YTA Kullanarak Arızaya Dayanıklı Veri Düzlemi
Fault Tolerant Data Plane Using SDN

Barış Yamansavaşçılar*, Ahmet Cihat Baktır*, Atay Özgövde† ve Cem Ersoy*
*Boğaziçi Üniversitesi Bilgisayar Mühendisliği
{baris.yamansavascilar, cihat.baktir, ersoy}@boun.edu.tr
†Galatasaray Üniversitesi Bilgisayar Mühendisliği
aozgovde@gsu.edu.tr

***Özetçe* —İnternet teknolojisindeki son gelişmeler, Yazılım Tanımlı Ağlar'ın (YTA) öneminin artmasına sebep olmuştur. Ağı merkezi olarak kontrol eden bu yeni ağ modelinin büyük faydalarından dolayı, pek çok servis sağlayıcı ve üretici, YTA'nın geleneksel ağların yerine geçmesini beklemektedir. Ancak, merkezileştirilmiş yapısından dolayı, hem veri hem de kontrol düzleminde güvenilirlik ve arızaya dayanıklılık sorunları göz önüne alındığında, savunmasızdır. Sonuç olarak, bu tarz arızaya dayanıklı bir YTA tasarımının geliştirilmesi oldukça önemlidir. Bu çalışmada, çeşitli ağ ve başarım ölçümleri düşünülerek, veri düzlemindeki arıza dayanıklılığı üzerinde durulmuştur. Deneylerde, topoloji büyüklüğünün, paketlerin geliş sıklığının ve mevcut rotada bulunan akış sayısının toparlanma süresine olan etkisi test edilmiştir. Ayrıca, yerel ve bütünsel toparlanma yaklaşımları karşılaştırılmıştır.**

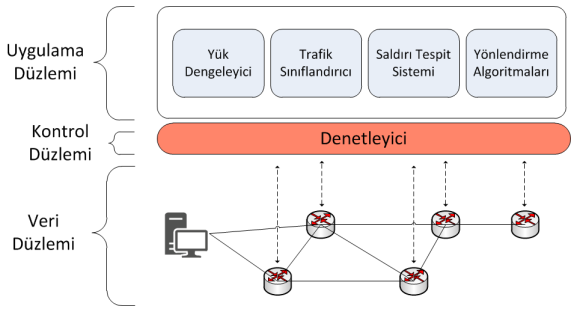
Anahtar Kelimeler—arızaya dayanıklılık, yazılım tanımlı ağlar, veri düzlemi

***Abstract*—Recent developments in Internet technology have led to an increased importance of Software-Defined Networks (SDN). Due to advantages of this new network model that controls the network centrally, many service providers and vendors expect that traditional networks should be superseded by SDN. However, because of their centralized nature, they are vulnerable in terms of reliability and fault-tolerance issues both on data and control planes. Thus, developing such a fault-tolerant SDN design is quite important. In this study, fault tolerance on the data plane is targeted by considering various network and performance measurements. In the experiments, the impact of the topology size, frequency of packets, and the number of flows in the current route on the recovery time is tested. Moreover, local and global recovery approaches are compared.**

Keywords—fault tolerance, software-defined networks, data plane

I. Giriş

Yazılım tanımlı ağ, kontrol ve veri düzlemlerinin birbirinden ayrıldığı yeni bir ağ mimarisidir. Bu ayrıma ek olarak, uygulama düzlemi de kontrol düzleminden ayrılmıştır ve Şekil 1'de gösterildiği gibi ayrı bir katman olarak kullanılmaktadır. Bu yeni ağ modelinde, kontrol düzlemi mantıksal olarak merkezileştirilmiştir ve veri düzlemi basit yönlendirme cihazlarından (anahtarlayıcı) ibarettir. Bu merkezileştirilmiş yaklaşımından dolayı, merkezde, çalışan uygulamaya bağlı olarak ağ hakkında kararlar veren ve bu kararları anahtarlayıcılara ileten bir denetleyici bulunmaktadır. Böylece, bu programlanabilir ağ kavramını kullanarak ağı yönetmek, geleneksel yönteme göre daha kolay yapılmaktadır.



Şekil 1: Yazılım tanımlı ağ mimarisi

YTA'da, denetleyici ve anahtarlayıcı cihazlar arasındaki iletişim güney uygulama programı arayüzü (UPA) kullanılarak gerçekleştirilmektedir. Günümüzde, denetleyicinin bir anahtarlayıcı ile iletişimini sağlamak için, fiili standart olan OpenFlow [1] protokolü kullanılmaktadır. OpenFlow'un kullanılması, açık kaynak bir protokol kullanarak farklı anahtarlayıcılar içindeki akış tablolarını programlamaya imkan tanımasından kaynaklanmaktadır. Böylece, araştırmacılar akışları kontrol edebilmektedir. Ancak, diğer taraftan, kuzey UPA olarak adlandırılan, denetleyici ve uygulama düzlemi arasındaki iletişim için herhangi bir standart bulunmamaktadır.

YTA yaklaşımının geleneksel ağ mimarisine göre çeşitli faydaları vardır. İlk olarak, bu yeni yaklaşım, programlama yapmaya imkan tanımasıyla, ağ üzerinde büyük bir kontrol olanağı sağlar [2]. Örneğin, geleneksel ağlarda, bir program parçasının gerçeklenmesi için ağ üzerindeki tüm cihazlara ayrı çözüm uygulanması gerekmektedir. Diğer taraftan, YTA'da, denetleyici ağ üzerindeki her bir cihazın akış tablosuna uygun kuralları göndererek ilgili program parçasının çalışmasını sağlamaktadır. Bu doğrultuda, ağ üzerindeki tüm uygulamalar aynı bütünsel ağ bilgisine sahip olmaktadır. Ayrıca, farklı uygulamaların bütünleşmesi daha kolay olmaktadır [3].

Tüm faydalarına rağmen, YTA'da çeşitli zorluklar vardır. Bunlardan biri arızaya dayanıklılığıdır, yani ağın çalışmaya devam edebilme yeteneğidir. YTA'da bu durum üç alanda değerlendirilmektedir [4]: (1) veri düzlemi (anahtarlayıcı ya da bağlantı hataları), (2) kontrol düzlemi (anahtarlayıcı-denetleyici bağlantısındaki hatalar), ve (3) denetleyicinin kendisi. Bu çalışmada, veri düzlemi üzerindeki arızaya dayanıklılık üzerine odaklanılmıştır. Anahtarlayıcılar arasındaki bağlantı aksaklıkları baz alınarak, (i) topoloji büyüklüğünün, (ii) paketlerin geliş sıklığının ve (iii) mevcut

rotadaki akış sayısının toparlanma süresine olan etkisi yerel toparlanma yaklaşımı kullanılarak araştırılmıştır. Buna ek olarak da, yerel toparlanma ve bütünsel toparlanma yaklaşımları arasında karşılaştırma yapılmıştır. Çalışmanın ayrılıkçı yönünü, bu özelliklerin etkilerinin incelenmesi oluşturmaktadır.

II. İlgili Çalışmalar

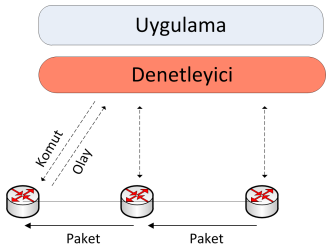
Arızaya dayanıklılık, YTA'da önemli bir sorundur. Reitblatt ve ekibi, YTA'da arızaya dayanıklı ağ programları yazmak için yeni bir dil önermiştir [5]. Düzenli ifadelere dayanan bu dil, geliştiricilere, arızaya dayanıklılık derecesinin gerektirdiği şekilde, paketlerin ağ üzerinde ilerleyeceği yolu belirlemesini sağlamaktadır. James Kempf ve ekibinin çalışmasında [6] OpenFlow anahtarlayıcılarını izleyen bir fonksiyon, denetleyiciye bir işlem yükü getirmeden, izleme mesajlarını yaymak için gerçeklenmiştir. Yazarlar bu çalışmada, bu izleme fonksiyonunu desteklemek için OpenFlow 1.1 protokolünü genişletmeye odaklanmıştır. Yaptıkları deneyler, bu fonksiyon kullanılarak, veri düzlemindeki arızaların toparlanmasının 50 milisaniye içerisinde gerçekleştirilebileceğini göstermiştir.

Arızaya dayanıklılık ile ilgili, veri düzlemine ek olarak, kontrol düzlemini ve denetleyicileri baz alan pek çok çalışma yapılmıştır. Koponen ve ekibi, fiziksel olarak dağıtık çalışan ve ağ durumunu saklamak için dağıtık depoları kullanan Onix denetleyicisini tanıtmışlardır [7]. Diğer taraftan, fiziksel olarak dağıtık ancak mantıksal olarak merkezileştirilmiş denetleyici olan ONOS, Onix'den dört yıl sonra tanıtılmıştır [8]. Benzer şekilde, Botelho ve ekibi, kendi mimarileri olan SMaRtLight'ı gerçekleştirmek için çoğaltılan durum makinesi olarak bir veri deposu kullanmışlardır [9]. Başka bir çalışmada [10] yazarlar, yayımla-takip et mesajlaşma örneğini denetleyiciler arasında kullanarak ağ olaylarını denetleyici kopyalarına çoğaltmışlar ve bu sistemi önerdikleri HyperFlow denetleyicisinde kullanmışlardır. Katta ve ekibinin çalışmasında [11] ise bu yaklaşım genişletilerek, olası denetleyici hatalarına karşı anahtarlayıcı durumunun tutarlılığı ele alınmış ve bu doğrultuda iki aşamalı çoğaltma tekniği öne sürülmüştür.

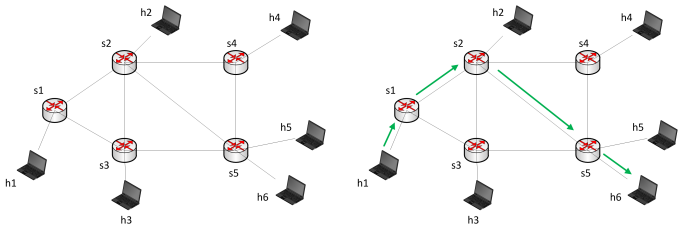
Bu çalışmada tek bağlantı arızaları üzerinde durulmuş ve bir önceki bölümde bahsedilen üç değişkenin toparlanma süresine olan etkisi incelenmiştir. Bildiğimiz kadarıyla, literatürde bu değişkenlerin etkisini birlikte inceleyen bir çalışma bulunmamaktadır.

III. Arızaya Dayanıklı Sistem Tasarımı

Sistem, Ryu [12] denetleyicisi kullanılarak, bir kuzey arayüzü uygulaması olarak gerçekleştirilmiştir. Denetleyici, veri düzlemindeki tüm anahtarlayıcılara doğrudan bağlanmıştır. Dolayısıyla, ağ olaylarından bütünsel şekilde (anahtarlayıcılardan gelen olaylar, bağlantıların durumu vb.) haberdar olabilmektedir. Bu sebeple, bir bağlantı arızası olduğu zaman yeni yol buluşsal tekniklerle belirlenebilir. Bu doğrultuda, biz de çalışmamızda, bağlantı arızası sonrasında yeni yolu bulmak için, buluşsal tekniklerden olan fırsatçı algoritmayı kullandık. Ancak, şu andaki kurulumda, tüm bağlantıların aynı maliyete sahip olması sebebiyle algoritma, doğası gereği enine arama şeklinde davranmıştır.



Şekil 2: YTA'da olay ve komut mekanizması



(a) Küçük topoloji (b) Küçük topolojideki rota

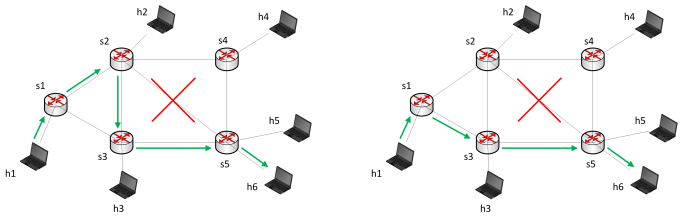
Şekil 3: Verilen ağ topolojisinde 2 bilgisayar arasında rota oluşturma

A. Rota Oluşturma

YTA'da, yeni bir akışın ilk paketi anahtarlayıcıya ulaştığı zaman, anahtarlayıcı, kendi akış kuralları tablosunda henüz bir bilgi olmadığı için, denetleyiciye bu akışın paketleriyle ilgili ne yapması gerektiğini danışır. Şekil 2'de gösterilen bu işleme olay adı verilirken, denetleyicinin verdiği cevap komut olarak isimlendirilir. Sonrasında, anahtarlayıcı, denetleyiciden gelen kuralları işler ve bunları kendi tablosuna yerleştirir. Bu işlem, anahtarlayıcı her yeni akış ile karşılaştığında tekrar edilmektedir. Ancak, bu çalışmada, bu mekanizma sadece yeni akışın uğradığı ilk anahtarlayıcı için kullanılmıştır. Gerçekleştirilen sisteme ağ topolojisi önceden verildiği için, ilgili rota tetiklenen ilk olaydan sonra hesaplanmakta ve ilgili kurallar tüm anahtarlayıcılara denetleyici tarafından gönderilmektedir. Böylece işlemci kullanımı ve rota oluşturma süresi azaltılmıştır. Örneğin, Şekil 3 (a)'da verilen ağ için, eğer bilgisayar h1, bilgisayar h6 ile iletişim kurmak isterse, aralarındaki rota Şekil 3 (b)'deki gibi olmaktadır.

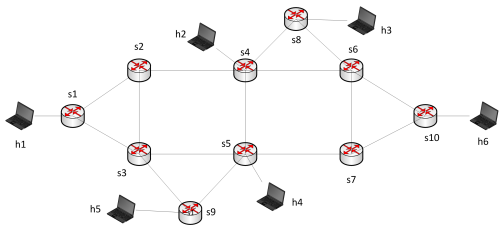
B. Hızlı Toparlanma

Şekil 3 (b)'de bilgisayar h1 ve h6 için rotası çizilen ağ topolojisinde, bir süre sonra, anahtarlayıcı s2 ve s5 arasındaki bağlantıda (akışın geçtiği yerde) bir arıza oluştuğunu düşünelim. Ryu denetleyicisi, arıza oluştuktan sonra bağlantıyla ilgili gelen belirli sayıdaki LLDP (Link Layer Discovery Protocol) paketlerindeki gecikmeye bağlı olarak mevcut bağlantıda bir sorun olduğunu tespit etmektedir. Bu tespitin ardından kullanılabilecek iki tane yaklaşım vardır: (1) yerel toparlanma, (2) bütünsel toparlanma. Yerel toparlanmada, topolojide daha az akış tablosunda değişiklik yapılarak arızadan kurtulabilmek için, yeni rota Şekil 4 (a)'da gösterildiği gibi anahtarlayıcı s2'den itibaren hesaplanmaktadır. Bütünsel toparlanmada ise mevcut topolojideki en etkili (kısa) yolu kullanabilmek için, yeni rota Şekil 4 (b)'de gösterildiği gibi bilgisayar h1'den itibaren belirlenmek-



(a) Yerel toparlanma (b) Bütünsel toparlanma

Şekil 4: Mevcut rotadaki bağlantıda arıza olduğu zaman toparlanma



Şekil 5: Büyük ölçekli topoloji

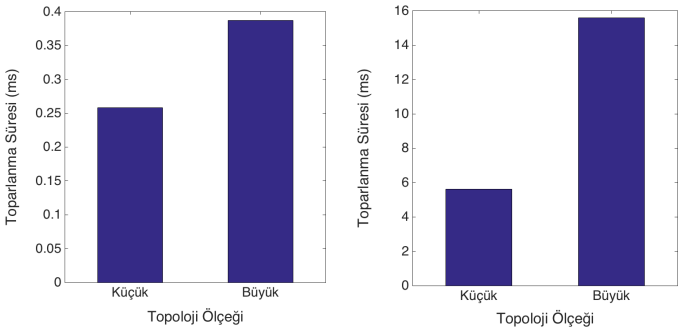
tedir. Bu çalışmada, iki yaklaşımın karşılaştırılması haricindeki tüm deneyler yerel toparlanma yaklaşımı kullanılarak gerçekleştirilmiştir.

IV. Deneysel Sonuçlar

Yapılan deneylerde (1) ağ topolojisinin büyüklüğünün, (2) paketlerin geliş sıklığının ve (3) rotadaki akış sayısının toparlanma süresine olan etkisini yerel toparlanma yaklaşımını kullanarak inceledik. Ayrıca, bütünsel yaklaşım ile yerel yaklaşımın bir karşılaştırmasını yaptık. Bu amaçla, iki tane araç kullandık: Mininet [13] ve D-ITG (Distributed Internet Traffic Generator) [14]. Mininet, veri düzlemini anahtarlayıcıları kullanarak oluşturan ve bu düzlem ile denetleyici arasındaki etkileşimi sağlayan bir öykünüm aracıdır. Bu araç, Python UPA'yı kullanarak kullanıcılara özelleştirilebilir ağ yaratma imkanı verdiğinden dolayı, geliştirdiğimiz sistemi farklı ağ ölçekleri için test edebilmemizi sağlamıştır. Diğer taraftan, D-ITG ise akışların uzunluk, protokol vb. özelliklerinin ayarlanabildiği bir trafik üretme aracıdır. Bu çalışmada, akış sayısının etkisinin incelenmesi haricindeki deneylerde bir akış kullanılmıştır. Tüm deneylerdeki akışlar ise 15 saniye uzunlukta olup, UDP (User Datagram Protocol) paketlerinden oluşmaktadır. Ayrıca, tüm deneyler 10 kere tekrar edilmiştir. Bu doğrultuda, tüm senaryolarda test sonuçlarının elde edilmesi için toplamda 80 deney gerçekleştirilmiştir.

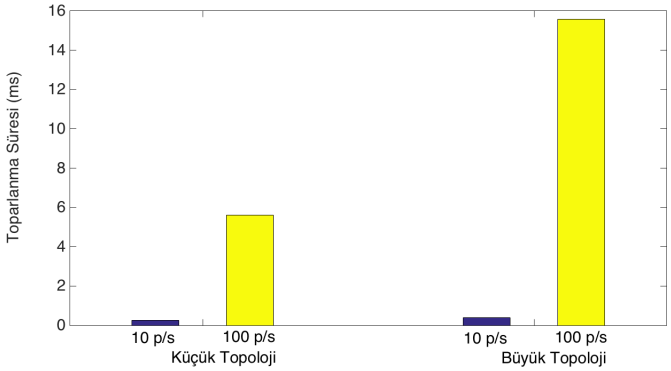
A. Topoloji Büyüklüğünün Etkisi

Topoloji büyüklüğünün toparlanma süresine olan etkisini değerlendirmek için küçük ve büyük ölçekli olmak üzere iki topoloji kullandık. Her iki topoloji için, ilk olarak iki bilgisayar arasında bir iletişim başlattık ve ilgili rotayı oluşturduk. Sonrasında ise, iletişim (akış) devam ederken, mevcut rota üzerinde bulunan bir bağlantıyı bozduk ve bozulan iletişimlerin toparlanma sürelerini değerlendirdik.



(a) 10 p/s için (b) 100 p/s için

Şekil 6: Topoloji ölçeğinin toparlanma süresine etkisi



Şekil 7: Paket sıklığının toparlanma süresine etkisi

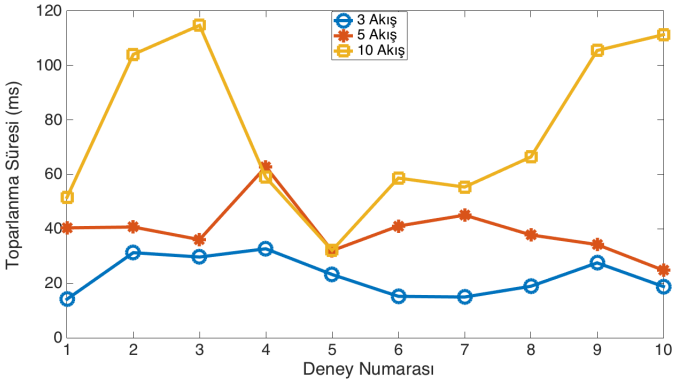
Şekil 3 (a)'da gösterilen küçük topoloji için bilgisayar h1 ve h6 arasında iletişim kurularak, anahtarlayıcı s2 ve s5 arasındaki bağlantı bozulmuştur. Şekil 5'deki büyük ölçekli topoloji için ise bilgisayar h1, bilgisayar h6 ile iletişimde bulunurken anahtarlayıcı s2 ile s4 arasındaki bağlantı koparılmıştır. Her iki topoloji için, 10 p/s (saniyedeki paket sayısı) ve 100 p/s'deki durumlar baz alınarak, test sonuçları Şekil 6'da gösterilmiştir. Topoloji ölçeği arttıkça toparlanma süresinin de arttığı net şekilde görülmektedir. Bu durum, algoritma tarafından yol bulmak için hesaba katılan anahtarlayıcı ve bağlantı sayılarının artmasından kaynaklanmaktadır.

B. Paketlerin Geliş Sıklığının Etkisi

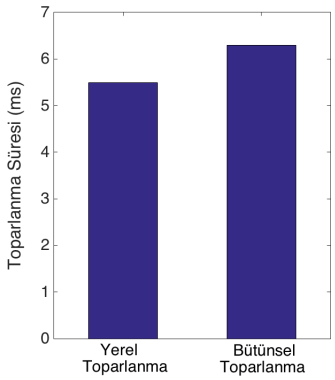
D-ITG trafik üreteci, ilk olarak, verilen akışın paket sıklığını belirlemek için kullanılmıştır. Bu amaçla paketler, kaynaktan varış noktasına doğru 10 p/s ve 100 p/s olacak şekilde gönderilmiştir. Paket sıklığının etkisinin test edilmesi sürecinde topoloji büyüklükleri baz alınmıştır. Şekil 7'deki sonuçlarda görüldüğü üzere paket sıklığı artınca, her iki topolojideki toparlanma sürelerinde artış meydana gelmiştir. Sürenin artmasının sebebi, anahtarlayıcılara birim süre için daha fazla paket geldiğinden dolayı, paketlerin ara depolamada (buffer) daha uzun süre kalmasıdır.

C. Akış Sayısının Etkisi

Mevcut rota üzerindeki akış sayısının toparlanma süresine olan etkisini incelemek için 3, 5 ve 10 akıştan oluşan ağ yükleri ile deneyler gerçekleştirdik. Her bir senaryoda, önceki durumlara benzer şekilde, trafik mevcut rotadan geçerken bağlantı



Şekil 8: Akış sayısının toparlanma süresine etkisi



Şekil 9: Yerel ve bütünsel toparlanma sürelerinin karşılaştırılması

arızası oluşturduk ve toparlanma süresini ölçtük. Deneyleri, küçük topolojide ve her bir akış 100 p/s olacak şekilde gerçekleştirdik. Bu senaryo için bütün deneylerin sonuçlarının gösterildiği Şekil 8'de görüldüğü üzere, bir trafikteki akış sayısı fazlalaştıkça ortalama toparlanma süresi ve bu sürenin değişkenliğinde de artış meydana gelmektedir. Geliş sıklığının etkisine benzer şekilde, akış sayısı arttıkça anahtarlayıcıya birim zamanda gelen paket sayısı arttığından, toparlanma süresi de buna bağlı olarak yükselmektedir.

D. Bütünsel Toparlanma

Bütünsel toparlanma, bağlantı hatası olduktan sonra, iletişimde bulunan iki bilgisayar arasında, maliyet yönünden en uygun rotayı bulmaktadır. Maliyet, ağ topolojisinin bağlantı ve anahtarlayıcı özelliklerine göre değişmekte olup, bu çalışma için sekme sayısıdır. Bu sebeple, rota bulma algoritmasının daha az anahtarlayıcıyı kullanmasını sağlayan yerel yaklaşım ve sekme sayısını en aza düşüren bütünsel yaklaşımı toparlanma zamanı yönünden küçük topolojiyi kullanarak karşılaştırdık. Topolojide arıza oluşması ve yeni rota belirlenmesi, her iki yaklaşım için Şekil 4 (a) ve Şekil 4 (b)'deki gibi olmaktadır. Her iki yöntemin toparlanma süresine etkisi ise Şekil 9'da gösterilmiştir. Bütünsel toparlanmada s1 anahtarlayıcısına geri dönüldüğü ve bu esnada eski rotada bulunan anahtarlayıcılarda ilgili akış kuralları silindiği için toparlanma zamanı daha yüksek çıkmaktadır.

V. Sonuç

Bu çalışmada, veri düzlemini baz alarak YTA'daki arıza dayanıklılığını inceledik. Bunu yaparken, arıza gerçekleştikten sonra sistemin toparlanma süresini etkilemesi beklenen topoloji büyüklüğü, saniyede gelen paket miktarı ve akış sayısı değişkenlerinin etkisini yerel toparlanma yaklaşımını kullanarak araştırdık. Gerçekleştirdiğimiz deneyler sonucunda bu üç değişkenin de toparlanma süresini etkilediğini gördük. Buna ek olarak, bütünsel yaklaşımı yerel yöntem ile karşılaştırdık. Gelecekte, gerçek dünya koşullarına daha yakın olmak için, bu çalışmayı gerçek cihazlarla çalışacak şekilde genişletmeyi ve sonuçları değerlendirmeyi düşünüyoruz. Buna ek olarak, birden fazla bağlantı arızası, anahtarlayıcı problemi ve dağıtık denetleyicileri içeren durumları incelemeyi planlıyoruz. Ayrıca, linklerin güvenirliğinin önceden bilinebildiği durumlarda, yeniden yönlendirme algoritmasının daha güvenilir yolları tercih etmesi konusunda da testler yapmayı düşünüyoruz.

Kaynakça

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow," ACM SIGCOMM Computer Communication Review, vol. 38, no. 2, p. 69, Mar. 2008.

[2] W. Xia, Y. Wen, C. H. Foh, D. Niyato, and H. Xie, "A Survey on Software-Defined Networking," IEEE Communications Surveys & Tutorials, vol. 17, no. 1, pp. 27–51, Jan. 2015.

[3] M. Casado, N. Foster, and A. Guha, "Abstractions for software-defined networks," ACM Commun., vol. 57, no. 10, pp. 86–95, Sep. 2014.

[4] Hyojoon Kim, M. Schlansker, J. R. Santos, J. Tourrilhes, Y. Turner, and N. Feamster, "CORONET: Fault tolerance for Software Defined Networks," in 2012 20th IEEE International Conference on Network Protocols (ICNP), 2012, pp. 1–2.

[5] M. Reitblatt, M. Canini, A. Guha, and N. Foster, "FatTire: Declarative Fault Tolerance for Software-Defined Networks," in Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking - HotSDN '13, 2013.

[6] James Kempf, Elisa Bellagamba, Andras Kern, David Jocha, Attila Takacs, and Pontus Sköldström, "Scalable Fault Management for OpenFlow", in IEEE International Conference on Communications (ICC), 2012.

[7] T.Koponen,M.Casado,N.Gude,J.Stribling,L.Poutievski,M.Zhu,R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker, "Onix: A Distributed Control Platform for Large-scale Production Networks," in OSDI, Oct. 2010.

[8] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, and G. Parulkar, "ONOS: Towards an Open, Distributed SDN OS," in HotSDN, Aug. 2014.

[9] F.Botelho,A.Bessani,F.M.V.Ramos,andP.Ferreira,"On the Design of Practical Fault-Tolerant SDN Controllers," in 2014 Third European Workshop on Software Defined Networks, 2014, pp. 73–78.

[10] A. Tootoonchian and Y. Ganjali, "HyperFlow: A Distributed Control Plane for OpenFlow," in INM/WREN, Apr. 2010.

[11] N. Katta, H. Zhang, M. Freedman, and J. Rexford, "Ravana: Controller Fault-Tolerance in Software-Defined Networking," in Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research - SOSR '15, 2015, pp. 1–12.

[12] Ryu OpenFlow controller, available at http://osrg.github.io/ryu/

[13] R. L. S. de Oliveira, C. M. Schweitzer, A. A. Shinoda, and Ligia Rodrigues Prete, "Using Mininet for emulation and prototyping Software-Defined Networks," in 2014 IEEE Colombian Conference on Communications and Computing (COLCOM), 2014, pp. 1–6.

[14] D-ITG, http://traffic.comics.unina.it/software/ITG/