ÖZ-DÜZENLEMELİ HARİTA (SOM) KULLANARAK DDoS SALDIRILARININ SINIFLANDIRILMASI

Safai TANDOĞAN

Yüksek Lisans Tezi

Fen Bilimleri Enstitüsü
Elektrik Elektronik Mühendisliği Anabilim Dalı

Mayıs – 2007

JÜRİ VE ENSTİTÜ ONAYI

Safai TANDOĞAN'ın **Öz-Düzenlemeli Harita Kullanarak DDoS Saldırılarının Sınıflandırılması** başlıklı **Elektrik Elektronik Mühendisliği** Anabilim Dalındaki, Yüksek Lisans tezi 16/05/2007 tarihinde, aşağıdaki jüri tarafından Anadolu Üniversitesi Lisansüstü Eğitim-Öğretim ve Sınav Yönetmeliğinin ilgili maddeleri uyarınca değerlendirilerek kabul edilmiştir.

		Adı-Soyadı	İmza
Üye (Tez Danışmanı)	:	**Yard. Doç. Dr. EMİN GERMEN**	**……**
Üye	:	**Yard. Doç. Dr. HAKAN G. ŞENEL**	**……**
Üye	:	**Yard. Doç. Dr. CÜNEYT AKINLAR**	**……**

Anadolu Üniversitesi Fen Bilimleri Enstitüsü Yönetim Kurulu'nun …… tarih ve …… sayılı kararıyla onaylanmıştır.

Enstitü Müdürü

ÖZET

Yüksek Lisans Tezi

ÖZ-DÜZENLEMELİ HARİTA (SOM) KULLANARAK DDOS SALDIRILARININ SINIFLANDIRILMASI

Safai TANDOĞAN

Anadolu Üniversitesi
Fen Bilimleri Enstitüsü
Elektrik-Elektronik Mühendisliği Anabilim Dalı

Danışman: Yard. Doç. Dr. Emin GERMEN

2007, 57 sayfa

Bu tezde, DDoS saldırılarının, Öz-Düzenlemeli Harita kullanılarak, ağ üzerindeki normal trafikten ayrıştırılması üzerine çalışılmıştır. OMNeT++ ağ simülasyon motoru ve INET ağ simülasyonu kütüphanesi kullanılarak bir DDoS saldırısı simüle edilmiş ve bu saldırı sırasında ortaya çıkan paketler kaydedilmiştir. Ağ üzerindeki paketlerin, IP ve TCP başlığından kaynak kapı, hedef kapı, yük uzunluğu ve pencere boyutu parametreleri seçilmiştir. Bu parametreler her bir paket için bir girdi vektörü haline getirilmiştir. Hazırlanan bu vektörler bir Öz-Düzenlemeli Harita kullanılarak sınıflandırılmaya çalışılmıştır. Öz-Düzenlemeli Harita sonucunda oluşan ağ LVQ3 algoritması kullanılarak gruplanmış ve normal trafiğin, saldırı trafiğinden başarıyla ayrıştırılabildiği gözlenmiştir. Çıkan sonuçlar ışığında geliştirilen yöntemin nasıl gerçeklenebileceği tartışılmıştır.

Anahtar Kelimeler: Öz-Düzenlemeli Harita, LVQ3, DDoS, OMNeT++, INET, TFN2K

ABSTRACT

Master of Science Thesis

Classification of DDoS Attacks Using Self Organizing Maps, SOM

Safai TANDOĞAN

Anadolu University
Graduate School of Sciences
Electrical and Electronics Engineering Program

Supervisor: Assist. Prof. Dr. Emin GERMEN
2007, 57 pages

In this thesis, classification of DDoS attacks using Self Organizing Maps (SOM) is studied. Using OMNeT++ network simulation engine and INET network simulation library, a DDoS attack network is simulated and during this attack, generated packets are recorded. Using these packets, source port, destination port, payload length, and window size parameters of IP headers, and TCP headers are selected. For each packet, an input vector is created using these parameters. These input vectors have been classified using a self organizing map. Resulting network of SOM has been clustered using LVQ3 algorithm. Using the results, implementation of the developed technique is discussed.

Keywords: Self Organizing Map, LVQ3, DDoS, OMNeT++, INET, TFN2K,

İÇİNDEKİLER

ŞEKİLLER DİZİNİ

ÇİZELGELER DİZİNİ

1. GİRİŞ

Son yıllarda, bilgi, kurumların en değerli varlığı haline gelmiştir ve kurumlar tarafından yürütülen işler büyük ölçüde bu bilgiler kullanılarak gerçekleştirilmektedir. Bu nedenle, ağa bağlı bilgisayarlarda daha önce hiç olmadığı kadar çok bilgi depolanmakta ve işlenmektedir. Dolayısıyla, kurumlarda, bilgiye hızlı erişmek ve bilgiyi yönetmek çok önemli bir olgu haline gelmiştir. Bu süreçte, sistem kaynaklarının güvenilirliği, bütünlüğü ve erişilebilirliği, ağa bağlı bilgisayar sistemlerinin geliştirilmesi ve işletilmesinde temel ilgi alanlarından biri olmuştur. Bilgisayar altyapılarının genişlemesi, bu sistemlerin güvenlik tehditlerine, saldırılara ve izinsiz girişlere karşı kırılganlığını ortaya çıkarmıştır [1].

Servis Yıkımı (Denial of Service, DoS) saldırıları bu bağlamda zorlu bir problem olarak ortaya çıkmaktadır. DoS saldırılarında temel hedef, servisin kendisine zarar vermekten daha çok, servise ya da sunucuya ulaşımı sınırlayarak servisi yıkmaktır. Bu tür saldırılar, ağın, bant genişliği ya da bağlanırlığını hedef alarak, normal servis sağlamaya elverişsiz duruma gelmesine sebep olurlar [2]. *Dağıtık Servis Yıkımı* (Distributed Denial of Service, DDoS), Internet kaynaklarına saldırmak için oldukça basit ama güçlü bir tekniktir. DDoS saldırıları, DoS problemine çok boyutluluk katarak, saldırıların önlenmesi zorlaştırır ve etkisini oransal olarak arttırır.

Eylül 1996'den beri, Internet ortamındaki birçok site, servis yıkımı saldırısıyla karşı karşıya kalmıştır. Bu saldırılardan en önemlisi, TPC/IP (Transmission Control Protocol/Internet Protocol) protokolündeki zayıflıktan faydalanan *SYN Sel* (SYN Flood) saldırısıdır. Bu zayıflık protokolde önemli değişiklikler yapılmadan düzeltilemez. Buna karşın servis yıkımı saldırıları çok kolay bir şekilde başlatılabilmektedir [3]. Bu tür saldırılar, bir saldırganın, kurban makineye sahte kaynak adresler kullanarak çok fazla TCP bağlantı isteği göndermesi mantığıyla çalışır. Her bir istek, hedef alınan makinedeki sınırlı kaynak havuzundan bir kısmının kullanılmasına sebep olur. Hedef makinenin kaynakları tükendiğinde, yeni TCP bağlantıları kurulamaz, böylece asıl kullanıcıların servise erişmesi engellenmiş olur.

Günümüz ağ teknolojileri göz önünde bulundurulduğunda, DDoS saldırıların engellenmesi yönünde çeşitli yöntemlerin denendiği görülmektedir. Ancak saldırının engellenebilmesi için de bu trafiğin normal akış trafiğinden ayrılması gerekmektedir. Sadece saldırı yoğunluğuna göre bir süzme işlemi normalde yoğun trafik yaratabilecek olaylarla geçişime neden olacağından, trafik karakteristiğinin belirlenmesi ve buna göre davranılması kaçınılmazdır. Trafik karakteristiği için de bu trafiği oluşturan paket parametrelerinin sınıflandırılması oldukça akla yakın bir çözümdür.

Gerek Sayısal Sinyal Analizi gerekse Yapay Sinir Ağları alanlarında hem örüntü tanıma hem de sınıflandırma için sayısız yöntem önerilmiş ve önerilmektedir. Yapay Sinir Ağlarının gerek uyarlanabilir yapısı gerekse doğrusal olmayan analizlerdeki başarıları göz önünde bulundurulduğunda, bu çalışmada bir yapay sinir ağı modeliyle trafik karakteristiklerinin belirlenmesi ve sonuçta ayırt edilmesi çalışmanın temel yapısını oluşturmaktadır.

Bu çalışmada, bir Yapay Sinir Ağı modeli olan Öz-Düzenlemeli Harita (Self Organizing Map, SOM) yöntemi deneyler sonucunda elde edilen özellik vektörlerinin sınıflandırılmasında kolaylıkla uygulanabileceği düşünülerek çalışmalar bu yöntem üzerinde yoğunlaştırılmıştır. *Öz-Düzenlemeli Harita*, bir girdi örüntüleri kümesini, bağımsız bir şekilde, değişik sınıflar halinde düzenlemek için öğreticisiz öğrenme tekniği kullanır. Model, genelde iki boyutlu bir nöron dokusu (harita) şeklinde oluşturulur, fakat daha yüksek boyutlu topolojiler de mümkündür [4]. Her bir nörona N boyutlu bir vektör atanır, her bir vektörün, girdi kümesindeki özelliklere karşılık gelen N tane elemanı vardır. Eğitim sonrasında, benzer girdi örüntüleri, düzenli bir harita oluşturur.

Bu çalışmada, TCP paketlerindeki belirli özellikler seçilerek, SOM yardımıyla, DDoS saldırı trafiğinin, normal ağ trafiğinden ayrıştırılması araştırılmıştır. İkinci bölümde, DDoS saldırıları hakkında detaylı bilgi sunulmuş, üçüncü bölümde SOM algoritması incelenmiş, dördüncü bölümde DDoS saldırısının simülasyonu için kullanılan OMNeT++ ağ simülatörü hakkında bilgi verilmiştir. Beşinci bölümde, yapılan simülasyon ve çıkan sonuçlar sunulmuş, son bölümde ise bu sonuçlara dayanılarak ileride ne gibi çalışmalar yapılabileceği hakkında görüş belirtilmiştir.

2. DDOS

2.1 Tanım ve Tarihçe

DoS saldırısı, bir hizmetin asıl kullanıcılarının, o hizmeti kullanmasının, saldırganlar tarafından açıkça engellenmesi çabası olarak tanımlanmaktadır. [5] Örnek olarak:

- Ağ trafiğinin yoğunluğunu aşırı biçimde arttırmak, böylece normal ağ trafiğini engellemek,
- İki cihaz arasındaki bağlantıyı engellemeye çalışmak, böylece hizmete ulaşımı engellemek,
- Herhangi bir kişinin bir hizmete ulaşmasını engellemeye çalışmak,
- Hizmetin özel bir sisteme ya da kullanıcıya ulaşmasını engellemeye çalışmak, gösterilebilir.

Internet topluluğunda DoS saldırılarına özel bir ilgi gösterilir ve bu saldırıların oluşmasından kaygı duyulur. Çünkü DoS saldırları hedef sistemleri etkisiz hale getirmeyi ve/veya hedef ağları ulaşılamaz hale getirmeyi amaçlar. Geleneksel DoS saldırları, genellikle herhangi bir ana bilgisayarı veya ağı kullanarak çok büyük miktarlarda trafik oluşturur. Bu şekildeki saldırılara karşı, saldırıya uğrayan sistemin, bu durumu belirleyip, kendisini savunabilme ihtimali vardır. Dağıtık Servis Yıkımı (DDoS) saldırısı, DoS saldırılarından çok daha tehlikelidir; çünkü bu tür saldırılar bir ya da birden çok hedefe farklı kaynaklardan eşzamanlı olarak yürütülmek üzere tasarlanmışlardır.

DoS saldırıları her zaman iletişim gündemine değişik kılıklarla oturmakta, verdiği zararlarla da sürekli bu gündemi işgal etmektedir. DDoS saldırıları ise daha yeni bir konu olup, ilk saldırılar ve buna bağlı olarak yıkımlar 1999 Haziran ayı sonu ve Temmuz ayı başında görülmüştür. İlk ciddi anlamda kaydı tutulmuş DDoS saldırısı 1999 Ağustos ayında meydana gelmiştir. Bu saldırıda Trinoo

olarak adlandırılan bir DDoS saldırı aracı, en az 114'ü Internet2[1] üzerinde olmak üzere, en az 227 sisteme konuşlandırılmış ve Minnesota Üniversitesine ait bir bilgisayar hedef olarak seçilmiştir. Hedef bilgisayar 2 gün boyunca hizmet dışı kalmıştır.

Kamuoyuna yansıyan ve medyada yer alan ilk DDoS saldırısı ise 2000 Şubat ayında meydana gelmiştir. 7 Şubat'ta *Yahoo* DDoS'a kurban seçilmiş ve 3 saat boyunca Internet portalı erişilemez hale gelmiştir. 8 Şubat'ta *Amazon*, *Buy.com*, *CNN* ve *eBay* DDoS saldırısına uğramış ve bunun sonucunda tamamıyla işlevlerini kaybetmişler ya da belirgin bir şekilde yavaşlamışlardır. Son olarak 9 Şubat'ta *E*Trade* ve *ZDNet*, DDoS saldırılarına maruz kalmışlardır. Analistlerin tahminlerine göre *Yahoo,* ulaşılamaz kaldığı 3 saat boyunca, 500.000$ civarında bir e-ticaret ve reklam geliri kaybına uğramıştır. Kitap satıcısı *Amazon.com*, kendi açıklamasına göre ulaşılamaz olduğu 10 saat boyunca 600.000$ kayba uğramıştır. DDoS saldırıları boyunca, *Buy.com*' un kullanılabilirliği %100 den %9.4 düşmüş, *CNN.com*'un kullanıcıları normal değerinin %5'ine gerilemiş ve *ZDNet* ve *E*Trade.com* siteleri hemen hemen erişilemez olmuşlardır.

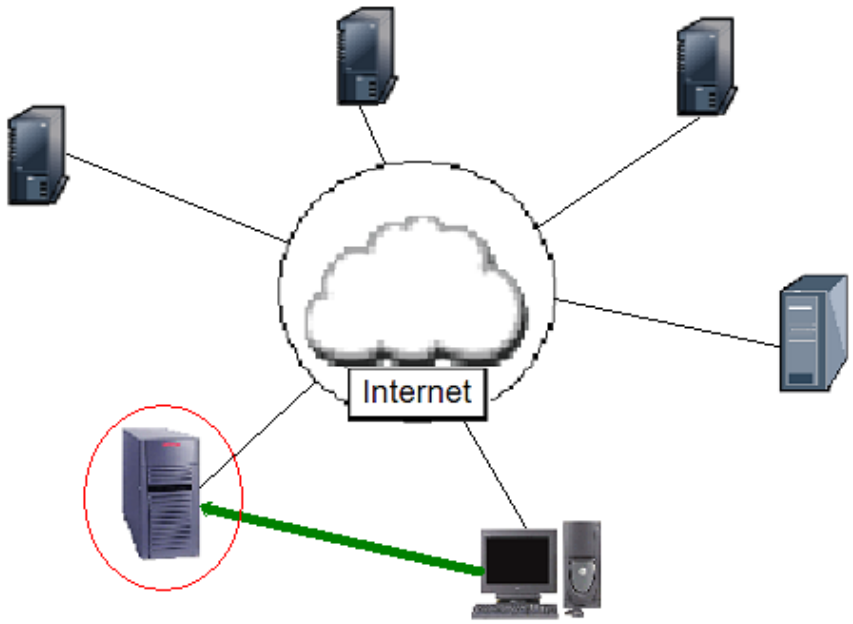
2.2 Saldırı Senaryosu

DDoS saldırılarının yapısı DoS yapısıyla karşılaştırıldığında daha karışıktır ve böyle bir saldırı için her zaman birden çok işlev yükümlenmiş aracıya gereksinim duyulur. Tipik bir DDoS saldırı senaryosu şu şekilde gelişir:

1. *Davetsiz kullanıcılar*, Internet üzerinde ele geçirilip üzerine konuşlanılacak bir ya da birden çok sistem bulur. Bu işlem genellikle çok sayıda kullanıcısı ve/veya dikkatsiz yöneticileri bulunan, tercihen Internet'e geniş bant ile bağlı sistemlerden çalınan hesap bilgileri kullanılarak başarılır. Kolej ve üniversite yerleşkelerinde bu tanıma

1 Internet2 yada UCAID(University Corporation for Advanced Internet Development) eğitim ve yüksek-hızda veri transferi amaçlı, üst seviyede ağ uygulamaları geliştiren ve kullanan, kar amacı olmayan bir birliktir.

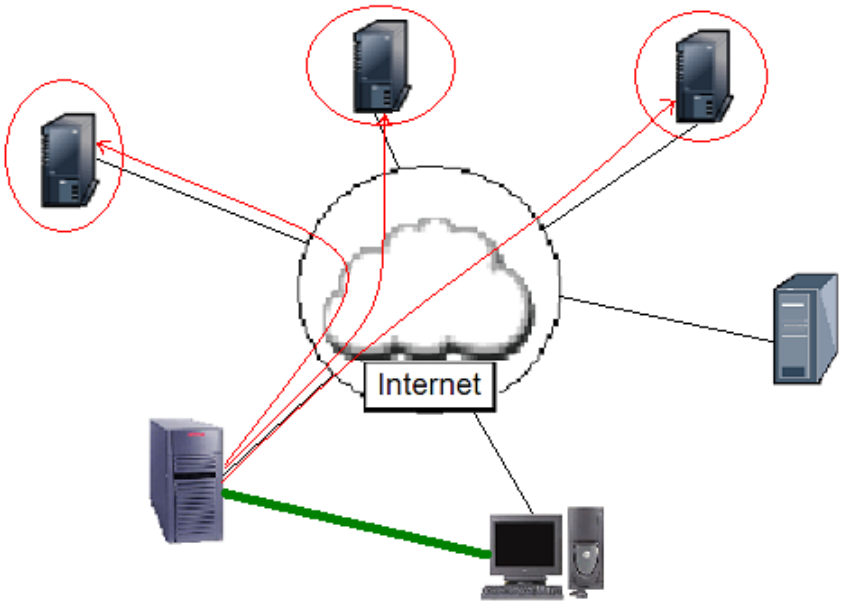
uyan sistemler bulunmaktadır. Şekil 2.1 DDoS saldırı senaryosunun ilk aşamasını göstermektedir.



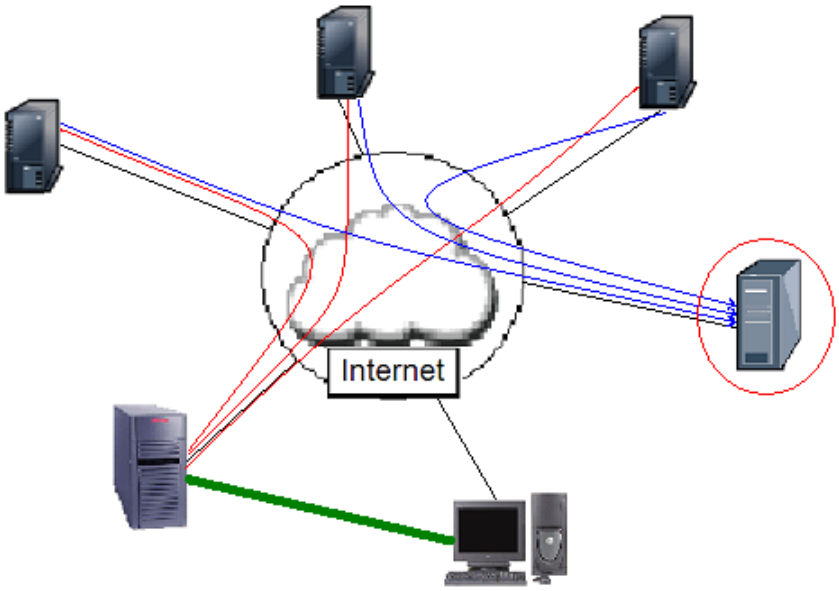
Şekil 2.1. Davetsiz kullanıcı, üzerine konuşlanılacak sistemi bularak saldırıyı koordine eder.

2. Ele geçirilen sisteme tarayıcılar, faydalanma (exploit) araçları, işletim sistemi dedektörleri, kök (root) kitleri ve DoS/DDoS programları gibi birçok korsan ve kırıcı programlar yükler. Bu sistem DDoS denetçisi olur. Denetçi yazılım, ele geçirilebilecek ve daha fazla konuşlanılabilecek diğer sistemlerin bulunmasına yardımcı olur. Saldırgan güvenlik açığı bulunan servisler çalıştıran sistemleri bulmak için geniş aralıklı IP adres bloklarını tarar. Bu ilk işgal (mass-intrusion) evresi, yüzlerce ana makineyi ele geçirmek ve bu makinelere DDoS ajanları yüklemek için otomatikleşmiş araçlar kullanır. Bu evrede kullanılan araçlar DDoS araç kitine dahil değildir fakat yasadışı korsan grupları arasında kullanımı yaygın olan araçlardır. Bu ele geçirilen sistemler DDoS saldırısının ilk kurbanları olurlar. Sonradan ele geçirilen bu sistemlere asıl saldırıyı gerçekleştirecek olan DDoS hayalet programları (daemon) yüklenir. Şekil 2.2 DDoS saldırı senaryosunun ikinci aşamasını göstermektedir.

3. *Davetsiz kullanıcı* ele geçirdiği, DDoS hayalet programlarının yüklü olduğu sistemlerin bir listesini tutar. Asıl servis yıkımı saldırısı evresi, saldırganın DDoS hayalet programlarına, ana sistemde bulunan denetçi programını çalıştırarak, saldırıyı başlatma komutunu göndermesiyle başlar. DDoS kurbanı bu aşamada senaryoya dahil olur. Şekil 2.3 DDoS saldırı senaryosunun üçüncü aşamasını göstermektedir.



Şekil 2.2. DDoS denetçisi, DDoS hayalet programlarını yükleyeceği başka sistemler bulur.



Şekil 2.3. DDoS denetçisinden gelen komutla, hayalet programlar saldırıya başlar.

Denetçi ve hayalet programlar arasındaki trafik karıştırılabilir, dolayısıyla denetçi bilgisayarın bulunması zorlaştırılır. DDoS ağı içerisinde denetçinin konumuyla ilgili bir ya da birkaç makinede bazı deliller bulunabilir, fakat hayalet programlar otomatik işlem yaptıkları için denetçi ve geri kalan DDoS ağı arasında süregelen bir iletişim olmasına gerek yoktur. Aslında, DDoS ağı içerisinde denetçinin konumunu ve kimliğini gizlemek için genellikle farklı yöntemler kullanılır. Bu yöntemler devam eden bir saldırının incelenmesini ve ayrıca saldırı trafiğinin engellenmesini ve kaynağının izinin sürülmesini zorlaştırır.

Çoğu durumda, bulaşılan sistemlerin yöneticileri, denetçi programların kendi sistemlerinde bulunduğundan habersizdir. Her hangi bir tarayıcı program kullanıp bu zararlı DDoS yazılımını bulup, yok etseler bile, bu yazılımın başka hangi sistemlere yüklendiği konusunda bir bilgiye ulaşmaları son derece zordur. İstismara uğrayan popüler sistemler sitelerin, Web, E-posta, alan-adı, ve diğer sunucularını içerir. Bu tür sistemlerin çok sayıda açık kapıları (port) bulunur, yüksek miktarda trafik üretirler ve de saldırı bu sunuculara kadar izlense bile, yaptıkları iş gereği ağ bağlantıları kolay bir şekilde kesilemez.

2.3 Saldırı Tipleri

DoS saldırıları iki ana başlık altında sınıflandırılabilir. [6]

2.3.1 Sel saldırıları: Sel saldırılarının amacı hedef alınan sunucudaki kaynakları (CPU gücü ve hafıza) ya da ağdaki kaynakları (bant genişliği ve paket tamponları) tüketmek için tasarlanmış sürekli bir trafik seli ile uzak sistemin bunaltılmasıdır. Bu saldırılar, hizmetin yavaşlamasına veya durmasına neden olur. Sel saldırları da dört ana başlık altında toplanabilir:

- **TCP SYN Sel Saldırısı:** TCP protokolündeki üç-yollu anlaşma tarzının eksikliklerinden yararlanarak, saldırgan, ulaşılamayan kaynak adresleri kullanır ve kurban sunucuya yönlendirilmiş bağlantı istekleri gönderir. Sunucunun bağlantı isteklerini tamamlayamaması sonucu kurban bütün şebeke kaynaklarını tüketir. Göreceli olarak ufak bir sahte paket seli, hafıza, işlemci ve uygulamaları kilitleyerek sunucunun

iş göremez duruma gelmesine neden olur.

- **Smurf IP Saldırısı:** Saldırgan savunmasız ve açıkları bulunan ağların genel yayın (broadcast) adreslerine sahte ICMP ECHO paketleri yollar. Bu ağlardaki bütün sistemler ICMP ECHO cevaplarıyla kurbana cevap yollar. Bu da, hızlı bir şekilde hedefin var olan bant genişliğini tüketerek, sistemin asıl kullanıcılarına hizmet vermesini etkili bir şekilde engeller.

- **UDP Sel Saldırısı:** UDP bağlantısız bir protokoldür ve veri aktarmak için herhangi bir bağlantı kurulmasına ihtiyaç duymaz. UDP Sel saldırısı, saldırganın kurban sistemdeki rastlantısal bir bağlantı kapısına bir UDP paketi göndermesiyle başlar. Kurban, paketi aldığında hedef bağlantı noktasında hangi uygulamanın beklediğini belirleyecektir. Bağlantı noktasında bir uygulamanın beklemediğini fark ettiğinde, sahte adresteki kaynak için, ulaşılamaz hedef ICMP paketi oluşturacaktır. Böylesi tasarlanmış bir yöntemle kurbanın bağlantı noktalarına yeterince UDP paketi gönderilirse, sistem büyük olasılıkla gerçek istemcilere cevap veremeyecek hale gelecektir.

- **ICMP Sel Saldırısı:** Bu saldırı tipinde temel iki yöntemden söz edilebilir; Floods ve Nukes. Flood yöntemli bir ICMP seli, çok sayıda ping ya da UDP paketi yayınlanarak gerçekleştirilir. Temel fikir, hedef sisteme çok fazla veri yollayarak, sistemi yavaşlatmak ve ping zaman aşımı yüzünden IRC bağlantısını kesmektir. Nukes ise bazı işletim sistemlerindeki hataları hedef alır. İşletim sisteminin idare edemeyeceği paketler gönderilerek, sistemin kilitlenmesine sebep olur.

2.3.2 Mantık ya da yazılım saldırıları: Bu tip saldırılar, hedef sistemde çalışan yazılım üzerindeki hataları bulup bu hataları kullanarak sistemi bloke etmek için tasarlanmış olup az sayıda kötü biçimlendirilmiş (malformed) paket kullanılarak yapılır. Bu saldırıları, açığı ortadan kaldıran yazılım

yamaları yüklenerek ya da bu paketleri hedef sisteme ulaşmadan süzecek kalkan kuralları oluşturarak kolayca engellenebilir. Bu saldırılar da dört ana başlıkta toplanabilir.

- **Ping of Death:** Saldırgan, kurbana, olası en büyük IP paket boyutundan daha büyük ICMP ECHO istek paketleri gönderir. Kurban, paketin büyüklüğünden dolayı, paketleri bir araya toplayamaz. Sonuç olarak, işletim sistemi çöker ya da yeniden başlar.

- **Teardrop:** Saldırgan, paketin konum bilgisini değiştirerek, düzgünce bir araya getirilemeyecek birden çok parça yollar ve böylece kurban sistemin kilitlenmesine ya da yeniden başlamasına sebep olur.

- **Land:** Saldırgan, kaynak ve hedef adresleri aynı olan sahte bir paket yollar. Bu saldırı sonunda kurban sistem şaşıracak ve çökecek ya da yeniden başlayacaktır.

- **Echo/Chargen:** Karakter üreteci (chargen) hizmeti, basitçe, bir karakter akımı üretmek için tasarlanmıştır. Başlangıçta test amaçlı kullanılmıştır. Davetsiz kullanıcılar sistem kaynaklarını tüketerek bu hizmeti kendi zararlı amaçları için kullanabilirler. Sistem sonsuz bir döngü içine sokularak, ağır yük altına girer ve diğer hizmetlerini yerine getiremez duruma gelir.

2.4 DDoS Araçları

Günümüzde kullanılan DDoS araçları ortaya çıkışlarına göre aşağıdaki şekilde sıralanabilir:

- *Trinoo*, *Trin00* olarakta adlandırılır, bilinen ilk DDoS aracıdır. Haziran-Temmuz 1999'da görünmeye başlamıştır. Trin00 dağıtık bir SYN DoS saldırısıdır, denetçi ve hayelet programların kullandıkları kapılar Çizelge

2.1'de gösterilmiştir.

Çizelge 2.1 DDoS araçlarının kullandıkları kapılar

DDoS Aracı	Davetsiz Kullanıcı-Denetçi İletişimi	Denetçi-Hayalet Program iletişimi	Hayalet Program-Denetçi İletişimi
Trinoo	27665/tcp	27444/udp	31335/udp
TFN	ICMP ECHO/ ECHO Cevabı	ICMP ECHO Cevabı	ICMP ECHO/ ECHO Cevabı
Stacheldraht	16660/tcp	65000/tcp	ICMP ECHO Cevabı
Trinity	6667/tcp	6667/tcp (33270/udp)	
Shaft	20432/tcp	18753/udp	20433/udp

- *Tribe Flood Network* (TFN) trinoo'dan sonra görünmeye başlamıştır. TFN istemci ve denetçi programları ICMP sel saldırısı, SYN saldırısı, UDP sel saldırısı ve SMURF tipi saldırılar gibi bir çok saldırıyı yapabilecek şekilde tasarlanmış bir DDoS ağı gerçeklerler. TFN, trinoo'dan ciddi derecede farklıdır, istemci(saldırgan), işleyici ve ajanlar arasındaki iletişim ICMP ECHO ve ECHO REPLY paketleri kullanılarak gerçekleştirilir. TFN istemcisinden hayalet programa kurulan iletişim ICMP ECHO REPLY paketleri kullanılarak sağlanır. TCP ve UDP trafiği olmaması bu paketlerin belirlenmesini zorlaştırır, çünkü çoğu ağ izleme aracı ICMP trafiğini kaydetmek ve göstermek üzere ayarlanmamıştır.

- *Stacheldraht* 1999 yazının sonlarında ortaya çıkan bir DDoS aracıdır ve TFN ve trinoo'nun özelliklerini beraber kullanır. Ayrıca, saldıgan-denetçi arasında şifrelenmiş iletişim ve otomatik ajan güncellemeleri gibi ileri özellikler barındırır. Olası saldırıları, TFN'e benzerlik gösterir.

- 2000 Ağustosunda, Apache ağ sunucularına karşı gerçekleştirilen bir saldırı ortaya çıkmıştır. Saldırı, Apache ağ sunucularına gönderilen

binlerce eğik çizgi (“/”) içeren bir URL'nin, sunucuyu çok yüksek seviyede işlemci zamanı kullanan bir duruma sokabilmesi açığını kullanılmıştır. Bu özel saldırı 500'den fazla Windows makinesinden başlatılmıştır ve tahminen 1.2.5 sürümünden eski olan Apache ağ sunucuları üzerinde etkili olmuştur.

- Takip eden ayda *Trinity* adında yeni bir DDoS aracı rapor edilmiştir. Trinity, kurban site üzerine, UDP, fragment, SYN, RST, ACK gibi çok çeşitli saldırılar yapma yeteneğine sahiptir. Ancak, işleyici ya da davetsiz kullanıcı programından ajanlarla kurulan iletişim Internet Relay Chat (IRC) ve ICQ kullanılarak gerçekleştirilir. Trinity öncelikle 6667 portunu kullanır ve 33270 TCP portunu dinleyen bir arka kapı programı vardır.

- Kasım 1999'da, *Shaft* DDoS aracı ortaya çıkmıştır. Shaft ağı, kavramsal olarak trinoo'ya benzerlik gösterir; paket seli saldırısı yapar ve istemci sel paketlerinin boyutunu ve saldırı süresini kontrol eder. Shaft'a ait ilginç bir özellik, bütün TCP paketlerinin sıra numarasının 0x28374839 olmasıdır.

- *Tribe Flood Network 2K* (TFN2K) Aralık 1999 da yayınlanmıştır. TFN2K, trafiğinin belirlenmesi ve süzülmesini zorlaştırmak, uzaktan komut çalıştırmak, IP adresi aldatması kullanarak saldırıyı yapan gerçek kaynağı saklamak, UDP, TCP, ve ICMP dahil bir çok taşıyıcı protokol üzerinden TFN2K trafiğini taşımak gibi gelişmiş özellikler içeren karmaşık bir TFN türevidir. TFN2K saldırıları, TFN'de olduğu gibi sel saldırıları içerir, Teardrop ve Land saldırılarında olduğu gibi bozuk ya da geçersiz paketler kullanarak bazı sistemlerini çökertmeye çalışır.

2.5 Savunma Mekanizmaları

En iyi savunma stratejisi, saldırının olmasını en başından engellemektir. Sistemleri, saldırganlardan korumaya çalışan birçok savunma mekanizması vardır[2]:

- **Küresel Düzenlenmiş Süzgeçler Kullanmak:** Ingress/Egress süzgeçler, kısaca, yerel şebekeye ait kaynak adresli paketlerin dışarıdan iç ağa girmesini ve ağ dışı kaynak adreslere sahip paketlerin ağ içerisinden ağ dışına çıkışını engellemektir.

- **Kullanılmayan Hizmetlerin Etkisizleştirilmesi:** Eğer hizmetler kullanılmıyorsa, saldırları engellemek için etkisiz hale getirilmelidirler.

- **Güvenlik Yamalarının Uygulanması:** Saldırı riskini en aza indirgemek için sistemler en son güvenlik yamalarıyla donatılmalıdır.

- **IP Broadcast Özelliğinin Kapatılması:** Bu özellik kapatıldığında, sistem ICMP Flood ve Smurf saldırılarında güçlendirici olarak kullanılamaz.

3. ÖZ-DÜZENLEMELİ HARİTA (SOM)

Yapay Sinir Ağları (YSA) fen bilimlerinden sosyal alanlara kadar geniş bir yelpaze içerisinde oldukça fazla uygulama alanı bulmuş ve halen günümüzde üzerinde oldukça yoğun biçimde çalışılan ve uygulamalarda sıkça kullanılan yöntemleriyle bilim insanlarının ilgi odağı halindedir. YSA üzerindeki araştırmalar 1940 yılında başlamış ve 1970'lerde yeni öğrenme algoritmaları, VLSI teknolojisi ve paralel işleme tekniklerindeki gelişmelere bağlı olarak hız kazanmıştır [7, 8]. Bu hızlı gelişim ile değişik sinir ağı modelleri, teknikleri ve algoritmaları ortaya çıkmıştır. Öğrenme kurallarına göre, bu ağlar üç ana grup altında toplanabilir.

- *Öğreticili öğrenme algoritması kullanan ağlar:* Öğreticili öğrenme algoritmalarının temel niteliği, ağ parametrelerinin düzenlenmesinin bir dış öğretici tarafından belirlenmesidir. Bu öğretici, verilen girdi sinyaline göre ağın vermesi istenilen tepki olarak düşünülebilir. Ağın gerçek tepkisi ve istenilen tepki arasındaki fark, hata sinyali olarak adlandırılır ve bu sinyale göre ağ parametreleri ayarlanır. Geri Yayılma [9], LVQ Algoritması [10], Adaline/Madaline [11], Boltzmann Makinesi (BM) [12] bu modellere örnek olarak gösterilebilir.

- *Güçlendirilmiş öğrenme algoritması kullanan ağlar*: Güçlendirilmiş öğrenme algoritmalarında, ağ ağırlıkları, gereği gibi yapılan uygulamalarda güçlendirilir, yetersiz yapılan uygulamalarda zayıflatılır. Bütün teori, girdi-çıktı haritalaması arasındaki deneme ve hata süreci boyunca, takviye sinyali olarak adlandırılan bir sayıl performans endeksini maksimuma çıkartmak üzerine dayalıdır. [13, 14]

- *Öğreticisiz öğrenme algoritması kullanan ağlar*: Kohonen Öz-Düzenlemeli Haritası'nın ana katkısı bu kategoriyle ilgilidir. Burada, ağ elemanları etkinleştirme evresi boyunca bir birbirleriyle rekabet eder. Rekabet, karşılıklı yanal etkileşimler sayesinde gerçekleşir ve öğrenme

öğreticisiz bir şekilde devam eder. Bu kategorinin önemli örnekleri olarak SOM [15], ART1 [16], ART2 [17], BAM [18], Hofield Hafızası [19] gösterilebilir.

Bu bölümde SOM felsefesi üzerinde durulacak ve ağın performansını arttırmak için kullanılan değişik teknikler gösterilecektir. Ayrıca, ağın yakınsama kıstaslarıyla ilgili değişik metotlar karşılaştırılarak, tüm öğrenme işleminin teorik geçmişi incelenecektir.

3.1 Öz-Düzenlemeli Harita (SOM)

SOM, yüksek boyutlu girdi vektörü uzayının, doğrusal olmayan bir şekilde, bir ya da iki boyutlu bir düzen üzerine iz düşümünü temel alan bir sinir ağıdır. Kohonen [15] 1981 yılında bu teorinin temel algoritmasını ortaya koymuştur. İzleyen yıllarda, algoritma olgunlaşmıştır [20-23] ve Kohonen, SOM ile ilgili materyalleri ve SOM bağlantılı teori ve uygulamaları içeren iki kitap yayınlamıştır [24, 25]. Daha sonra, SOM haritasının etkinliğini arttırmak için birçok teori ortaya atılmış ve yayınlanmıştır. [26-29]

Tüm teori, beyindeki retinotopic, somatosensory ve cortical haritalar üzerinde yapılan biyolojik araştırma sonuçlarından esinlenilerek hazırlanmıştır. Deneylerden elde edilen kanıtlara göre, beyin dokusunun bazı kısımları girdi sinyaline göre örgütlenmiştir. Beyinin değişik bölgeleri, bazı özgün görevleri yerine getirmek üzere adanmıştır ve beynin bu bölümlerinde topolojik olarak yakın olan nöronlar birbirlerine bağlıdırlar. Yerelleşme işleminin en etkileyici noktası, bu alanların (haritaların) otomatik ve uyarlanır oluşmasıdır. SOM' un, bir ya da iki boyutlu düzeni içerisindeki işlem birimlerinin (nöronların) topolojik bağlantısı, bu fikrin YSA alanına uygulanmasıdır. Topolojik bağlı nöronların girdi sinyaline göre uyarlanır ve otomatik olarak nasıl örgütlendiği sorunun çözümünde rekabetçi öğrenim teorisi ve vektör niceleme (VQ) teorisi yardımcı olmuştur. VQ teorisinin ana fikri, girdi sinyalinin, gerçek sinyal vektöründen çok daha küçük bir sayıda bir kod tablosu vektörü (codebook vector) ile en olası şekilde nasıl belirtileceğini bulmaktır. VQ algoritmasının çıktısı, girdi verisi dağılımını SOM

gibi gösteren bir eğitilmiş kod tablosu vektörüdür. Ancak SOM da, nöronların topografik yerleşimi, girdi verisi kümesi içerisinde bulunan özgün veri örüntülerinin benzerlikleri hakkında da bilgi içermektedir.

SOM algoritmasının sıfırlama evresinde, iki boyutlu nöron yapısının örgüsündeki her bir nörona atanan girdi verisiyle aynı boyuta sahip vektörler rastlantısal olarak oluşturulurlar. Girdi vektörü örgüdeki her bir öğeye tam bağlıdır. Eğitim sırasında, her bir nörona rekabetçi öğrenme düzeni uygulanır ve aktivasyon şiddetine göre, kazanan nöron en uygun öğe (BMU) olarak seçilir. Bu işlem en geniş kullanıma sahip olan VQ algoritmalarından biri olan Linde-Buzo-Gray algoritması ile benzerlikler gösterir. [30, 31]. BMU bulunduktan sonra, komşuluk fonksiyonu ile belirlenen BMU komşuluğu içerisinde kalan nöronlar güncellenir.

3.2 Terminoloji

Bu bölümde, bu tez içerisinde kullanılan temel gösterim anlatılmaktadır.

N : Nöron ve eğitim girdisi verisinin boyutu,

i : Nöron dizini, $i = 1, 2, \ldots, J$,

$\boldsymbol{M}_i(k)$: k. yinelemedeki i nöronun vektörü,

$m_{i,n}(k)$: $\boldsymbol{M}_i(k)$ nın n. koordinatı, $n = 1, 2, \ldots, N$,

$x_{M_i}(k)$: k. yinelemedeki i nöronunun topolojik x dizini,

$y_{M_i}(k)$: k. yinelemedeki i nöronunun topolojik y dizini,

X : x yönündeki nöron sayısı,

Y : y yönündeki nöron sayısı,

J : Ağdaki toplam nöron sayısı, iki boyutlu ağlar için $J = X \cdot Y$, tek boyutlu ağlar için $J = X$

k : Yineleme adımı, $k = 1, 2, \ldots, k_{max}$,

$\Lambda(k)$: k. yineleme adımında verilen eğitim verisi, $\Lambda^t = (\Lambda_1, \Lambda_2, \ldots, \Lambda_N)$ ve Λ_n veri vektörünün n. koordinattaki değeri

$M_c(k)$: En uygun öğenin (BMU) (kazanan nöron) k. adımdaki ağırlık vektörü

c : En uygun öğe dizini

$\alpha(k)$: Öğrenim oranı katsayısı

$R(k)$: k. yineleme adımındaki BMU çevresindeki etki alanı, BMU çevresindeki Gauss fonksiyonunun genişliğini hesaplamak için kullanılır.

$dd_{i,c}(k)$: BMU ve $\boldsymbol{M_i}$ arasındaki topografik uzaklık

$\beta(c, i, k)$: Komşuluk fonksiyonu

3.3 Algoritma

Eğitim adımları şöyle sıralanabilir [25]:

1. Her bir nörona rastlantısal bir vektör atanır.

2. Her bir eğitim verisi için, Euclid uzaklığı ölçüsü kullanılarak en uygun öğe (BMU) bulunur.

$$c = \arg\min_i \left[\sum_{n=1}^{N} \left(\Lambda_n(k) - m_{i,n}(k) \right)^2 \right] \tag{3.1}$$

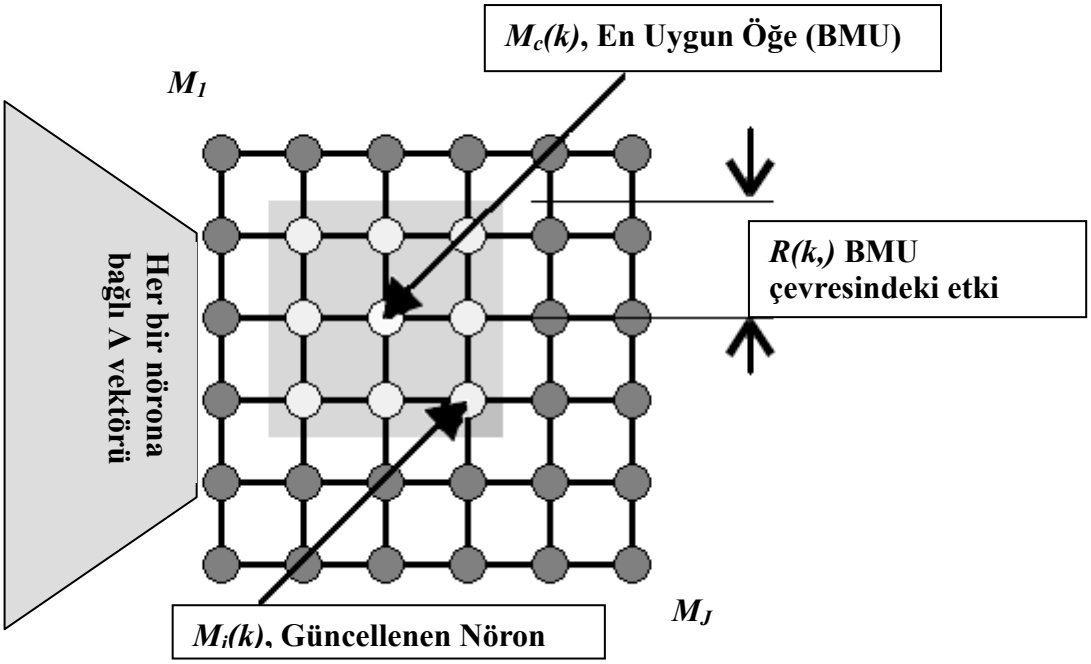
$\Lambda_n(k)$ k. yinelemedeki veri örneği vektörünün n. öğesidir ve $m_{i,n}(k)$ k zamanındaki i. nöronun n. öğesinin konumudur.

3. $\alpha(k)$ ve $\beta(c,i,k)$ kullanılarak her bir nöron güncellenir ve ağ yinelemeli olarak eğitilir, $\alpha(k)$ öğrenme katsayısını ve $\beta(c,i,k)$ komşuluk fonksiyonunu gösterir.

$$M_i(k) = M_i(k-1) + \alpha(k) \cdot \beta(c,i,k) \cdot \left(\Lambda(k) - M_i(k-1) \right) \tag{3.2}$$

Burada $0 < \alpha(k) < 1$ ve $0 < \beta(c,i,k) < 1$.

4. $k=k_{max}$ olana kadar *2* ve *3* tekrar edilir.



Şekil 3.1. Örgü şeklinde bağlanmış iki-boyutlu nöronlar ve en uygun öğe (BMU) çevresindeki topolojik komşuluğa bir örnek

3.4 Öğrenme Oranı ve Komşuluk Fonksiyonu

Bütün eğitim işleminin performansı ve ortaya çıkan topolojinin girdi verisi dağılımına göre kalitesi, öğrenme oranı ve komşuluk fonksiyonu katsayılarının seçimiyle yakından ilişkilidir. Öğrenme oranı sayısal bir değerdir ve adaptasyon kazancı olarak da düşünülebilir. SOM gibi öz-yinelemeli olasılıksal algoritmalar da, bu kazanç sistemin kararlılığını kontrol eder. [32]. Komşuluk fonksiyonu en uygun öğesi çevresindeki nöronların adaptasyon gücünü belirlemektedir. Bu fonksiyon yineleme sırasında, kazanan nöron çevresindeki esnek yüzeyin bükülmezliğini tanımlar. Öğrenme oranı ve komşuluk fonksiyonu için belirtilmiş bir tanım yoktur, fakat eğitim süresince azaltılan bir öğrenme oranı genelde tercih edilen öğrenme yöntemleri içerisinde kullanılmaktadır.

En çok kullanılan öğrenme oranları aşağıda verilmiştir:

- Eksponansiyel azalan öğrenme oranı [33] (Eksponansiyel alfa):

$$\alpha(k) = \alpha_{ilk}\left(\frac{\alpha_{son}}{\alpha_{ilk}}\right)^{\frac{k}{k_{\max}}} \tag{3.3}$$

α_{ilk} ve α_{son} değerleri deneysel olarak belirlenir.

- Tersine azalan öğrenme oranı [34] (Ters alfa):

$$\alpha(k) = \frac{C \cdot \alpha_{ilk}}{C + k} \tag{3.4}$$

burada C probleme bağlı bir sabittir ve bu değer için $C = {k_{\max}}/{100}$ makul bir seçimdir.

- Doğrusal azalan öğrenme oranı [34] (Doğrusal alfa):

$$\alpha(k) = \alpha_{ilk} \cdot \frac{k_{\max} - k}{k_{\max}} \tag{3.5}$$

- Mulier ve Cherkassy tarafından önerilen öğrenme oranı [35] (Mulier alfa):

$$\alpha(k) = \frac{1}{p \cdot (k-1) + 1} \tag{3.6}$$

Burada

$$p = \frac{1 - \frac{J}{k_{\max}}}{J - \frac{J}{k_{\max}}} \tag{3.7}$$

olarak seçilir.

Öğrenme oranı gibi, problemin doğasına bağlı olarak birçok komşuluk fonksiyonu kullanılabilir. BMU çevresinde komşuluk fonksiyonunun artan olmaması tek sınırlamadır. En yaygın komşuluk fonksiyonları şunlardır:

- Doğrusal daralan Gauss komşuluk fonksiyonu [34] (LinGauss komşuluğu):

$$\beta(c,i,k) = e^{-\left(\frac{dd_{i,c}(k)^2}{2\cdot R(k)^2}\right)} \tag{3.8}$$

Burada $$R(k) = 1 + (R_{ilk} - 1)\cdot\frac{k_{\max} - k}{k_{\max}} \tag{3.9}$$

olarak alınır,

ve iki boyutu topoloji için:

$$dd_{i,c}(k) = \sqrt{\left(x_{M_i}(k) - x_{M_c}(k)\right)^2 + \left(y_{M_i}(k) - y_{M_c}(k)\right)^2} \tag{3.10}$$

olur. R_{ilk}, *Max(X,Y)* olarak seçilir.

- Eksponansiyel daralan Gauss komşuluk fonksiyonu [36] (ExpGauss komşuluğu):

$$\beta(c,i,k) = e^{-\left(\frac{dd_{i,c}(k)^2}{2\cdot R(k)^2}\right)} \tag{3.11}$$

Burada $$R(k) = R_{ilk}\left(\frac{R_{son}}{R_{ilk}}\right)^{\frac{k}{k_{\max}}} \tag{3.12}$$

R_{ilk} = Max(X, Y) ve R_{son} = 1 olarak seçilir.

- Kabarcık komşuluk fonksiyonu [34] (Kabarcık komşuluğu) :

$$\beta(c,i,k) = \begin{cases} 1, & if \ \ |c-i| \le s(k) \\ 0, & otherwise \end{cases} \tag{3.13}$$

Burada *s(k)*, *k.* yineleme adımıyla, doğrusal veya eksponansiyel azalan bir fonksiyondur.

Genelde, *eksponansiyel alfa* ve *ters alfa* gibi hızlı azalan öğrenme oranları kullanılırken, öğrenme işleminin iki evreye bölünmesi önerilir. İlk evrede, nispeten büyük ilk alfa değerleri kullanılır, (örn. α = 0.9, α = 1). Bu evre, genel düzenlemenin olduğu zaman aralığı olarak düşünülür. Bu dönemden sonra, küçük alfa ilk değerleri (örn. α = 0.1, α = 0.01) kullanılan, nöronların ince ayarının gerçekleştiği ikinci bir evre gelir [33].Yapılan deneylerde, iki evreli bu eğitim düzeninin, eğitim performansını arttırdığı gözlenmiştir. Bu gerçek [37] de detaylı olarak çalışılmıştır. Bu çalışmanın çok şaşırtıcı olan sonucuna göre, sonuçta elde edilen topolojinin biçimlenmesinde, sadece girdi verisinin son yüzde 20'lik kısmının etkisi vardır. Bu sonuçla, ağa sunulan verinin sırasının, öz-düzenleme

evresinin performansı üzerinde büyük bir etkisi olduğunu tahmin etmek zor olmaz. (3.6) da gösterilen öğrenme oranını kullanarak, Mulier ve Chekassky, nöronların nihai topolojik yapısının biçimlenmesinde her bir girdi verisinin eşit katkısını sağlamışlardır.

3.5 Öğrenen Vektör Nicemlemesi (LVQ)

SOM öğrenme algoritması kullanılarak kod tablosu vektörleri elde edildikten sonra, sınıflandırma bölgelerinin ayırt edilebilmesi için, Öğrenen Vektör Nicemleme (Learning Vector Quantization, LVQ-3) yöntemi kullanılarak, harita alt uzaylara bölünür. LVQ algoritması, SOM haritasında elde edilen kod tablosu vektörleriyle gösterilen, olası sınıflandırma bölgelerinin merkezleri arasındaki Gauss sınırlarının düzenlenmesi üzerine kurulu bir sınıflandırma algoritmasıdır [38]. Literatürde LVQ algoritmasının birçok gerçeklemesi mevcuttur. Temelde, bu algoritmalar, öğreticili yöntemler kullanarak, kod tablosu vektörlerini, öğrenme verisi içerisindeki merkezlerini yansıtan pozisyonlara taşımaya çalışırlar. Gerçekte amaç, yanlış-sınıflandırma (miss-classifiction) oranını düşürerek, farklı sınıflara ait olan kod tablosu vektörleri arasındaki Gauss sınırlarını bulmaktır. LVQ-1 algoritmasında öğrenme sonucu elde edilen sınırlarda, sınıflandırma boyutunda sorunlar oluşmakta ancak bu yöntemin geliştirilmesiyle elde edilen, LVQ-2 ve LVQ-3 algoritmaları bu sorunların çözülmesini ve daha iyi düzenleme yapılabilmesi için elverişli yaklaşımlar sunmaktadır.

LVQ-3 algoritması,

$$\begin{aligned} M_{\mathrm{j}}(k+1) &= M_{\mathrm{j}}(k) + \mu(k)\cdot\left(\Lambda(k) - M_{\mathrm{j}}(k)\right) \\ M_{\mathrm{i}}(k+1) &= M_{\mathrm{i}}(k) - \mu(k)\cdot\left(\Lambda(k) - M_{\mathrm{i}}(k)\right) \end{aligned} \tag{3.14}$$

formülleriyle açıklanabilir. Burada, M_i ve M_j, $\Lambda(k)$ değerine en yakın olan iki kod tablosu vektörüdür, bu sayede $\Lambda(k)$ ve M_j aynı sınıf içinde yer alırken, $\Lambda(k)$ ve M_i farklı sınıflar içinde yer alırlar; dahası, $\Lambda(k)$

$$min\left(\frac{d_1}{d_2}\frac{d_2}{d_1}\right) > s \quad \text{ve}\, s = \frac{1 - pencere}{1 + pencere} \tag{3.15}$$

şeklinde tanımlanmış bir *pencerenin* bölgesi içinde kalmalıdır. Burada, d_1 ve d_2, M_i - $\Lambda(k)$ ve M_j - $\Lambda(k)$ kod tablosu vektörleri arasındaki uzaklıktır. Ayrıca bu algoritmadaki diğer bir gereksinim ise:

$$\begin{aligned} M_i(k+1) &= M_i(k) + \varepsilon(k)\cdot\mu(k)\cdot(\Lambda(k) - M_i(k)) \\ M_j(k+1) &= M_j(k) + \varepsilon(k)\cdot\mu(k)\cdot(\Lambda(k) - M_j(k)) \end{aligned} \tag{3.16}$$

eşitliklerinde ifade edilmektedir.

Bu eşitliklerde, M_i ve Mj, $\Lambda(k)$ değerine en yakın olan iki kod tablosu vektörüdür ve $\Lambda(k)$, Mj ve M_i aynı sınıf içinde yer aldığında kullanılır. $\mu(k)$ ve $\varepsilon(k)$ parametreleri algoritmadaki öğrenme oranlarıdır.

4. OMNeT++

Simülasyon, var olan ya da önerilen bir sistemin modellenerek, belirli durumlardaki davranışlarının incelenmesi yöntemidir. Simülasyon yöntemi kullanılarak sistem davranışı zamana bağlı olarak gözlemlenebilir. Ayrık olay sistemlerinde, sistemdeki durum değişiklikleri (olaylar) kesikli zaman örneklerinde oluşur ve durum değişiklikleri sıfır zamanda meydana gelir. İki ardışık olay arasında başka bir olayın gerçekleşmediği, bu iki olay arasında sistemin durumunun değişmediği varsayılır. Ayrık olay simülasyonlarında, zamanlayıcı (scheduler), çalıştırılacak olayları içeren bir veri yapısı sağlar (Future Event Set, FES) ve bunları tek, tek çalıştırır. Bu tür simülatörlerin çalışma mantığı aşağıda verilmiştir.

sıfırla – bu evre modelin inşa edilmesi ve ilk olayların FES'e eklenmesini kapsar

while (FES boş değilse ve simülasyon henüz tamamlanmamış ise)
{
    FES den ilk olayı getir
    t:= olayın zaman bilgisi
    olayı işle
    (bu işlem FES'e yeni olaylar eklenmesine ya da varolanların silinmesine sebep olabilir)
}

Simülasyonu bitir (istatistiksel sonuçları yaz, vb.)

Bilgisayar ağları alanında, ağ simülasyonu, yazılım yardımıyla bir ağın davranışını simüle etme yöntemidir. Ağ simülasyonları a*yrık olay simülasyonlarıdır*. Bu yöntem ile yeni geliştirilen protokoller test edilebilir, ağ cihazları üzerinde değişik durumlar için testler çalıştırılabilir. Ayrıca, simülasyon teknikleri kullanılarak ağ ve/veya yazılım tasarımı da yapılabilir. Bu alanda NS2[2] ve OMNeT++[3] gibi açık kaynak kodlu özgür yazılımların yanı sıra OPNET[4], Shunra[5] ve Qualnet[6] gibi ticari yazılımlarda bulunmaktadır. Bu çalışmada açık kaynak kodlu olması, gelişmiş grafik özelliklere sahip olması ve C++ dilinde programlanabilir olması sebebiyle OMNeT++ kullanılmıştır.

[2] http://nsnam.isi.edu/nsnam/index.php/Main_Page
[3] http://www.omnetpp.org/
[4] http://www.opnet.com/
[5] http://www.shunra.com/
[6] http://www.qualnet.com/

4.1 OMNeT++

OMNeT++ nesne tabanlı, modüler bir ayrık olay ağ simülatörüdür. OMNeT++ simülatörünün sıklıkla kullanıldığı alanlar aşağıda verilmiştir [39]:

- Telekomünikasyon ağlarının trafik modellemesi,
- Protokol modellemesi,
- Kuyruklama (queuing) ağlarının modellenmesi,
- Çok işlemcili sistemlerin ya da diğer dağıtık sistemlerin modellenmesi,
- Donanım mimarisini doğrulanması,
- Karmaşık yazılım sistemlerinin performansının hesaplanması,
- Ayrık olay yaklaşımının uygun olduğu diğer sistemlerin modellenmesi.

Bir OMNeT++ modülü hiyerarşik düzende iç içe geçmiş modüllerden oluşur. Modül gömmenin derinliği sınırlı değildir, böylece gerçek sistemin mantıksal yapısı modül yapısına doğrudan uydurulabilir. Modüller mesaj aktarımı yöntemiyle haberleşir. Mesajlar isteğe bağlı olarak karmaşık veri yapıları içerebilir. Modüller mesajları doğrudan hedeflerine gönderebilecekleri gibi, kapılar ve bağlantılar kullanarak daha önceden tanımlanmış bir yolu da kullanabilirler.

Modüllerin kendi parametreleri olabilir. Parametreler, modülün davranışını kişiselleştirmek için kullanılabileceği gibi modelin topolojisini ifade etmek için de kullanılabilir. Modül hiyerarşisinde an alt seviyede bulunan modül, davranışı belirler. Bu modüller basit modül olarak adlandırılır ve simülasyon kütüphanesi kullanılarak C++ dilinde programlanır.

OMNeT++ simülasyonları hata ayıklama ve toplu işlemler gibi değişik amaçlar için çeşitli kullanıcı arabirimleri sağlar. Gelişmiş kullanıcı arabirimleri, modelin içinin kullanıcı tarafından görülebilmesini sağlar ve simülasyonun çalışması sırasında kullanıcıya müdahale etme ve model içindeki değişkenleri/nesneleri değiştirme yeteneği verir.

Simülasyon motoru ve kullanıcı arabirimleri ile simülasyon araçları, Windows ve birçok Unix sistemleri üzerinde C++ derleyicileri kullanılarak

çalıştırılabilirler.

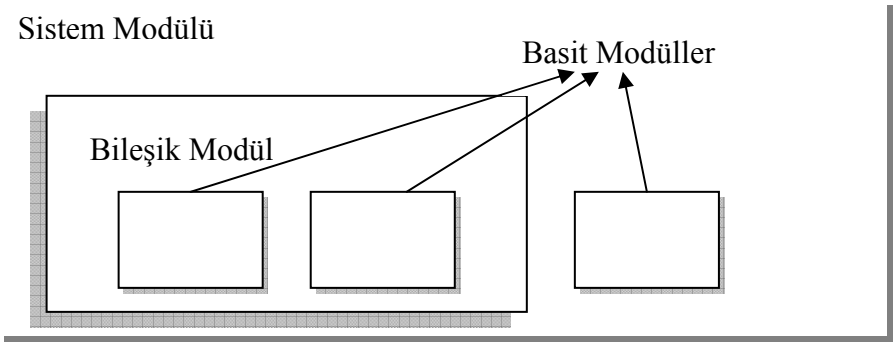
4.2 Modelleme Kavramları

OMNeT++ gerçek sistemin yapısının tanımlanması için etkili araçlar sağlar. Bu araçların bazıları aşağıda listelenmiştir:

- Hiyerarşik olarak iç içe geçmiş modüller,
- Modül tiplerinden örneklenen modüller,
- Kanallar yoluyla mesajlaşarak haberleşen modüller,
- Esnek modül parametreleri,
- Topoloji tanımlama dili.

4.2.1 Hiyerarşik modüller

Bir OMNeT++ modülü hiyerarşik olarak iç içe geçmiş modüllerden oluşur. Şekil 4.1 bir OMNeT sistem modülünü göstermektedir. Bu modüller birbirlerine mesaj göndererek haberleşirler. OMNeT++ modüllerinden çoğunlukla *ağ* olarak söz edilir. En üst seviyedeki modül, *sistem modülü* (system module) olarak tanımlanmaktadır. Sistem modülü *alt modüller* (submodules) içerir ve bu alt modüller de kendi alt modüllerini içerebilir. Bu düzenin derinliği önceden bahsedildiği gibi sınırsızdır. Model yapısı OMNeT++'nın NED dili kullanılarak tanımlanır.



Şekil 4.1. Sistem modülü birçok alt modülden oluşur.

Alt modüller içeren modüller *bileşik modüller* (compound modules) olarak adlandırılır. Modül hiyerarşisinde en alt seviyede bulunan modüller ise *basit modüller* (simple modules) olarak tanımlandırılmaktadır ve model içindeki algoritmaları içermektedirler. Kullanıcı, bu modülleri, OMNeT++ simülasyon sınıfı kütüphanesini kullanarak C++ programlama dilinde gerçekler.

4.2.2 Modül tipleri

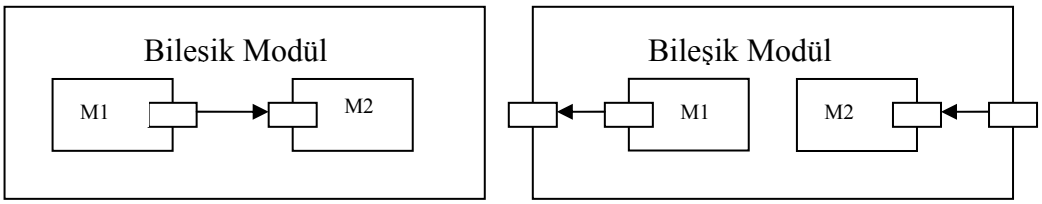
Basit ve bileşik modüllerin her ikisi de *modül tipi* (module type) örnekleridir. Model tanımlanırken kullanıcı, modül tiplerini tanımlar; bu modüllerin örnekleri daha karmaşık modül tipleri için bileşen olarak kullanılır. Son olarak, kullanıcı daha önce tanımlanmış bir modül tipi kullanarak sistem modülünü oluşturur.

4.2.3 Mesajlar, kapılar, bağlantılar

Modüller *mesaj* değişerek haberleşirler. Gerçek bir simülasyonda, mesajlar bir bilgisayar ağındaki çerçeve ve paketleri, kuyruklama ağında iş ya da müşterileri veya diğer mobil bileşenleri temsil eder. Mesajlar isteğe bağlı olarak karmaşık veri yapıları içerebilirler.

"Yerel simülasyon zamanı", modül bir mesaj aldığı zaman ilerler. Mesaj başka bir modülden gelebileceği gibi aynı modülden de gelmiş olabilir (bu tip mesajlar zamanlayıcı gerçeklenmesi için kullanılırlar). *Kapılar* modülün girdi ve çıktı arabirimleridir; mesajlar kapılardan gönderilir ve kapılardan alınır.

Tüm *bağlantılar*, modül hiyerarşisi içinde tek bir seviyede oluşturulur; bileşik bir modül içerisinde, iki alt modülün karşılıklı kapıları, ya da basit modülün bir kapısı ve bileşik modülün bir kapısı bağlanabilir (Şekil 4.2). Basit modülün kapıları, bileşik modülü içeren başka bir bileşik modülün kapılarına doğrudan bağlanamaz.



Şekil 4.2. Bileşik modüllerde bağlantı örneği

Modelin hiyerarşik yapısından dolayı, mesajlar genellikle bir dizi bağlantı üzerinden yolculuk yaparlar. Basit bir modülden başka bir basit modüle giden bu şekildeki bağlantılara *rota* (route) adı verilmektedir.

4.2.4 Paket iletimlerinin modellenmesi

Bağlantılara üç parametre atanabilir. Bu parametreler, y*ayılım gecikmesi* (propagation delay), *bit hata oranı* (bit error rate), *veri hızı* (data rate), haberleşme ağlarının modellenmesi kolaylaştırabileceği gibi, diğer modellerde de yardımcı olurlar. Her bir bağlantı için ayrı ayrı parametre tanımlanabileceği gibi, değişik parametrelere sahip bağlantı tipleri tanımlanarak bütün modelde bu tipler de kullanılabilmektedir.

Yayılım gecikmesi, mesajın kanal üzerinde yolculuk yaparken, iletiminin geciktirileceği süredir. Bit hata oranı, bir bit'in yanlış iletilebilmesi olasılığını tanımlar, böylece basit gürültülü kanal modellemesi yapılabilir. Veri hızı bit/saniye olarak tanımlanır ve paketin iletim zamanının hesaplanmasında kullanılır.

4.2.5 Parametreler

Modüllerin parametreleri olabilir. Bu parametreler ilk değerlerini NED dosyalarından ya da ana konfigürasyon dosyası olan *omnetpp.ini* dosyasından okurlar. Parametreler, karakter dizisi değerleri, sayısal ve mantıksal değerler alabileceği gibi XML veri ağaçlarından da değer alabilirler. Parametreler basit modüllerin davranışlarını kontrol etmek için ve model topolojisini daha dinamik tanımlamak için kullanılabilirler. Sayısal değer alan parametreler kullanılarak,

topolojilere esneklik katmak olanak dahilindedir. Birleşik bir modül içerisinde, parametreler kullanılarak alt modüllerin sayısı, kapıların sayısı ve iç bağlantıların nasıl yapılacağı tanımlanabilir.

4.2.6 Topoloji tanımlama yöntemi

Modelin topolojisi NED dili kullanılarak belirlenir ve bu dil, ağın modüler olarak tanımlanmasına yardımcı olur. Bir ağ tanımı, birden çok bileşen tanımından (kanallar, basit/bileşik modül tipleri) oluşur. Daha sonra bu tanımlar başka bir ağ tanımında tekrar kullanılabilir.

4.2.7 Algoritmaların programlanması

Bir model oluşturan modüller birlikteliği, C++ dilinde yazılabilen fonksiyonlardan oluşmaktadır ve bu modüllerde C++ dilinin hemen tüm özelliklerini kullanmak olasıdır. Simülasyon nesneleri (mesajlar, modüller, kuyruklar, vb.), C++ sınıfları kullanılarak programlanır.

Aşağıdaki sınıflar simülasyon kütüphanesinin bir parçasıdır:

- Modüller, kapılar, bağlantılar, vb.,
- Parametreler,
- Mesajlar,
- Kapsayıcı sınıflar (örneğin, kuyruk, dizi),
- Veri toplama sınıfları.

4.3 INET

INET kütüphanesi OMNeT++ üzerine kurulan, kablolu ve kablosuz ağlarda kullanılan cihaz ve protokollerin modellenmiş olduğu, açık kaynak kodlu bir yazılımdır.

4.3.1 Modüller ve protokoller

INET içerisinde, protokoller, basit modüller olarak gerçeklenmişlerdir. Basit bir modülün dış arabirimleri (kapılar ve parametreler) NED dosyasında tanımlanır, gerçeklenmesi ise aynı isimli bir C++ sınıfında bulunur. Bu modüller, NED dili kullanılarak, ana makine ve ağ cihazları oluşturmak üzere özgürce birleştirilebilir. Bu işlem sırasında C++ kodu yazmaya ya da yeniden derlemeye gerek yoktur. Kütüphane içinde, ana makine, yönlendirici, anahtar (switch), erişim noktası gibi kullanıma hazır birçok model bulunur ve bu modeller isteğe göre değiştirilebilir.

Ağ arabirimleri (Ethernet, 802.11, vb.) genellikle bileşik modüllerdir ve basit modüller kullanılarak hazırlanmışlardır. Bu modüller bir kuyruk (queue), bir MAC, ve diğer basit modüllerden oluşur.

INET'te simülasyon tasarımında kullanılan tüm modellerin sadece protokol gerçekleştirmesi beklenmez. Aksi takdirde simülasyon dinamiğini oluşturmak neredeyse olanaksızlaşır. Bu yüzden INET yapısında haberleşmelere yardımcı olan, ağ içerisinde oto konfigürasyon gerçekleştiren veya hareketli uçların hareketini (devinimini) simüle eden modüller de tanımlanmıştır.

Protokol başlıkları ve paket biçimleri mesaj tanımı dosyalarında (msg dosyaları) tarif edilir. Bu dosyalar OMNET++'ın *opp_msgc* aracı kullanılarak C++ sınıflarına dönüştürülür. Oluşturulan tüm sınıflar OMNET++'ın *cMessage* sınıfından türetilir.

4.3.2 Ana makine ve yönlendiricilerde kullanılan ortak modüller

Bütün PC, yönlendirici ve cihaz modellerinde kullanılan bazı ortak modüller vardır.

- *InterfaceTable*. Bu modül ana makinedeki ağ arabirimlerinin tablosunu tutar, kendisi mesaj gönderip almaz. Bu modüle, diğer modüller, genel C++ üye fonksiyonları kullanarak ulaşılır. Ağ arabirimleri, arabirimi gerçekleyen modül tarafından dinamik olarak kaydedilerek tabloya eklenir.

- *RoutingTable*. Bu modül IP (v4) yönlendirme tablosunu içerir ve çalışmak için InterfaceTable'a ihtiyaç duyar. Bu modüle de, Interfacetable gibi diğer modüller tarafından C++ sınıfının genel üye fonksiyonları kullanılarak erişilir. Rota sorgulamak, eklemek, çıkarmak ve verilen bir IP adresi için en uygun rotayı bulmak için fonksiyonları mevcuttur.

- *NotificationBoard*. Bu modül sayesinde birçok modül yayınla-abone ol tarzında haberleşir. Örneğin, radyo modülü, radyo kanalının durumu değiştiği zaman, "radyo durumu değişti" uyarısını yayınlar, bu uyarı, daha önceden bu uyarı kategorisine abone olan modüllere iletilir. Uyarı mekanizması da C++ fonksiyon çağrıları ile çalışır, mesaj gönderme işlemi yapılmaz.

4.3.3 Ağ katmanındaki ortak modüller

Global ağ seviyesinde bazı modüllerin sadece bir örneği bulunur:

- *FlatNetworkConfigurator* ana makinelere ve yönlendiricilere IP adresi atar ve sabit rotaları hazırlar.
- *ScenarioManager* simülasyonu betikler (scripts) ile çalışabilir hale getirir.
- *ChannelControl* kablosuz ağ simülasyonları için gereklidir. Uçların, diğer uçların girişim mesafesinde olup olmadığını kontrol eder.

4.3.4 Protokol katmanları arasındaki haberleşme

INET kütüphanesinde, üst katmandaki bir protokol alt katmandaki bir protokole veri göndermek istediğinde, üst katmandaki modül alt katmandaki modüle sadece paketi temsil eden bir mesaj nesnesi gönderir, alt modül bu mesaj nesnesini sarmalar ve gönderir. Alt katmandaki bir protokole paket ulaştığında bu işlemin tersi gerçekleştirilir.

Genellikle paket içerisinde ilave bilgi de taşımak gerekir. Örneğin, uygulama-katmanındaki bir modül TCP üzerinden veri göndermek istediğinde,

TCP için bir bağlantı tanıtıcısı tanımlamak gerekir. TCP, IP üzerinden bir kesim gönderdiğinde, IP'nin hedef adrese ve TTL gibi başka parametrelere ihtiyacı olacaktır. IP, Ethernet arabirimine iletim için bir datagram gönderdiğinde, hedef MAC adresi belirtilmelidir. Bu ilave bilgi, mesaja, *kontrol bilgisi* olarak eklenir.

Kontrol bilgisi, bir sonraki protokol katmanı için yardımcı bilgi barındırır ve bu bilginin ağ üzerinden diğer makine ve yönlendiricilere gönderilmesi gerekmez.

5. YÖNTEMİN SINANMASI

Yapılan bu çalışmanın hem ağ simülasyonu ortamının oluşturulması, hem de bu tasarlanan ağ yapısı üzerinde günümüz ağ sistemleri üzerinde oldukça zararlı etkileri bulunan DDoS saldırılarının yaratılması ve bu saldırı sonucu oluşan ağ trafiğinin izlenmesi ve zararlı trafiğin ayırt edilmesi için bir yapay sinir ağı modeli olan SOM haritasının kullanılması göz önünde bulundurulduğunda iki yönü bulunmaktadır. Bu DDoS saldırısı, günümüzde oldukça geçerliliği olan OMNeT++ ve INET programları kullanılarak simüle edilmiş, simülasyon boyunca kurban sisteme gelen paketler kaydedilmiştir. Daha sonra bu paketler yukarıda bahsedildiği gibi SOM kullanılarak sınıflandırılmaya çalışılmıştır. Bu çalışmanın sonucunda saldırı trafiğinin normal trafikten ayrıştırılabildiği görülmüştür.

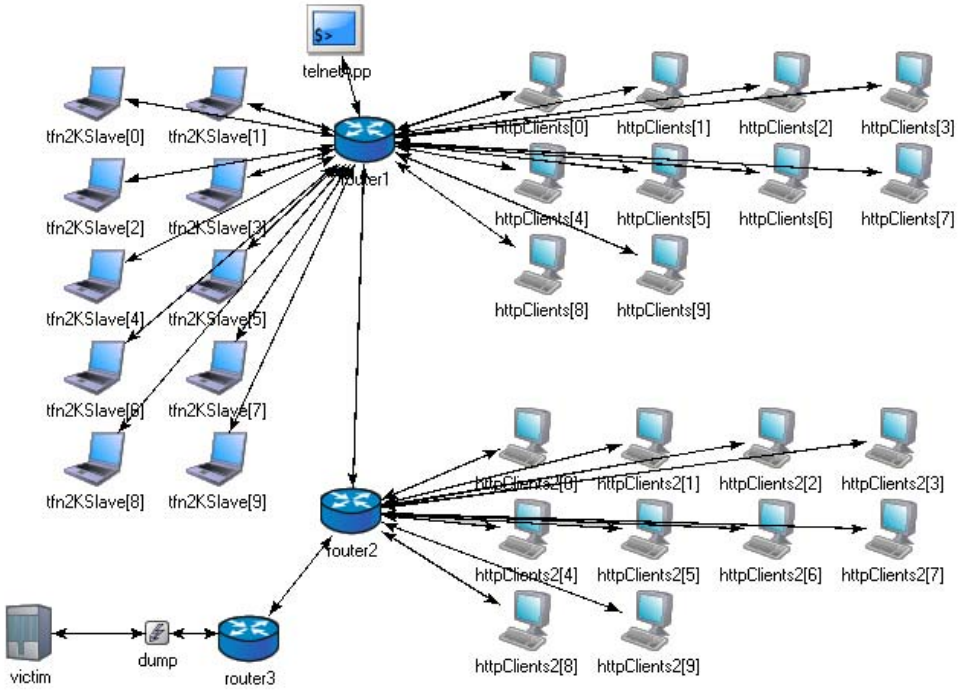
Çalışmanın ana noktalarından birisini oluşturan DDoS aracı olarak TFN2K seçilmiştir. TFN2K bir yandan yayınlanmış en son, öte yandan gelişmiş özellikleri nedeniyle izlenmesi ve engellenmesi en zor DDoS aracıdır. Bu DDoS saldırısı aracında saldırı sırasında oluşturulan IP paketlerinin hedef adres dışındaki bütün parametreleri rastlantısal yöntemler kullanılarak oluşturulur ve paketin taşıdığı bilgi herhangi bir anlamı olmamasına karşın şifrelenir. TFN2K, ICMP, UDP ve TCP protokolünü kullanarak saldırılar gerçekleştirebilir.

Simülasyon ortamı hazırlanırken, INET içerisine TFN2K aracının TCP saldırısını simüle eden TFN2KTCP[7] modülü eklenmiştir. Tasarlanan ağda, normal trafik, kurban makine üzerinde çalıştırılan *http* sunucusu ve bu sunucudan servis alan basit *tcp* istemcileri tarafından sağlanmıştır. Ayrıca kurban makine üzerinde bir *telnet* sunucusu çalıştırılmış ve bazı istemcilerin bu sunucuyla konuşması sağlanmıştır (Şekil 5.1). Kurban makineye, *TFN2K* konuşlandırılmış makinelerden periyodik olarak saldırı paketleri yollanmış ve oluşan bütün trafik kaydedilmiştir. Saldırı simülasyonu değişik sayılarda saldırı programı, tcp istemcileri ve saldırı süreleri kullanılarak birçok kez tekrarlanmıştır.

Kaydedilen trafik içerisinden, kurbana gönderilen paketler süzülmüş ve bu paketlerin kaynak kapı, hedef kapı, yük uzunluğu ve pencere boyutu özelliklerinden oluşan bir veri kümesi oluşturulmuştur. Normal TCP trafiğinde,

[7] Bakınız EK-2

istemciden başlatılan iletişimler, sunucuda daha önceden belirlenmiş bir kapıya yönlenir ve sunucu bu kapıdan gelen istekleri, istemciyle anlaşarak değişik kapılara yönlendirir. DDoS saldırılarında ise bu kapılar rastlantısal yöntemler kullanılarak belirlenir. Dolayısıyla bu iki farklı durumun değişik örüntüler oluşturması beklenmektedir. Aynı şekilde yük uzunluğu ve pencere boyutu özellikleri de, iletişim kurulduktan sonra ağ durumuna ve toplam yük boyutuna bağlı olarak değişiyor olsa da, bir örüntü oluşturabilirler.



Şekil 5.1. Deneylerde kullanılan ağ topolojisi. (Şeklin sadeliği için http istemcileri ve saldırı istemcilerinin sayısı gerçekte olduğundan az gösterilmektedir.)

5.1 Deneyler

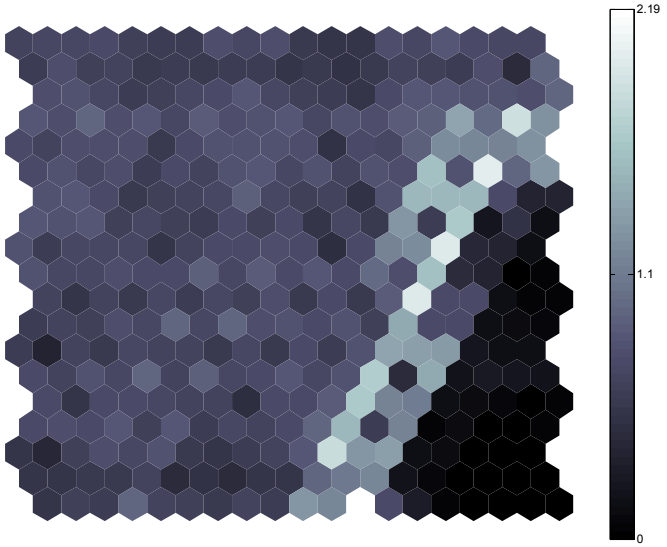
DDoS saldırılarının doğru biçimde sınıflandırılıp sınıflandırılamadığının analizinin yapılabilmesi için simülasyon ortamında çok sayıda deneyler yapılmıştır. Bu deneyler sonucunda sınıflandırma sınırlarının netleştirilmesi için de LVQ3 yöntemi kullanılmıştır. Bu deneyler aşağıda sırasıyla açıklanmıştır:

Deney 1: Bu deneyde TFN2K saldırısı aşağıdaki parametreler kullanılarak planlanmış ve saldırının simülasyonu yapılmıştır.

Ajan sayısı	:	20
Yönlendirici 1'e bağlı http istemcisi sayısı	:	100
Yönlendirici 2'ye bağlı http istemcisi sayısı	:	100
Saldırısız trafik süresi	:	50 saniye
Saldırılı trafik süresi	:	5 saniye

Bu veri kümesi öncelikle SOM analizinde kullanılmak üzere varyansları göz önünde alınarak normalize edilmiştir. Bu normalizasyon sonucunda elde edilen veri kümesiyle 10 x 10 boyutunda bir SOM ağı eğitilmiş ve bu eğitimin sonucunda oluşan ağın olası sınırlarının belirlenmesi için ayrıca LVQ3 algoritmasından yararlanılmıştır. Elde edilen ağın U-matris grafiği Şekil 5.2'de görülmektedir. SOM haritasının temel özelliklerinden belki en önemlisi sayılabilecek çok boyutlu vektörel yapının iki boyuttaki iz düşümünün elde edilmesinin sonucunda da bu boyut indiriminin görsel analizinin yapılması için geliştirilmiş U-matris, literatürde oldukça yaygın kullanılmaktadır [40].

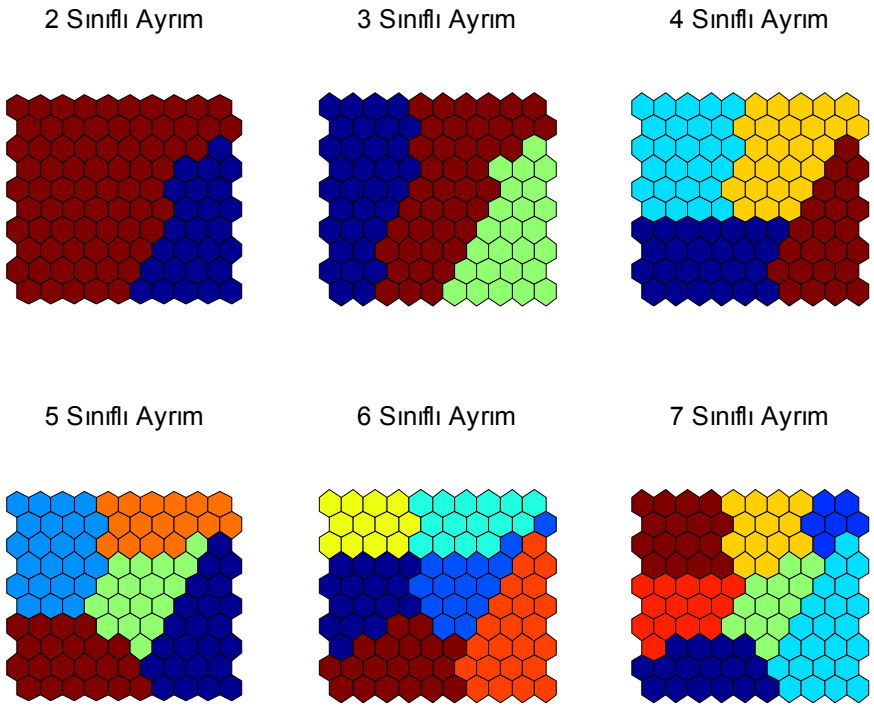
U-matris, SOM haritasını oluşturan iki boyutta birbirlerine bağlanmış çok boyutlu vektörler arasındaki uzaklığı tanımlamaktadır. Bu uzaklık tanımlaması için farklı metrikler kullanılabilmektedir. Bu çalışmada uzaklık ölçümü için Euclid metriği kullanılmıştır. Açık renk, komşu birimler arası vektörel uzaklıklarda benzerliği, tam tersi koyu renk ise farklılıkları göstermektedir. Ancak bu renklendirmede de farklılıklar olabilir. Bu farklılıkları niteleyebilmek için genelde U-matris grafiğiyle birlikte renklerin ne kadar farklılık gösterebileceğini belirtecek bir yardımcı çubuk da grafiğe ilave edilmektedir. U-matris grafiğinde ilgi çekici bir nokta, SOM harıtasını oluşturan birimlerden daha fazla altıgen formun görünmesidir. Bunun nedeni, haritada görülen her bir altıgen birimin SOM haritasını oluşturan kod vektörü arasındaki uzaklığı göstermesinden kaynaklanmaktadır.



Şekil 5.2. 1.Deneyde elde edilen U-matris grafiği

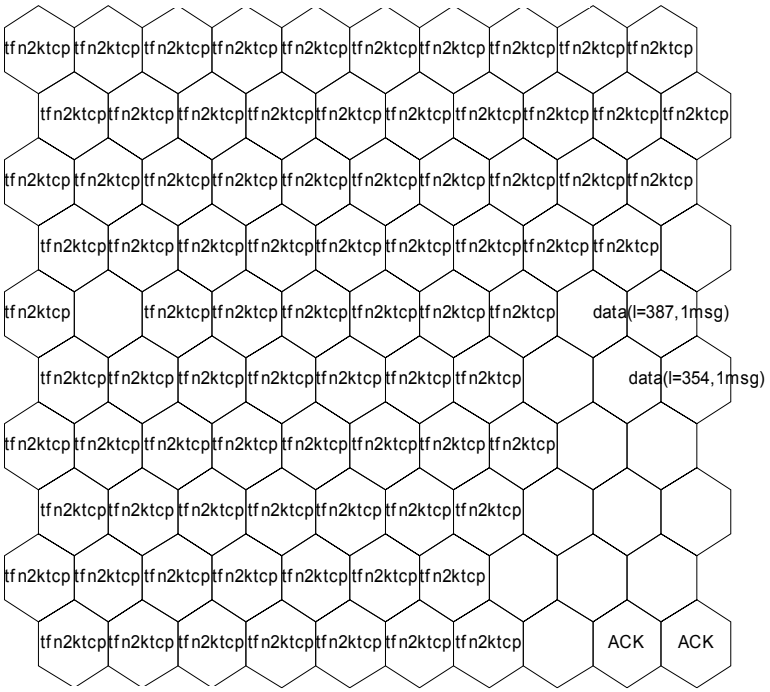
Yukarıdaki şekilde iki adet sınıf kolaylıkla fark edilmektedir. Şeklin alt sağ tarafında birbirlerine oldukça yakın kod vektörlerinin oluşturduğu koyu bölge, çaprazlamasına birbirlerinden uzak kod vektörleriyle ayrılmıştır. Şeklin diğer koyu bölgeleri ise diğer bir sınıfı oluşturmaktadır. Bu iki sınıfın varlığını kanıtlamak için bu harita üzerinde sırasıyla 2, 3, 4, 5, 6, 7 ayrı sınıf oluşturabilecek şekilde k-means algoritması çalıştırılmış ve Şekil 5.3'te bu algoritma sonuçları verilmiştir.

Bu sınıflandırma sonuçlarına bakılarak başta ayrımına varılan iki sınıflı yapıyı oluşturan bölgelerden sağ alt bölgenin değişmeyen bir sınıf olduğu görünmektedir. Bu bölgelere karşılık gelen deney verilerine bakıldığında bu bölgenin *http* trafik verilerine karşılık geldiği, diğer tüm bölgelerin ise saldırı trafiğine karşılık gelen kod vektörleri olduğu görünmektedir. Şekil 5.4'te bu ayrım çok net olarak görünmektedir. Şekilde "*tfn2ktcp*" olarak adlandırılan bölgeler saldırı verilerine karşı gelen kod vektörlerini bunun dışındaki alan ise normal trafik ile oluşan verilere karşı gelen kod vektörlerini göstermektedir.



Şekil 5.3. 1.Deney sonucu elde edilen farklı k-means sınıflandırma sonuçları

Açıklanan bu yöntemle elde edilen ağ haritası bundan sonra olası trafiğin nitelendirilmesi için kullanılabilecektir.

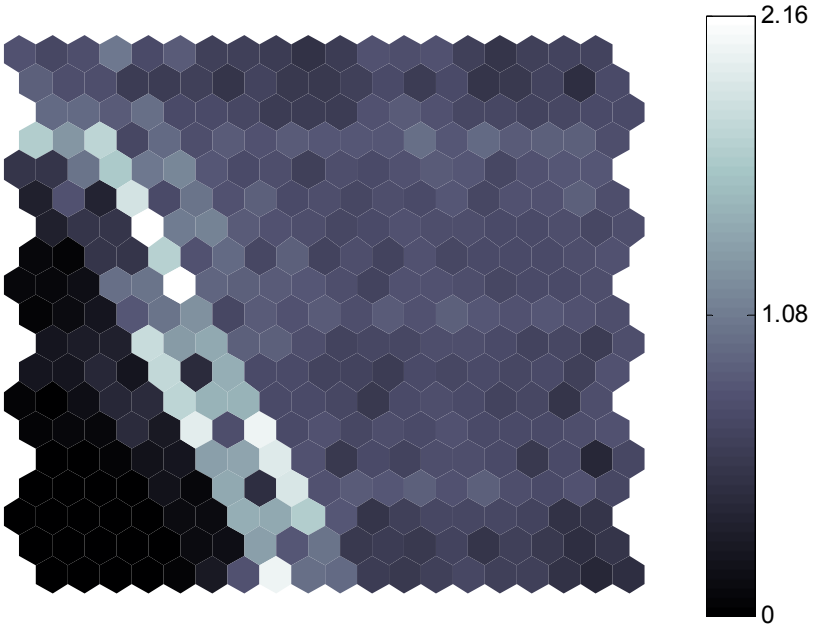


Şekil 5.4. 1.Deneyde trafik verilerine karşılık gelen kod vektörleri

Deney 2: Bu deneyde TFN2K saldırısı aşağıdaki parametreler kullanılarak planlanmış ve saldırının simülasyonu yapılmıştır.

Ajan sayısı	:	20
Yönlendirici 1'e bağlı http istemcisi sayısı	:	200
Yönlendirici 2'ye bağlı http istemcisi sayısı	:	200
Saldırısız trafik süresi	:	50 saniye
Saldırılı trafik süresi	:	10 saniye

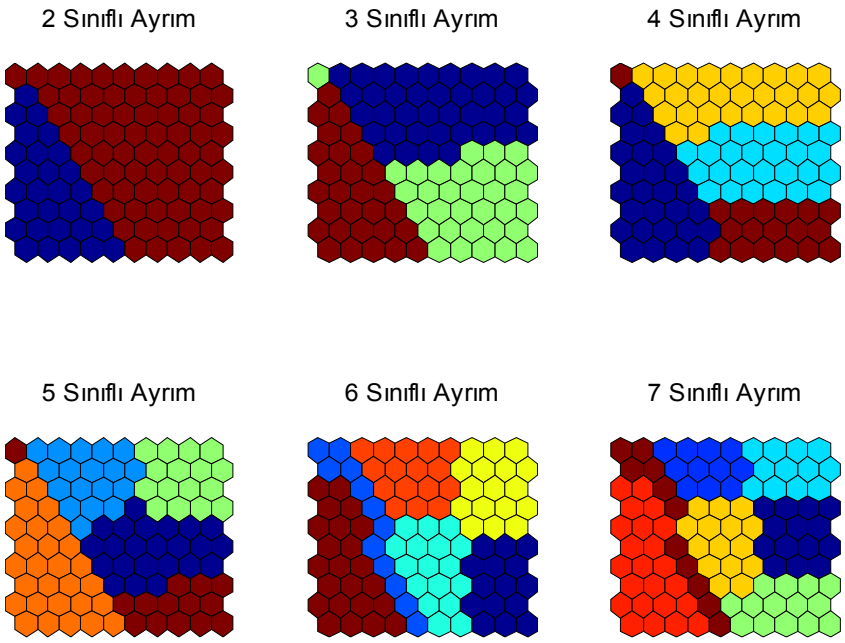
Veri kümesi normalize edildikten sonra, bu küme kullanılarak 10 x 10 boyutunda bir SOM ağı eğitilmiş ve bu eğitimin sonucunda oluşan ağın olası sınırlarının belirlenmesi için LVQ3 algoritmasından yararlanılmıştır. Elde edilen ağın U-matris grafiği Şekil 5.5'de görülmektedir.



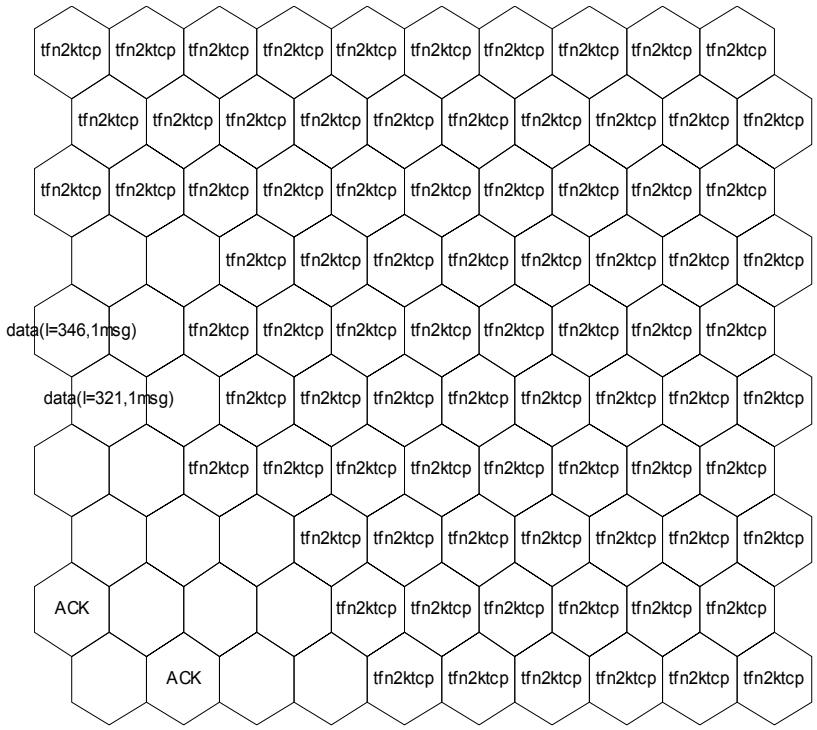
Şekil 5.5. 2. Deneyde elde edilen U-matris grafiği

Yukarıdaki şekilde iki adet sınıf kolaylıkla fark edilmektedir. Şeklin alt sol tarafında birbirlerine oldukça yakın kod vektörlerinin oluşturduğu koyu bölge, çaprazlamasına birbirlerinden uzak kod vektörleriyle ayrılmıştır. Şeklin diğer koyu bölgeleri ise diğer bir sınıfı oluşturmaktadır. Bu iki sınıfın varlığını kanıtlamak için bu harita üzerinde sırasıyla 2, 3, 4, 5, 6, 7 ayrı sınıf oluşturabilecek şekilde k-means algoritması çalıştırılmış ve Şekil 5.6'da bu algoritma sonuçları verilmiştir.

Bu sınıflandırma sonuçlarına bakılarak başta ayrımına varılan iki sınıflı yapıyı oluşturan bölgelerden sol alt bölgenin değişmeyen bir sınıf olduğu görünmektedir. Bu bölgelere karşılık gelen deney verilerine bakıldığında bu bölgenin *http* trafik verilerine karşılık geldiği, diğer tüm bölgelerin ise saldırı trafiğine karşılık gelen kod vektörleri olduğu görünmektedir. Şekil 5.4'te bu ayrım çok net olarak görünmektedir. Şekilde "*tfn2ktcp*" olarak adlandırılan bölgeler saldırı verilerine karşı gelen kod vektörlerini bunun dışındaki alan ise normal trafik ile oluşan verilere karşı gelen kod vektörlerini göstermektedir.



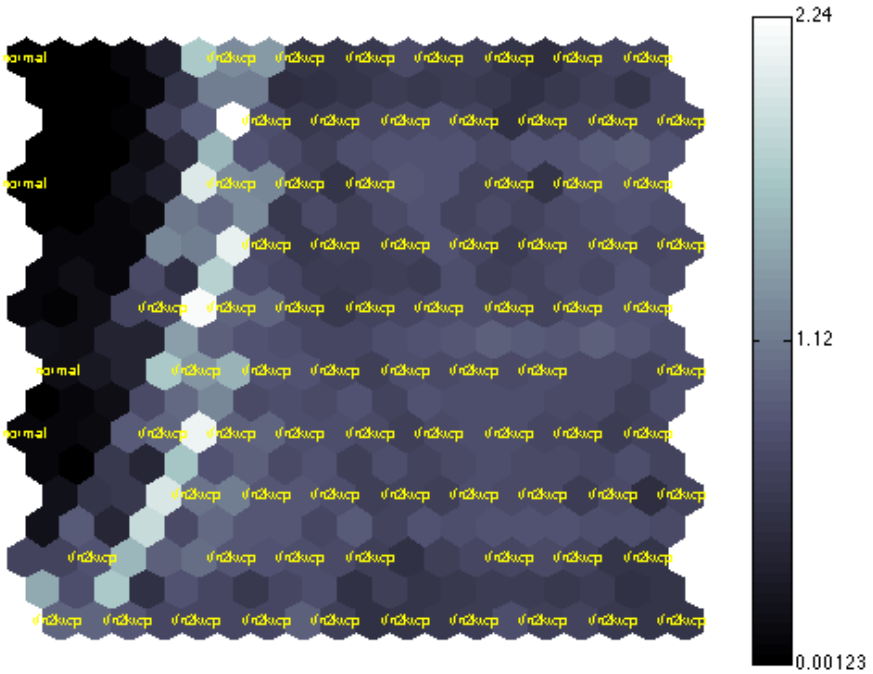
Şekil 5.6. 2.Deney sonucu elde edilen farklı k-means sınıflandırma sonuçları



Şekil 5.7. 2.Deneyde trafik verilerine karşılık gelen kod vektörleri

5.2 Sınıflandırma Sonucu

Sınıflandırmanın doğru yapılıp yapılmadığının nicel ölçümlemesinin yapılabilmesi için 2. Deney seti iki parçaya bölünmüştür. Bu bölümlemede %80 veri eğitim için kullanılmış, geri kalan %20'si ise elde edilen haritanın test edilmesi için kullanılmıştır. Bu test seti eğitimden sonra oluşan haritada en yakın kod vektörlere yönlendirildiğinde test verilerinin %100 doğru kod vektörlerle örtüştüğü görülmektedir. Bu örtüşme aşağıda U-matris halinde verilen haritanın üzerinde verilerin yerleştirilmesiyle görülebilmektedir.



Şekil 5.8. Test verilerinin U-matris üzerine yerleştirilmesi

6. SONUÇ

Bu çalışmada, bir Yapay Sinir Ağı modeli olan Öz-Düzenlemeli Harita yöntemi deneyler sonucunda elde edilen özellik vektörlerinin sınıflandırılmasında kullanılmış ve bu sınıflandırmanın doğruluğunu kontrol etmek için yapılan nicel ölçümlemede test verilerinin %100 doğru sonuç verdiği görülmüştür.

Deney verilerini oluşturmak için yapılan simülasyonlarda, DDoS saldırısı başladığında, sunucuya ulaşmaya çalışan asıl kullanıcıların paketlerinin büyük çoğunluğunun hedefine ulaşamadığı gözlemlenmiştir. Ayrıca sunucunun bulunduğu ağda, çok fazla sayıda TCP-RST ve ICMP ERROR paketi oluşmaktadır. Sunucuya gelen TCP paketleri daha önceden başlatılan bir konuşmaya ait olmadığı için sunucu, orijinal paketteki kaynak adresini kullanarak TCP-RST paketleri hazırlamakta, bu paketler yönlendiriciye ulaştığında, yönlendirici saldırının doğası gereği rastlantısal yöntemlerle seçilen bu adreslere giden bir yol bulamadığı için sunucuya ICMP ERROR cevabı döndürmektedir. Bu durum, DDoS saldırısına uğrayan makinelerin bulundukları ağlarda gözlemlenen bir anormalliktir.

Bu anormalliği denetleyen bir mekanizma yardımıyla, bu çalışmada ortaya çıkan SOM haritası kullanılarak bir korunma yöntemi geliştirilebilir. Ağda anormallik görüldüğünde, ağa gelen paketler SOM haritası kullanan süzgeçten geçirilerek yönlendirilir ise saldırıya uğrayan makine saldırı trafiğinden bağımsız çalışmasına devam edebilir.

Bu çalışmada, en etkili DDoS araçlarından TFN2K yazılımının en çok kullanılan saldırı türü olan TCP saldırısı incelenmiş ve bu saldırıya göre veri kümeleri oluşturulmuştur. İleride ICMP ve UDP saldırıları içinde benzer çalışmalar yaparak ortak bir süzgeç hazırlamak mümkündür.

KAYNAKLAR

[1] Kumar, P.G., Devaraj, D., "*Network Intrusion Detection using Hybrid Neural Networks*" IEEE- ICSCN 2007, MIT Campus, Anna University, Chennai, India. 563-569, 2007.

[2] Douligeris, C., Mitorkotsa, A., "*DDos Attacks And Defense Mechanisms: A Classification*", Signal Processing and Information Technology, 2003. ISSPIT 2003. Proceedings of the 3rd, IEEE International Symposium, 190-193, 2003.

[3] Schuba ve ark., "*Analysis of a Denial of Service Attack on TCP*", Security and Privacy, 208-223, 1997.

[4] DeLooze, L.L., "*Attack Characterization And Intrusion Detection Using An Ensemble Of Self-Organizing Maps*" Neural Networks, IJCNN '06, 2121-2128, 2006.

[5] CERT Coordination Center, Denial of Service Attacks, http://www.cert.org/tech_tips/denial_of_service.html

[6] Advanced Networking Management Lab (ANML) Distributed Denial of Service Attacks(DDoS) Resources, Types of DDoS Attacks, http://www.anml.iu.edu/ddos/types.html

[7] Freeman, J.A., Skapura, D.M., *Neural Networks*. Addison-Wesley Publishing Company, 1991.

[8] Haykin, S., *Neural Networks: A Comprehensive Foundation (2nd edition)*, Macmillan (New York), 1994.

[9] Rumelhart, D. E., Hinton, G. E., and Williams, R. J., "*Learning Internal Representations By Error Propagation*", Paralled Distributed Processing. Explorations in the Microstructure of Cognition. Volume 1: Foundations, 318-362. The MIT Press, Cambridge, MA., 1986.

[10] Kohonen, T., *Self-organization and Associative Memory (2nd edition)*, Springer-Verlag. Berlin, 1988.

[11] Widrow, B., Hoff, M.E., "*Adaptive Switching Circuits*". Institute of Radio Engineers, Western Electronic Show and Convention, Convention Record, **4**, 96-104, 1960.

[12] Hinton ve ark., "*Boltzmann Machines: Constraint Satisfaction Networks That Learn*", Technical Report No:CMU-CS-84-119, Department of Computer Science, Carneige-Mellon University,Pittsburgh,PA, 1984.

[13] Widrow, B., Gupta, N. K., Maitra, S., "*Punish/Reward: Learning With A Critic In Adaptive Threshold Systems*", IEEE Transactions on Systems, Man, and Cybernetics, **5**, 455-465, 1973.

[14] Sutton, R. S., "*Reinforcement Learning Architectures*", Proceedings ISKIT'92 International Symposium on Neural Information Processing, Fukuoka, Japan, 1992.

[15] Kohonen, T., "*Automatic Formation Of Topological Maps Of Patterns In A Self-Organizing System*", Proceedings of Second Scandinavian Conference on Image Analysis, 214, 1981.

[16] Carpenter, G.A., Grossberg, S., "*Absolutely Stable Learning Of Recognition Codes By A Self-Organizing Neural Network*", Neural Networks for Computing, American Institute of Physics, **151**, 77- 85, 1986.

[17] Carpenter, G.A., Grossberg, S., "*ART 2: Self-Organization Of Stable Category Recognition Codes For Analog Input Patterns*", Proceedings of the IEEE First International Conference on Neural Networks, II, 727-735, 1987.

[18] Kosko, B., "*Adaptive Bi-Directional Associative Memories*", IEEE Transactions on Systems Man and Cybernetics, **18(1)**, 49-60, 1988.

[19] Hopfield, J. J., Tank, D. W., "*Computing With Neural Circuits: A Model*", Science, **233,** 625 – 633, 1986.

[20] Kohonen, T., "*An Introduction To Neural Computing*", Neural Networks, **1(1)**, 3-16, 1988.

[21] Kohonen, T., "*Generalizations of the Self-Organizing Map*", In Proceedings of International Joint Conference on Neural Networks, Nagoya, Japan, 1993.

[22] Kohonen, T., "*Physiolocigal Interpretation Of The Self-Organizing Map Algorithm*", Neural Networks, **6(7)**, 895-905,1993.

[23] Kohonen, T., "*Things You Haven't Heard About The Self-Organizing Map*", In Proceedings of IEEE International Conference on Neural Networks, 1147-1156, San Francisco, California, 1993.

[24] Kohonen, T., "*The 'Neural' Phonetic Typewriter*", Computer, **21(3)**, 11-22, 1988.

[25] Kohonen, T., *Self-Organizing Maps*. Springer, Berlin, 1995.

[26] Koikkalainen, P., "*Progress With The Tree-Structured Self-Organizing Map*", 11th European Conference on Artificial Intelligence, European Committee for Artificial Intelligence (ECCAI), John Wiley & Sons, Ltd., 1994.

[27] Koikkalainen, P., "*Fast Deterministic Self-Organizing Maps*", Proceedings of ICANN'95, International Conference on Artificial Neural Networks Paris, France, **2**, 63-68, 1995.

[28] Fritzke, B., "*Growing Cell Structures -- A Self-Organizing Network In k-dimensions*", Artificial Neural Networks II, North-Holland, Amsterdam, 1051-1056, 1992.

[29] Bauer, H.U., Villmann, T., "*Growing a Hypercubical Output Space in a Self-Organizing Feature*", IEEE Transactions on Neural Networks, **8(2)**, 218-226, 1997.

[30] Linde, Y., Buzo, A., Gray, R.M., "*An Algorithm For Vector Quantizer Design*", IEEE Transactions on Communications, **28(1)**, 84-95, 1980.

[31] Buzo, A., "*Speech Coding Based Upon Vector Quantization*", IEEE Transasctions, ASSP, **28(5)**, 562-574, 1980.

[32] Ljung, L., "*Analysis Of Recursive Stochastic Algorithms*", IEEE Transactions, Auto. Cont., AC-22, 551-575, 1997.

[33] Kohonen, T., "*The Self-Organizing Map*", Proceedings of IEEE, 78:1464-1480, 1990.

[34] Kohonen, T., Hynninen, J., Kangas, J., Laaksonen, J., "*The Self-Organizing Map Program Package*", Version 3.1, 1995.

[35] Mulier, F., Cherkassky, V., "*Self-Organization As An Iterative Kernel Smoothing Process*", Neural Computation, 7(6):1165-1177, 1995.

[36] Cherkassky, V., Lari-Najafi, H., "*Constrained Topological Mapping For Nonparametric Regression Analysis*", Neural Networks, **4**, 27-40, 1991.

[37] Mulier, F., "*Statistical Analysis of Self-Organization*", Yayınlanmamış Doktora Tezi, University Of Minnesota, Electrical Engineering Dept., 1994.

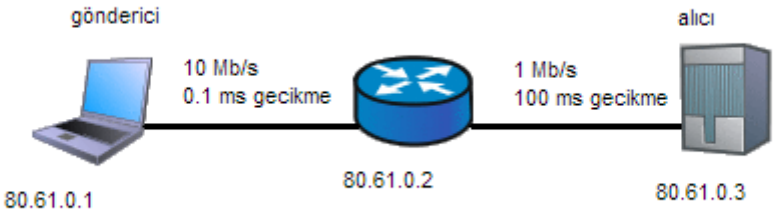
[38] Pedreira, C.E., ”*Learning Vector Quantization with Training Data Selection*”, IEEE Transactions on Pattern Analysis and Machine Intelligence, **28(1)**, 2006.

[39] Varga, A., *OMNeT++ Discrete Event Simulation System Version 3.2*, User Manual, 2005.

[40] Costa, J.A.F., de Andrade Netto, M.L., “*Cluster Analysis Using Self-Organizing Maps And Image Processing Techniques*”, Systems, Man, and Cybernatics, **5**, 1999

EK 1 Simülasyon Örneği

Bu bölümde, OMNeT++ ve INET programlarının nasıl çalıştığını gösteren basit bir örnek gösterilmektedir. Simülasyonu yapılan ağda, iki makine arasında 1 Mblık bir dosya transferi gerçekleştirilmektedir. Şekil 1'de görülen iki makine, bir yönlendirici ile birbirlerine bağlanmıştır. Gönderici makine ve yönlendirici arasındaki bağlantı hızı, 10 Mb/s ve gecikme 0.1 ms, alıcı makine ve yönlendirici arasındaki bağlantı hızı ise 1 Mb/s ve gecikme 100 ms olacak şekilde seçilmiştir.



Şekil 1 Dosya gönderici/alıcı ağı topolojisi

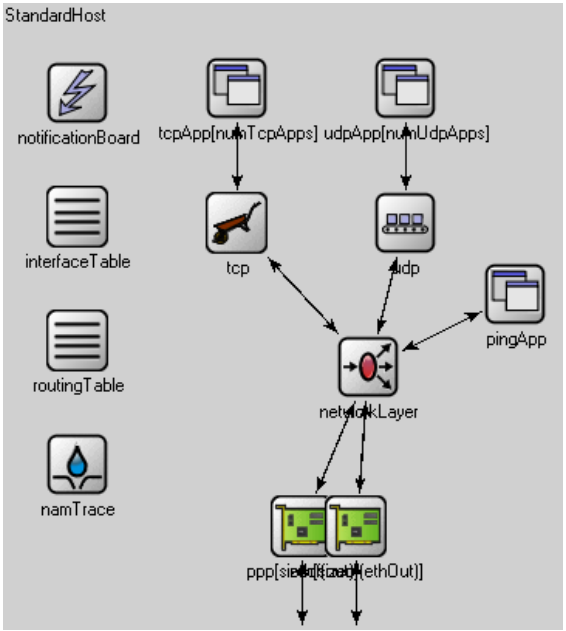
Bu ağın gerçeklemesi tamamıyla INET içerisinde hazır olarak bulunan modüller kullanılarak yapılmıştır. Gönderici ve alıcı modüller olarak *StandardHost* modülü, yönlendirici olarak da *Router* modülü kullanılmıştır. Ağ topolojisi, 3.2.1 de anlatıldığı gibi ".ned" dosyası içerisinde tanımlanmıştır. Aşağıda ".ned" dosyası içerisindeki tanımlar anlatılmaktadır.

Çizelge 1 Aktarım (Import) tanımları

```
import
    "StandardHost",
    "Router",
    "FlatNetworkConfigurator";
```

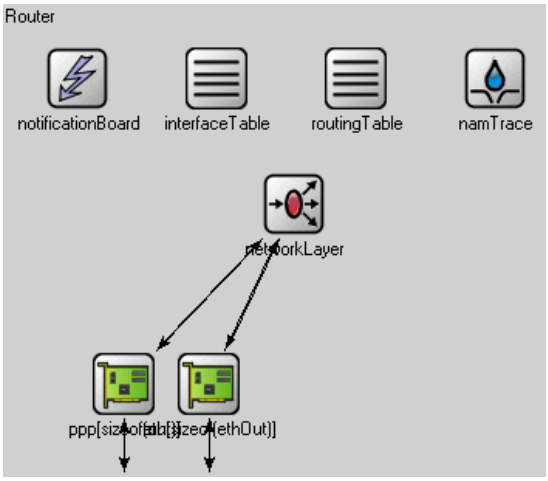
Dosyada, öncelikle simülasyonda kullanılacak olan modüller tanımlanır (Çizelge 1). Şekil 2'de gösterilen *StandardHost* modülü, Bölüm 3.3.2'de anlatılan *NotificationBoard*, *InterfaceTable* ve *RoutingTable* modülleri haricinde, ihtiyaca göre geliştirilecek/kullanılacak olan TCP ve UDP uygulamalarının modüle bağlanmasını sağlayacak *tcpApp* ve *udpApp* dizilerini içerir. Ayrıca modülün ağa

bağlanmasını sağlayan biri giriş, diğeri çıkış için olmak üzere iki adet PPP (Point-to-Point Protocol) bağlantı arabirimi dizisi vardır.



Şekil 2 StandardHost modülü

Şekil 3'de *Router* modülü görünmektedir. Bu modülün iki adet PPP bağlantı dizisi vardır. Yönlendirme işlemi C++ kaynak dosyasında tanımlanmıştır. Her iki modülünde kendilerine ait kişiselleştirme parametreleri vardır. Bu parametrelerin nasıl ve ne amaçla kullanıldıkları "omnetpp.ini" dosyası incelenirken gösterilecektir.



Şekil 3 Router Modülü

FlatNetworkConfigurator,bölüm 3.3.3'te anlatıldığı gibi, ağdaki IP modüllerine, adres atamalarını yapar ve topolojide tanımlanan bağlantılara göre sabit rotaları belirler.

Aktarım tanımlarından sonra, Çizelge 2'de gösterilen bağlantı tanımları gelir. Burada 2 adet hat tanımlanmaktadır; *fastlink*, 10 Mb/s hızında 0.1 ms gecikmeli hızlı bir hat, *slowlink* 1 Mb/s hızında 100 ms gecikmeli yavaş bir hattır.

Çizelge 2 Bağlantı tanımlamaları

```
channel fastlink
    datarate 10000000;
    delay 0.1ms;
endchannel

channel slowlink
    datarate 1000000;
    delay 100ms;
endchannel
```

Sistem modülü, *module* ve *endmodule* anahtar kelimeleri arasında tanımlanır. Çizelge 3'te gösterilen modülün adı *file_sender_receiver* olarak belirlenmiştir, *submodules* anahtar kelimesi kullanılarak, bu modül içinde yer alan alt modüller tanımlanmaktadır. Alt modüller tanımlanırken öncelikle değişken adı yazılır, ":" 'dan sonra değişkenin tipi yazılarak, ";" ile satır sonlandırılır. Eğer modülün ".ned" dosyası içinde ilklendirmek istenilen değişkenleri var ise, *parameters* anahtar kelimesi kullanılarak, değişkenlere değerleri atanır. Çizelge 3'te, *configurator* modülüne, ip adresi ataması gereken modül tipleri, ip atamaması gereken modül tipleri, atanacak ip adreslerinin ağ adresi ve ağ maskesi verilmektedir. *sender*, *router* ve *receiver* genel kullanım amaçlı modüller oldukları için bu ağ topolojisinde ilklendirilmesi gereken özel değişkenleri yoktur, fakat "omnetpp.ini" dosyası içerisinde gerekli değişken atamaları yapılacaktır. d*isplay* anahtar kelimesi kullanılarak, modüllerin simgeleri, pozisyonları gibi gösterim amaçlı kullanılacak olan bilgileri tanımlanır. c*onnections* anahtar kelimesi kullanılarak, topolojideki bağlantılar tanımlanır, *nocheck* değiştiricisi (modifier), bu ağdaki cihazların bütün kapılarının bağlı olup olmadığının ağ sıfırlanırken kontrol edilmesine gerek olmadığını belirtmektedir. Çizelgeden de

anlaşılabileceği gibi, gönderici, yönlendiriciye hızlı hat kullanılarak, alıcı ise yavaş hat kullanılarak bağlanmaktadır.

Çizelge 3 Sistem modülü tanımları

```
module file_sender_receiver
    submodules:
        configurator: FlatNetworkConfigurator;
            parameters:
                moduleTypes = "Router StandardHost",
                nonIPModuleTypes = "",
                networkAddress = "80.61.0.0",
                netmask = "255.255.255.0";
            display: "p=56,64;i=block/cogwheel";
        sender: StandardHost;
            display: "p=42,144;i=device/laptop";
        router: Router;
            display: "p=147,143;i=abstract/router";
        receiver: StandardHost;
            display: "p=252,144;i=device/server";
    connections nocheck:
        sender.out++ --> fastlink --> router.in++;
        sender.in++ <-- fastlink <-- router.out++;

        router.out++ --> slowlink --> receiver.in++;
        router.in++ <-- slowlink <-- receiver.out++;
endmodule
```

".ned" dosyası içerisinde birden çok modül tanımlanabilir. Bu nedenle, ağ seviyesindeki modül, Çizelge 4'te gösterildiği gibi, *network* ve *endnetwork* anahtar kelimeleri kullanılarak belirtilir. Bu örnekte, ağa *tutorial_network* adı verilmiştir, tipi ise Çizelge 3'te gösterilen *file_sender_receiver* olarak atanmıştır.

Çizelge 4 Ağ tanımlaması

```
network tutorial_network: file_sender_receiver
endnetwork
```

Simülasyon için gerekli olan konfigürasyon ve girdi verisi genellikle "omnetpp.ini" adı verilen bir dosyada tanımlanır. Bu dosyada öncelikle, *[General]* anahtar kelimesi kullanılarak, yüklenmesi gereken ".ned" dosyaları, simülasyon süresi ve simülasyonu yapılacak ağ tanımlanır (Çizelge 5).

Çizelge 5 Genel tanımlar

```
[General]
preload-ned-files = *.ned @E:\INET-20061020\nedfiles.lst
sim-time-limit = 8s

network = tutorial_network
```

Daha sonra ağda tanımlanan modüllerin dinamik değişkenlerine, değerleri atanır. Buradaki örnekte, her hangi bir UDP uygulaması kullanılmayacağı için, Çizelge 6'da, ***.numUdpApps*, 0 olarak atanmıştır. Bu şekilde bir kullanım, ağda bulunan, *numUdpApps* değişkenine sahip bütün modüllerin, bu değişkenlerini tek komutla sıfırlamaya yarar.

Çizelge 6 Uygulama tanımları (UDP)

```
# UDP Uygulamaları
**.numUdpApps=0
**.udpAppType="UDPBasicApp"
```

TCP uygulamaları için değer atanırken ise, alıcı ve gönderici için ayrı ayrı tanımlamalar yapılır (Çizelge 7). Gönderici için, INET içerisinde yer alan *TCPSessionApp* uygulaması seçilmiştir. Bu uygulama, bir TCP oturumunu temsil eder, dosya gönderimi için kullanılabileceği gibi oturum gerektiren diğer simülasyonlarda da ihtiyaca göre düzenlenerek kullanılabilir. Burada bağlantı adresi, bağlantı kapısı, bağlantının ne zaman açılacağı gibi özellikler tanımlanmaktadır.

Alıcı uygulaması olarak *TCPSinkApp* uygulaması seçilmiştir. Bu modül de INET içerisinde hazır bulunan bir modüldür, adından da anlaşılabileceği gibi gelen paketleri aldığını bildirdikten sonra paketleri işlemeden kuyruktan atar. İki uygulamada da *address* değişkenine değer atanmamış, *TCPSessionApp* uygulamasında *connectAddress* değişkenine *receiver* değeri atanmıştır. Bu değerler, simülasyon başlatılırken *FlatNetworkConfigurator* tarafından gerçek değerleriyle değiştirilecektir.

Çizelge 7 Uygulama tanımları (TCP)

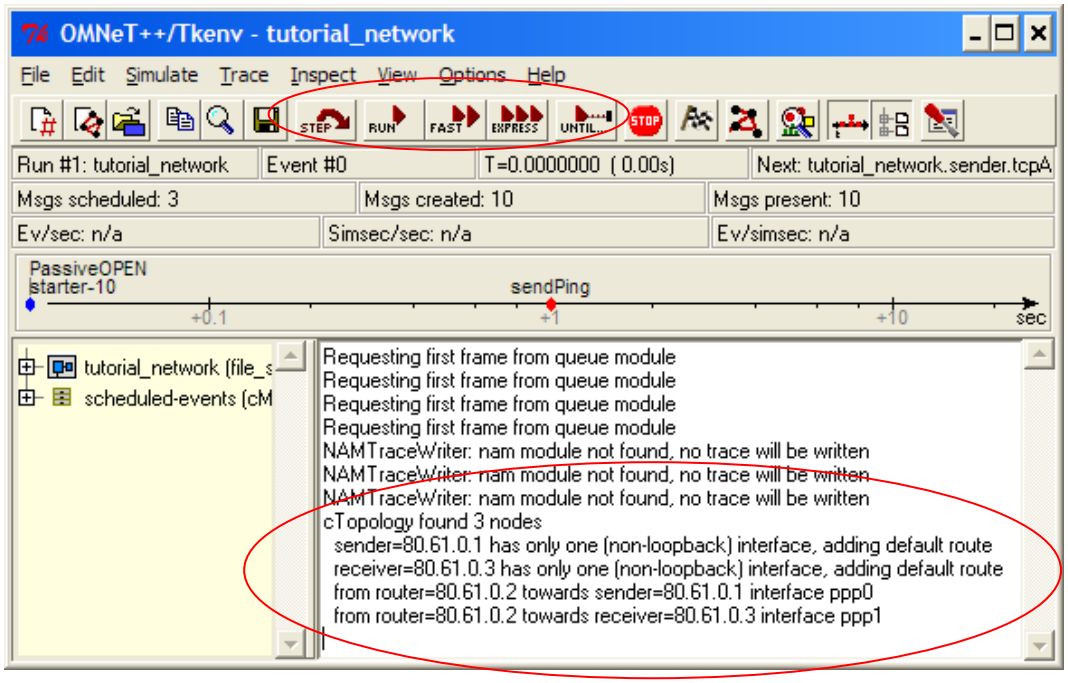
```
# TCP Uygulamaları
**.sender.numTcpApps=1
**.sender.tcpAppType="TCPSessionApp"
**.sender.tcpApp[0].active=true
**.sender.tcpApp[0].address=""
**.sender.tcpApp[0].port=-1
**.sender.tcpApp[0].connectAddress="receiver"
**.sender.tcpApp[0].connectPort=1000
**.sender.tcpApp[0].tOpen=exponential(0.1)
**.sender.tcpApp[0].tSend=0
**.sender.tcpApp[0].sendBytes=1000000  # 1 MByte
**.sender.tcpApp[0].sendScript=""
**.sender.tcpApp[0].tClose=0

**.receiver.numTcpApps=1
**.receiver.tcpAppType="TCPSinkApp"
**.receiver.tcpApp[0].address=""
**.receiver.tcpApp[0].port=1000
```

Bu tanımlardan sonra ağda bulunan cihazların TCP/IP yığıtlarındaki (stack) genel özellikleri belirlenir (Çizelge 8).

Çizelge 8 TCP/IP tanımları

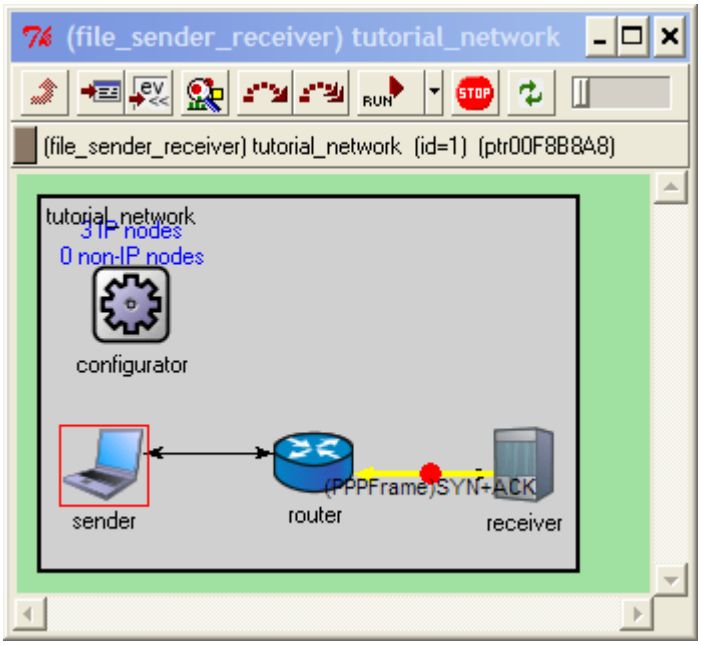
```
# TCP ayarları
**.tcp.recordStats=1
**.tcp.mss=1024
**.tcp.advertisedWindow= 65536
**.tcp.sendQueueClass="TCPVirtualDataSendQueue"
**.tcp.receiveQueueClass="TCPVirtualDataRcvQueue"
**.tcp.tcpAlgorithmClass="TCPReno"

# IP ayarları
**.routingFile=""
**.ip.procDelay=10us
**.sender.IPForward=false
**.receiver.IPForward=false

# ARP konfigürasyonu
**.arp.retryTimeout = 1
**.arp.retryCount = 3
**.arp.cacheTimeout = 100
**.networkLayer.proxyARP = true

# Makine ve yönlendiricideki NIC konfigürasyonu
**.ppp[*].queueType = "DropTailQueue"
**.sender.ppp[*].queue.frameCapacity   = 60
**.receiver.ppp[*].queue.frameCapacity = 60
**.router.ppp[*].queue.frameCapacity   = 6
```

Simülasyon çalıştırıldığında, OMNeT++ programının grafik ara biriminde, FlatNetworkConfigurator tarafından atanan ağ adresleri görülebilir (Şekil 4). Araç çubuğundan çalıştır komutu verilerek simülasyon başlatılır.



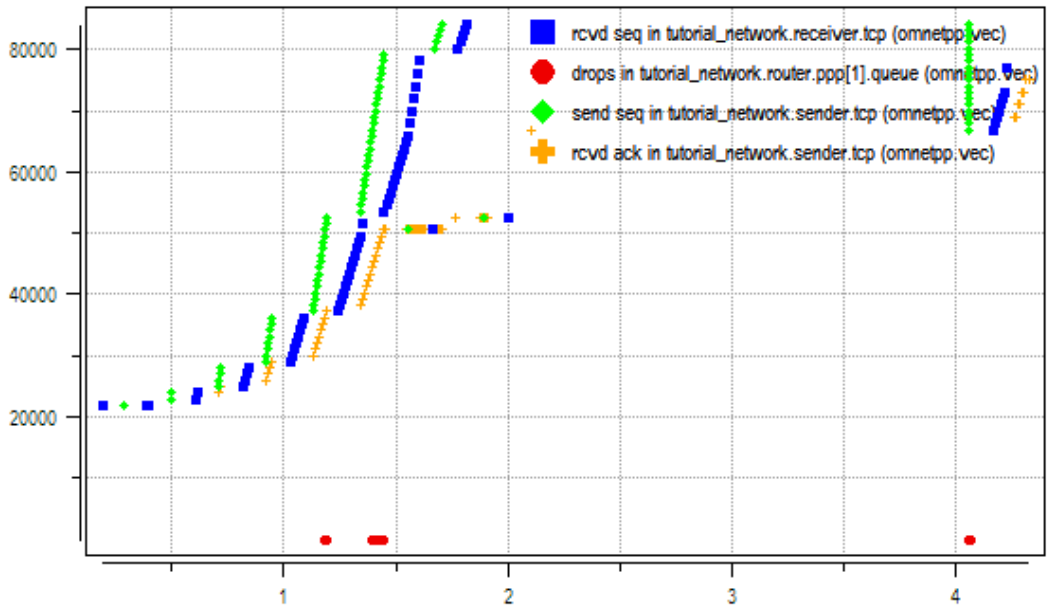
Şekil 4 Dosya gönderme simülasyonu

Simülasyon başlatıldığında, süreç görsel olarak izlenebilir (Şekil 5).



Şekil 5 Çalışan simülasyon

OMNeT++ simülasyon sırasında ağ ile ilgili birçok istatistik amaçlı kullanılabilecek veri oluşturur. Araç kutusu içinde bulunan *plove* programı kullanılarak bu veriler çizdirilebilir. Şekil 6'da, bu veriler içerisinden çizdirilmiş olan paket sırası numaraları (sequence number) gösterilmektedir. Şekilde, yavaş-başlangıç evresi açıkça görülmektedir, ayrıca kaybolan paketler ve bu paketlerin tekrar gönderimi de şekilde görülmektedir. Çizelge 8'den de görülebileceği gibi, paket kaybı oluşması için yönlendiricinin tampon kapasitesi özellikle çok düşük seçilmiştir.



Şekil 6 Paket sırası numaraları grafiği

EK 2 TFN2KTCP Modülü

Çizelge 1 TFN2KTCP.cc

```
#define _CRT_RAND_S

#include <stdlib.h>

#include "TFN2KTCP.h"
#include "IPControlInfo.h"
#include "IPv6ControlInfo.h"

// Modül, Define_Module() makrosu kullanılarak OMNeT++'e
kaydedilir
Define_Module(TFN2KTCP);

// Simülasyon motoru çalıştıktan sonra yapılacak olan
ilklendirmeler, burada yapılır
void TFN2KTCP::initialize()
{
    // İlk mesajın zamanı konfigürasyondan okunarak, gerekli
zamanlayıcı kurulur
    simtime_t t = par("t").doubleValue();
    scheduleAt(t, new cMessage("timer"));
}

// Simülasyon çekirdeği tarafından, sınıf bir mesaj iletisi
aldığında tetiklenen fonksiyondur
void TFN2KTCP::handleMessage(cMessage *msg)
{
   // Modül gelen mesajlara cevap vermemektedir
   if (msg->arrivedOn("ipv4In"))
      ;
   // Zamanlayıcı mesajı alınmış ise
   else if (msg->isSelfMessage())
   {
      // Atak paketi yollanır
      sendSpoofPacket();
      // Bir sonraki mesaj iletisi için zamanlayıcı tekrar
kurulur
      scheduleAt(simTime()+exponential(0.1),new
Message("timer"));
   }

   delete msg;
}

// Atak paketinin hazırlandığı fonksiyon
void TFN2KTCP::sendSpoofPacket()
{
    // Yeni bir TCP bölütü hazırlanır
    TCPSegment *tcpseg = new TCPSegment("tfn2ktcp");

    // Konfigürasyondan kaynak ve hedef adres okunur
    IPvXAddress dstAddr =
IPAddressResolver().resolve(par("dstAddress"));
    IPvXAddress srcAddr =
IPAddressResolver().resolve(par("srcAddress"));
```

```
    // Bölütün diğer özellikleri rastlantısal olarak atanır
    int srcPort = getrandom(0,65535);
    int dstPort = getrandom(0,65535);
    tcpseg->setSrcPort(srcPort);
    tcpseg->setDestPort(dstPort);
    tcpseg->setSequenceNo(getrandom(0,1)?
(getrandom(0,65535)+(getrandom(0,65535) << 8)) : 0);
    tcpseg->setAckNo(getrandom(0,1)?
(getrandom(0,65535)+(getrandom(0,65535) << 8)) : 0);

    if (getrandom(0, 1))
    {
      if (getrandom(0, 1))
            tcpseg->setSynBit(true);
      else
            tcpseg->setAckBit(true);
    }
    else
    {
      tcpseg->setSynBit(true);
      tcpseg->setAckBit(true);
    }

    tcpseg->setWindow(getrandom(0,1) ? getrandom(0,65535):
0);
    tcpseg->setUrgBit(false);
    tcpseg->setByteLength(TCP_HEADER_OCTETS);

    // TCP mesajının taşıyacağı veri yine rastlantısal olarak
atanır
    cMessage *msg = new cMessage("tfn2kpayload");
    msg->setByteLength(getrandom(1, 4096));

    tcpseg->addPayloadMessage(msg, tcpseg->sequenceNo()+1);
    tcpseg->setPayloadLength(getrandom(1, 4096));

    // Hazırlanan bölüt IP katmanına yollanır
    sendToIP(tcpseg, srcAddr, dstAddr);

}
// TCP bölütünü, IP paketi içine sarmalayan fonksiyondur
void TFN2KTCP::sendToIP(TCPSegment *tcpseg, IPvXAddress src,
IPvXAddress dest)
{
    // IP kontrol verisi hazırlanır
    IPControlInfo *controlInfo = new IPControlInfo();
    controlInfo->setProtocol(IP_PROT_TCP);
    controlInfo->setDiffServCodePoint(0);
    controlInfo->setTimeToLive(getrandom(200,255));
    controlInfo->setSrcAddr(src.get4());
    controlInfo->setDestAddr(dest.get4());
    tcpseg->setControlInfo(controlInfo);

    // IP paketi, ipv4Out kapısından ağa yollanır
    send(tcpseg,"ipv4Out");

}
```

```
// min ve max arasında rastlantısal bir sayı döndürür
long TFN2KTCP::getrandom(int min, int max)
{
      return ((intuniform(0, 65535) % (int)(((max)+1)-
(min)))+(min));
}
```

Çizelge 2 TFN2KTCP.h

```
#ifndef __TFN2KTCP_H_
#define __TFN2KTCP_H_

#include <omnetpp.h>
#include "INETDefs.h"
#include "TCPSegment.h"
#include "IPAddressResolver.h"


class INET_API TFN2KTCP : public cSimpleModule
{
  protected:
    void sendToIP(TCPSegment *tcpseg, IPvXAddress src,
IPvXAddress dest);
    unsigned long initialSeqNum();
    void sendSpoofPacket();
      long getrandom(int min, int max);

protected:
    virtual void initialize();
    virtual void handleMessage(cMessage *msg);
};

#endif
```

Çizelge 3 TFN2KTCP.ned

```
simple TFN2KTCP
    parameters:
        srcAddress: string,  // yerel adres
        dstAddress: string,  // hedef adres
        t: numeric const;    // atak başlangıç zamanı
    gates:
        out: ipv4Out;
        in: ipv4In;
endsimple
```

EK 3 Orijinal TFN2K Kodu

Çizelge 1 TFN iletim kodu

```
void
tfntransmit (unsigned long from, unsigned long to, int proto,
char id, char *target)
{
  char buf[BS], data[BS];
  struct ip *ih=(struct ip*) buf;
  struct icmp *ich=(struct icmp*)(buf + sizeof(struct ip));
  struct udp *udh=(struct udp*)(buf + sizeof(struct ip));
  struct tcp *tch=(struct tcp*)(buf + sizeof(struct ip));
  struct sa sin;
  char *p;
  int tot_len=sizeof(struct ip), ssock;

  memset (data,0,BS);
  data[0]=PROTO_SEP;
  data[1]=id;
  data[2]=PROTO_SEP;
  strncpy(data+3,target,BS-3);

  sin.fam=AF_INET;
  sin.add=to;
  memset(buf,0,BS);

  ih->ver=4;
  ih->ihl=5;
  ih->tos=0x00;
  ih->tl=0;
  ih->id=htons(getrandom(1024,65535));
  ih->off=0;
  ih->ttl=getrandom(200,255);
  ih->sum=0;
  ih->src=from;
  ih->dst=to;

  tot_len+=sizeof(struct tcp);
  ih->pro=TCP;
  ssock=socket(AF_INET,SOCK_RAW,TCP);
  p=buf+sizeof(struct ip)+sizeof(struct tcp);

  tch->src=htons(getrandom(0,65535));
  tch->dst=htons(getrandom(0,65535));
  tch->seq=getrandom(0,1)?htonl(getrandom(0,65535)+
          (getrandom(0,65535)<<8)):0;
  tch->ack=getrandom(0,1)?htonl(getrandom(0,65535)+
          (getrandom(0,65535)<<8)):0;
  tch->off=0;
  tch->flg=getrandom(0,1)?(getrandom(0,1)?SYN:ACK):SYN| ACK;
  tch->win=getrandom(0,1)?htons(getrandom(0,65535)):0;
  tch->urp=0;
  tch->sum=0;
  encode64(data,p,strlen(data));
  tot_len+=strlen (p);
  tch->sum=cksum((u16*)tch,tot_len>>1);
  ih->tl=tot_len;
```

```
  sin.dp=htons(tch->dst);


  setsockopt(ssock,IP,IP_HDRINCL,"1",sizeof("1"));
  if(sendto(ssock,buf,tot_len,0,(struct sockaddr*)&sin,
     sizeof(sin))<0)
    perror ("sendto");

  close(ssock);
}
```