin uygulamada bir adet controller, iki adet switch, iki adet host olduğu varsayılsın(Şekil 3-1). Topoloji Run komutuyla başlatıldığında sniff fonksiyonu çalışmaya başlar. Kullanıcı hangi linkin üzerinden geçen mesajı takip etmek isterse o linke sağ tıklayıp açılan pencerede linkin adını, o link üzerinden hangi mesajların geçtiğini(şekil : 3.2) ve bu mesajların içeriklerini(şekil : 3.3) rahatlıkla görebilir. Sniffleme ve arayüze yazma işlemi bir thread yardımıyla gerçek zamanlı olarak yapılır.



Şekil 3-1 Link üzerinden geçen ağ trafiği özelliği



Şekil 3-2 Link üzerinden geçen ağ trafiği mesajları



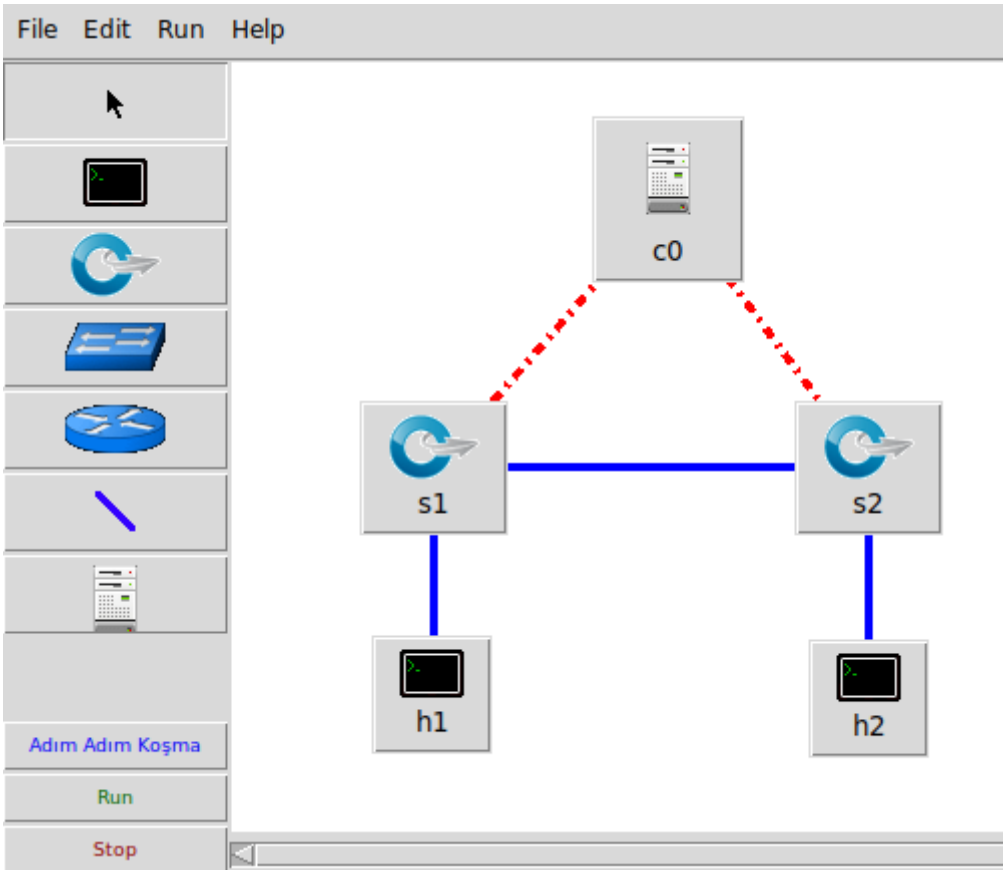
Şekil 3-3 Link üzerinden geçen ağ trafiği mesajlarının içeriği

Linkler üzerinden geçen mesaj bilgisi arayüzünü oluştururken "S*niffingOnLink"* adında bir class oluşturulur ve daha önceden oluşturulmuş olan *"MyThreadingClass"* classından yararlanılarak sistem geliştirilir. "S*niffingOnLink"* classının altı tane fonksiyonu vardır. Ayrıca oluşturulan arayüzün sistemde görülebilmesi için miniedit class'ı içinde "*sniffingOnLinkDetail"* adında bir fonksiyon tanımlanmıştır. İlk fonksiyon, miniedit gui'e bağlı bir arayüz oluşturmayı ve threadlerin oluşturulmasını sağlayan fonksiyondur. İkinci fonksiyon, sniffleme işleminin yapıldığı fonksiyondur. Üçüncü fonksiyon, ikinci fonksiyondan gelen veriyi parse eder. Dördüncü fonksiyon, paket detayının text şeklinde ekrana yazdırılmasını sağlar. Beşinci fonksiyon, thread çalışıp çalışmadığını kontol eder. Altıncı ve son fonksiyon, arayüzün kapatılmasını sağlayan kodları içerir. Bu fonksiyonlar sayesinde linkler üzerinden geçen mesaj bilgisi arayüzü oluşturulmuş olup threadler yardımıyla linklerden geçen mesajlar gerçek zamanlı olarak arayüzde görülmektedir.

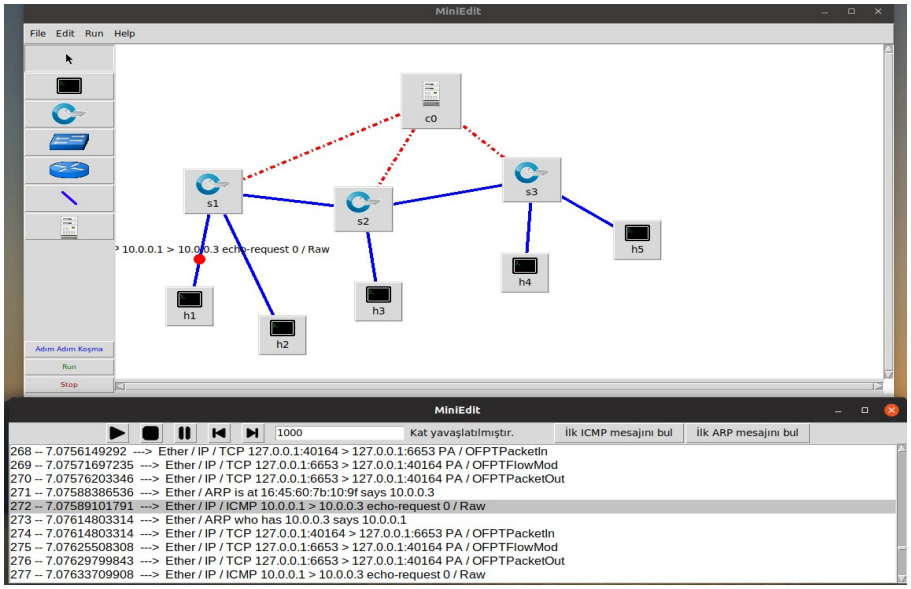
2.4 Adım Adım Koşma Arayüzü

Mininet uygulamasında ağ haberleşmesi oldukça hızlıdır(Yaklaşık 60 GBit/s). Ağ haberleşmesinin bu kadar hızlı olması gönderilen ve gelen paketlerin takibini zorlaştırmaktadır. Bu da kullanıcı için bir sorun teşkil etmektedir. Bu kapsamda Miniedit uygulamasına ağ haberleşmesini, ağdaki paketleri kolaylıkla görebilmek için adım adım koşma arayüzü oluşturulmuştur. Adım adım koşma arayüzü, ağ trafiğini yavaşlatıp ağdaki cihazlar arası haberleşmeleri arayüzde göstermeye yarayan bir özelliktir. Örneğin uygulamada bir adet controller, iki adet switch, iki adet host olduğu varsayılsın(Şekil 4-1).
Oluşturulan topoloji run komutuyla birlikte çalışmaya başlar ve ağdaki mesajlaşmalar kaydedilir. Kullanıcı, stop komutuna bastığında *"Adım Adım Koşma"* adında bir buton açılır. Bu butona tıklandığında ağdaki mesajlaşmaların olduğu bir pencere açılır. Bu pencerede adım adım koşmayı başlatacak olan start butonu, adım adım koşmayı durdurmak için pause butonu, bir sonraki adıma geçmek için nextstep butonu, geri gelip önceki paketi görmek için backstep butonu ve en başa dönmek için stop butonu bulunur. Kullanıcı, start butonu ile mesajları, mesajların içeriğini ve mesajların hangi güzergahtan hangi cihaza gittiğini gözlemler. İstediği zaman ağ trafiğini durdurup gözlem yapabilir. Nextstep ve backstep butonları sayesinde paketleri daha detaylı inceleyebilir. Stop butonu ile de başa dönüp tekrar ağ trafiğini gözlemleyebilir. Ayrıca kullanıcı, scrolla ağ trafiğini hızlandırıp yavaşlatabilir.

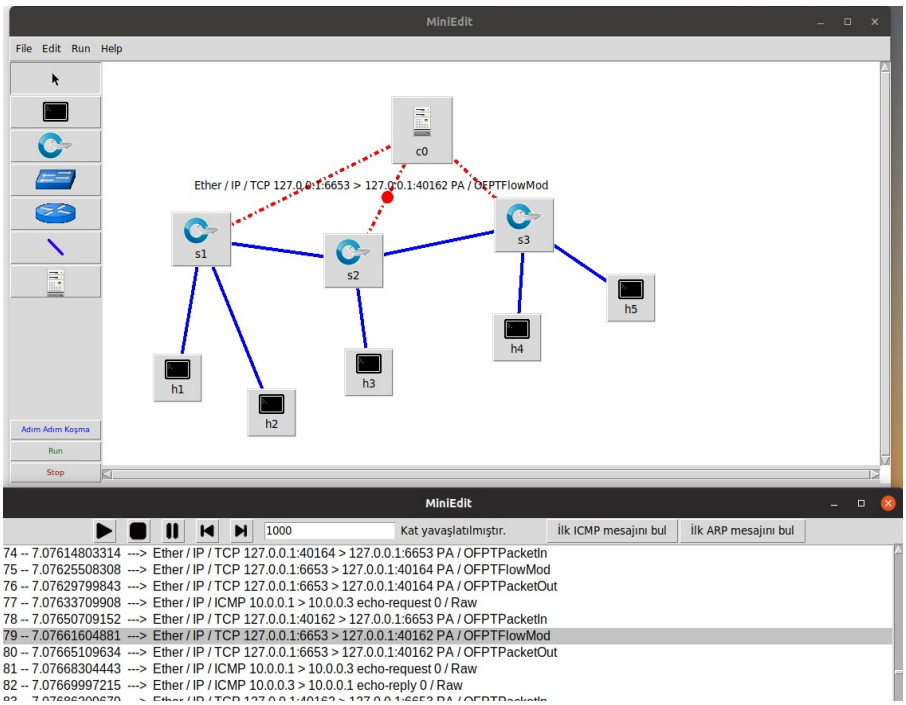
Adım adım koşma arayüzü sayesinde Miniedit arayüzü daha kullanışlı ve daha anlaşılır bir hale gelmiştir.



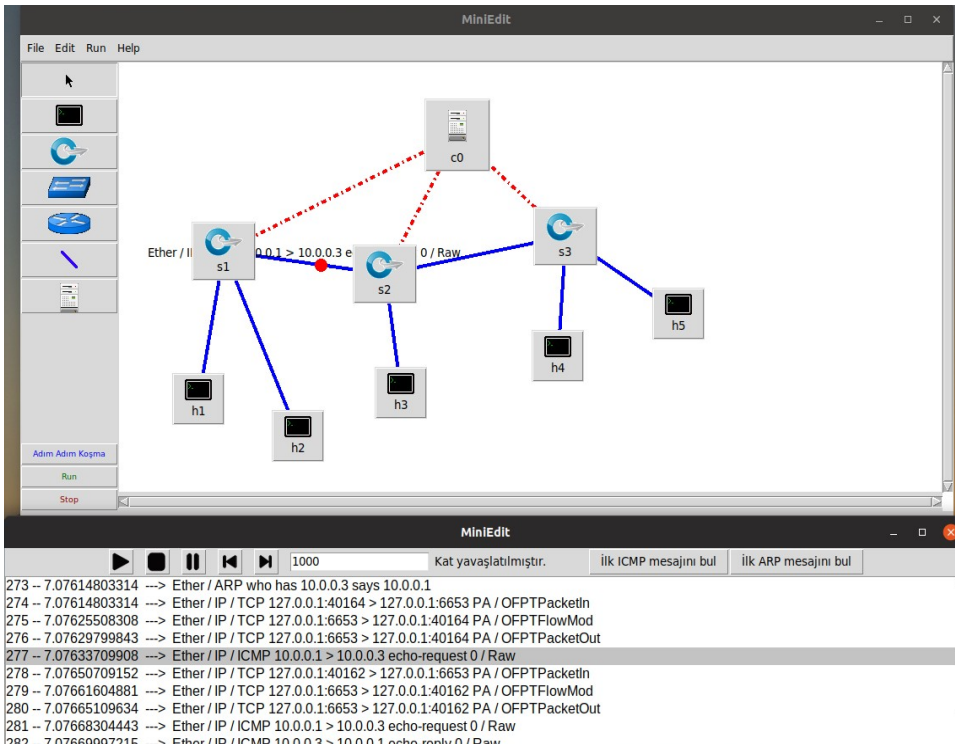
Şekil 4-1 Adım Adım Koşma



Şekil 4-2 Adım Adım Koşma Arayüzü Switch-Host arası giden paket



Şekil 4-3 Adım Adım Koşma Arayüzü Switch-Controller arası giden paket



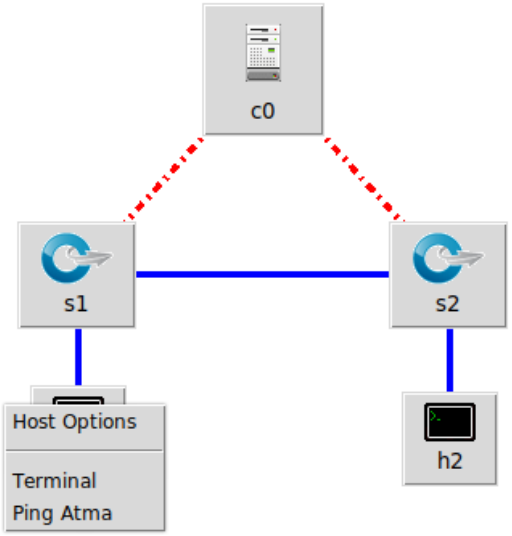
Şekil 4-4 Adım Adım Koşma Arayüzü Switch-Switch arası giden paket

Adım adım koşma arayüzü oluşturulurken *"DoStepByStep"* adında bir class oluşturulur. Paralel koşmayı sağlamak için oluşturulmuş olan *"MyThreadingClass"* classından yararlanılır. *"DoStepByStep"* classında *"mouseWheel"* fonksiyonu, scrolla hızı ayarlamak için oluşturulur. *"onselect"* fonksiyonu adım adım koşma arayüzünde istenilen paketten başlamayı sağlayan fonksiyondur. *"onDoubleButtonSelect"* fonksiyonu çift tıklandığında paketin seçilmesini sağlayan fonksiyondur. *"showToPacketDetail"* paketin içeriğini göstemeye yarayan fonksiyondur. *"addToListNodes"* yakalanan paketleri ekranda bulunan listbox a eklemeyi sağlar. *"partitionNodeCoordinate"* fonksiyonu, miniedit classı içinde oluşturulan topolojideki ağ cihazlarının koordinatlarını bulan fonksiyondan yararlanarak ağ cihazlarını ayrı ayrı listemeyi sağlayan fonksiyondur. *"doStart"*,"doStop", *"doPause"*, *"doNextStep"*, *"doBackStep"*, fonksiyonları butonları tanımlayan fonksiyonlardır. *"findPacketCoorAndDraw"* fonksiyonu, seçilen paketin koordinatlarını bulup çizdirme fonksiyonuna gönderen fonksiyondur. *"drawPacket"* fonksiyonu, *"findPacketCoorAndDraw"* fonksiyonundan gelen verileri ekrana bastırmaya yarayan fonksiyondur.

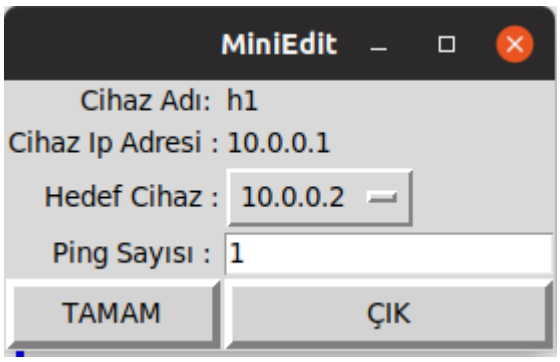
Miniedit classı içine de *"retunTopoCoordinate"* adında bir fonksiyonu oluşturulur. Bu fonksiyon topolojide oluşturulan tüm nodeların koordinatlarını döndüren fonksiyondur.

2.5 Ping Atma Arayüzü

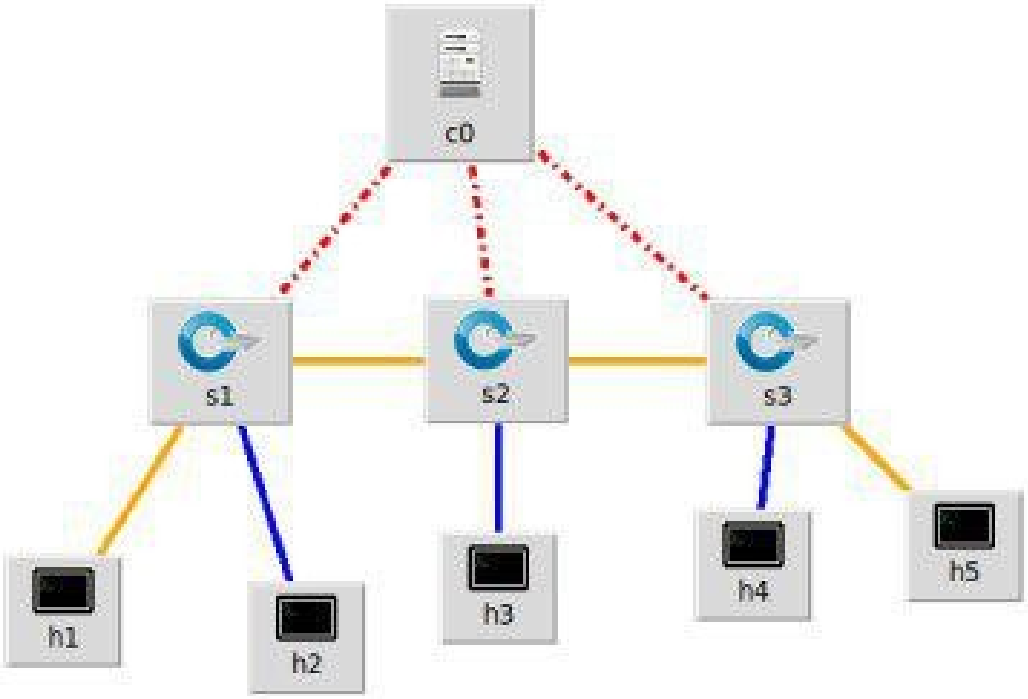
Uygulamada ping atmak için "Mininet için basit komut satırı arayüzü" ve "herhangi bir hostun terminal ekranından atılması" olmak üzere iki yöntem vardır; fakat bu yöntemler kullanım itibariyle zor ve karmaşık bir yapıya sahiptir. Örneğin uygulamada bir adet controller, iki adet switch, iki adet host olduğu varsayılsın. İlk yöntemde komut satırı arayüzünden ping atabilmek için komut satırı ekranına girip "h1 ping h2" komutu yazılır. Diğer yöntemde ise ping atılmak istenen hostun terminal ekranına girilip "ping 10.0.0.2" komutu yazılarak ping atma işlemi gerçekleştirilir. Geliştirilen ping atma arayüzü ise bu işlemlerin daha basit bir şekilde gerçekleştirilmesine olanak sağlar. Ping atmak istenilen cihazın arayüzüne girilir. Bu arayüzde cihazın ismi ve ip adresi görülür. Ping gönderilebilecek cihazların da ip adresleri opsiyonel olarak görülür ve bu ip adreslerine göre cihaz seçilir. Kaç adet ping gönderilmek istenirse o sayı yazılır ve ping atma işlemi gerçekleştirilir.



Şekil 5-1 Ping atma özelliği



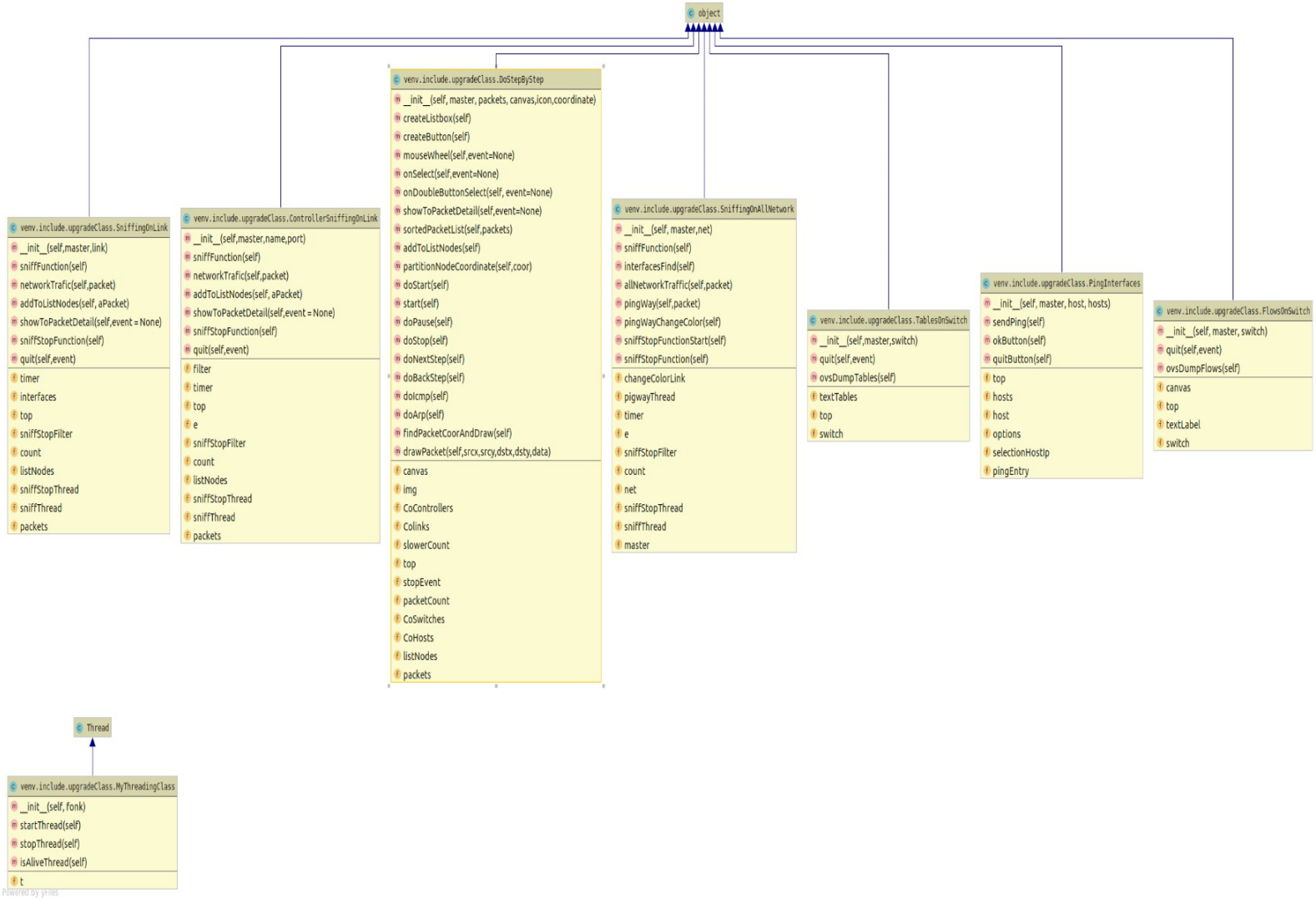
Şekil 5-2 Ping atma arayüzü



Şekil 5-3 Giden Ping Güzergahının Görülmesi

Ping atma arayüzü oluşturulurken iki adet class tanımlanmıştır. Bu classlardan ilki, *"PingInterface"* adlı classtır. İkinci class ise thread işlemi için oluşturulan "*MyTheadingClass"* tır. İlk class'ın içerisinde dört adet fonksiyon vardır. Bu fonksiyonlardan ilki, arayüz oluşturmaya yarayan fonksiyondur. Hostun adı, ip adresi, ping atılabilecek cihazlar ve ping sayısı bu fonsiyondan tanımlanır. Hostun adı "*self.host.name"* ile çağrılır. Ip adresi "*self.host.IP()"* ile belirlenir. Ping atılabilecek cihazlar option menuye atılır. Ping sayısı da bir Entry ile tanımlanır. Diğer fonksiyon ping atma komutunun yazıldığı fonksiyondur. "*ping -c %s %s* " komutunun girilmesi ile gerçekleştirilir. Diğer fonksiyon ise oluşturulan thread ile ping atma komutunun çağrıldığı fonksiyondur. Son fonksiyon ise arayüzden çıkmayı sağlayan fonksiyondur.

Diğer bir fonksiyon da miniedit class'ı içerisine yazılmış olan "*pingInterfaceDetail"* fonksiyonudur. Bu fonksiyon ise oluşturulan "*PingİnterfaceDetail"* classının Miniedit arayüzünde görülebilmesini sağlayan komutları içerir.


object
venv.include.upgradeClass.SniffingOnLink
__init__(self,master,link)
sniffFunction(self)
networkTrafic(self,packet)
addToListNodes(self, aPacket)
showToPacketDetail(self,event = None)
sniffStopFunction(self)
quit(self,event)
timer
interfaces
top
sniffStopFilter
count
listNodes
sniffStopThread
sniffThread
packets
venv.include.upgradeClass.ControllerSniffingOnLink
__init__(self,master,name,port)
sniffFunction(self)
networkTrafic(self,packet)
addToListNodes(self, aPacket)
showToPacketDetail(self,event = None)
sniffStopFunction(self)
quit(self,event)
filter
timer
top
e
sniffStopFilter
count
listNodes
sniffStopThread
sniffThread
packets
venv.include.upgradeClass.DoStepByStep
__init__(self, master, packets, canvas,icon,coordinate)
createListbox(self)
createButton(self)
mouseWheel(self,event=None)
onSelect(self,event=None)
onDoubleButtonSelect(self, event=None)
showToPacketDetail(self,event=None)
sortedPacketList(self,packets)
addToListNodes(self)
partitionNodeCoordinate(self,coor)
doStart(self)
start(self)
doPause(self)
doStop(self)
doNextStep(self)
doBackStep(self)
doIcmp(self)
doArp(self)
findPacketCoorAndDraw(self)
drawPacket(self,srcx,srcy,dstx,dsty,data)
canvas
img
CoControllers
Colinks
slowerCount
top
stopEvent
packetCount
CoSwitches
CoHosts
listNodes
packets
venv.include.upgradeClass.SniffingOnAllNetwork
__init__(self, master,net)
sniffFunction(self)
interfacesFind(self)
allNetworkTraffic(self,packet)
pingWay(self,packet)
pingWayChangeColor(self)
sniffStopFunctionStart(self)
sniffStopFunction(self)
changeColorLink
pigwayThread
timer
e
sniffStopFilter
count
net
sniffStopThread
sniffThread
master
venv.include.upgradeClass.TablesOnSwitch
__init__(self,master,switch)
quit(self,event)
ovsDumpTables(self)
textTables
top
switch
venv.include.upgradeClass.PingInterfaces
__init__(self, master, host, hosts)
sendPing(self)
okButton(self)
quitButton(self)
top
hosts
host
options
selectionHostIp
pingEntry
venv.include.upgradeClass.FlowsOnSwitch
__init__(self, master, switch)
quit(self,event)
ovsDumpFlows(self)
canvas
top
textLabel
switch
Thread
venv.include.upgradeClass.MyThreadingClass
__init__(self, fonk)
startThread(self)
stopThread(self)
isAliveThread(self)
t

3. SONUÇLAR

Software Defined Network teknolojisi çok yeni bir teknolojidir. Bu teklonoloji ağ ve internet alanına yepyeni bir soluk kazandırmış ve ileriki senelerde kullanımı daha da yaygınlaşacaktır. Miniedit uygulaması ise SDN teknolojisini anlamaya ve geliştirmeye yardımcı olan bir uygulamadır. Geliştirilen bu proje ile Miniedit uygulaması daha kullanışlı bir hale gelmiştir. YTA, OpenFlow protokolü ve OpenFlow mesajları daha anlaşılır olmuştur.

4. KAYNAKLAR

[1] İnternet sitesi: http://bit.ly/2CONwpA

[2] İnternet sitesi: http://bit.ly/2Rxmn2y

[3] İnternet sitesi:http://bit.ly/2BXBdpd

[4] İnternet sitesi: https://tr.wikipedia.org/wiki/Wireshark

[5] İnternet sitesi: https://scapy.net/

[6] internet sitesi: http://www.openvswitch.org/support/dist-docs/ovs-dpctl.8.txt

[7] internet sitesi:http://www.openvswitch.org/support/dist-docs/ovs-ofctl.8.txt

[8] internet sitesi: http://www.tcpipguide.com/free/t_ARPMessageFormat.htm