Düzce Üniversitesi Bilim ve Teknoloji Dergisi

Araştırma Makalesi

Derin Paket İncelemesi için Önerilen Yeni Bir Örüntü Eşleştirme Algoritması

Merve ÇELEBİ [a,*], Uraz YAVANOĞLU [b]

[a] *Bilgisayar ve Öğretim Teknolojileri Eğitimi, Eğitim Fakültesi, Hatay Mustafa Kemal Üniversitesi, Hatay, TÜRKİYE*

[b] *Bilgisayar Mühendisliği Bölümü, Mühendislik Fakültesi, Gazi Üniversitesi, Ankara, TÜRKİYE*

** Sorumlu yazarın e-posta adresi: merveorakci@mku.edu.tr*

DOI: 10.29130/dubited.1131544

Öz

Derin Paket İnceleme (Deep Packet Inspection-DPI), hem paket başlığı hem de paket yükü üzerinde ayrıntılı analizler gerçekleştirerek ağ trafiğinin tam görünürlüğünü sağlayan teknolojidir. DPI ile iyi bilinen kötü amaçlı yazılım imzaları ve saldırı sırası, saldırganın izlediği yol ve kullandığı tekniklerin birleşimi olarak tanımlanan saldırı deseninin tespiti yapılabilmektedir. Bu doğrultuda, ağ güvenliği veya devlet gözetimi gibi uygulamalarda kullanılabilmesi yönüyle DPI, kritik bir öneme sahiptir. Bu çalışmada, tek seferde taranan bayt sayısını artırarak DPI sürecini hızlandırmayı amaçlayan blok tabanlı bir örüntü eşleştirme algoritması önerilmiştir. Farklı sayıda örüntü içeren veri kümeleri kullanılarak Aho-Corasick (AC), Rabin-Karp (RK), Wu-Manber (WM) ve bu çalışmada önerilen algoritma üzerinde örüntü eşleştirme testleri gerçekleştirilmiş ve bu algoritmaların performansları karşılaştırılmıştır. AC, WU ve RK algoritmalarına kıyasla bu çalışmada önerilen algoritma, daha yüksek bir performans göstermiştir.

Anahtar Kelimeler: *Derin paket inceleme, Örüntü eşleştirme, Ağ güvenliği, Ağ trafiği analizi*

A New Pattern Matching Algorithm for Deep Packet Inspection

Abstract

Deep Packet Inspection (DPI) is a technology that provides full visibility into network traffic by performing detailed analysis on both packet header and packet payload. DPI technique is used to detect well-known malware signatures. Additionally, the attack pattern which includes the attack order, the path followed by the attacker and the techniques used by the attacker can be detected. Accordingly, DPI has a critical importance as it can be used in applications i.e network security or government surveillance. In this study, a block-based pattern matching algorithm is proposed, which aims to speed up the DPI process by increasing the number of bytes scanned at once. Pattern matching tests are performed on Aho-Corasick (AC), Wu-Manber (WM), Rabin-Karp (RK) algorithms and the algorithm in this study by using datasets containing different numbers of patterns, and the performances of these algorithms are compared. Compared to the AC, RK and WM algorithms, the proposed algorithm in this study has a higher performance.

Keywords: *Deep packet inspection, Pattern matching, Network security, Network traffic analysis*

Geliş: 16/06/2022, Düzeltme: 10/10/2022, Kabul: 15/11/2022

I. GİRİŞ

Modern ağlarda performansın korunması; ağın güvenliğini tehdit eden sorunların belirlenmesi, hizmet kalitesi (QoS) gereksinimlerinin karşılanması veya kaynak tüketiminin iyileştirilmesi gibi konuların önceliklendirilmesi ile sağlanır. Bu doğrultuda, ağ trafiğinin izlenmesi ve analiz edilmesi, ağ hizmetlerinin erişilebilir olmasını ve beklendiği gibi çalışmasını garanti eden uygulamaları içerir. Bu süreçte, ağ paketleri incelenmesi gereken hedefler olarak kabul edilir [1].

Ağ paketlerini inceleme sürecinde derinliğe bağlı olarak sınıflandırılmış üç temel yöntem kullanılır [2]. Yüzeysel Paket İnceleme (Shallow packet inspection-SPI) tekniği ile ağ paketinin yalnızca başlık bilgisi incelenebilir. Bu doğrultuda, ağ paketinin başlık kısmından gönderici ve alıcının IP adresleri, port numaraları, iletilen mesajların parçalandığı paket sayısı veya paketlerin hedefe ulaşmadan evvel yapabileceği atlama sayısı gibi bilgiler elde edilebilir. Ancak, birçok uygulamanın rastgele port numarası kullanması, şifreli protokollerin ortaya çıkması, bazı uygulamaların aldatma amacıyla diğer protokollere atanan port numaralarını kullanması veya bir protokolün başka bir protokolün yerini alması gibi sebepler, bu yöntemin uygulama protokolünü tanımlama hususundaki becerisinin yetersiz olduğunu kanıtlar niteliktedir [3], [4]. Orta seviyeli paket inceleme (Medium packet inspection-MPI), ağ trafiğini incelemek için kullanılan bir diğer tekniktir. Bu teknik, yaygın olarak uygulama vekil sunucuları tarafından kullanılır ve ağ paketinin başlık bilgisinin yanı sıra yük kısmını sunum katmanına kadar inceler. Bu yönüyle MPI, SPI tekniğine göre daha kapsamlı bir analiz gerçekleştirir [5]. MPI tekniği kullanılarak şüpheli olarak nitelendirilen içeriklerin açılmasını veya indirilmesini engelleyen uygulamalar geliştirilebilir ve uygulama katmanında bulunan uygulama komutları ile sunum katmanındaki dosya formatları incelenerek paket önceliklendirmesi yapılabilir. SPI ve MPI tekniklerinin aksine DPI tekniği, bir ağ paketinin tüm katmanlarının başlık bilgilerini ve yükünü kapsayacak şekilde bir inceleme gerçekleştirir. SPI, MPI ve DPI tekniklerinin OSI ve TCP/IP modellerine göre derinlikleri Tablo 1'de gösterilmiştir.

Tablo 1. *Ağ paketi inceleme teknikleri*

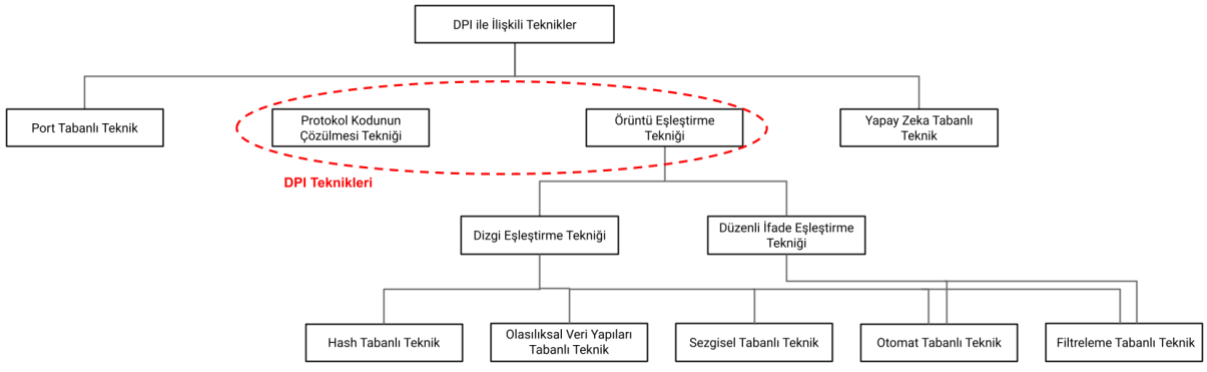
Seviye	SPI	MPI	DPI	OSI	TCP/IP
7			✓	Uygulama katmanı	Uygulama katmanı
6		✓	✓	Sunum katmanı	
5		✓	✓	Oturum katmanı	
4	✓	✓	✓	Taşıma katmanı	Taşıma katmanı
3	✓	✓	✓	Ağ katmanı	Ağ katmanı
2	✓	✓	✓	Veri bağlantı katmanı	Veri bağlantı katmanı
1	✓	✓	✓	Fiziksel katman	Fiziksel katman

DPI, belirli bir ağ noktasından geçen ağ paketlerinin derinlemesine analiz edilmesini ve analiz sonuçlarına göre çeşitli eylemlerin gerçekleştirilmesini sağlar. Bu inceleme yalnızca paket başlığını değil, aynı zamanda paket yükünü de kapsadığı için DPI tekniği derin inceleme olarak bilinir. DPI süreci tanıma ve eylem olmak üzere iki aşamadan oluşur. Tanıma, ağ paketini inceleme sürecini kapsar. Bu süreçte; uygulama protokolleri, virüsler, solucanlar, telefon numaraları veya kredi kartı numaraları gibi özel formata sahip veriler tespit edilebilir. Saldırı tespit sistemleri (intrusion detection systems-IDS), tanıma sürecine örnek olarak verilebilir. Bu sistemler, kötü amaçlı yazılımları tespit etmek için paket yükü içeriğini önceden tanımlanmış bir dizi imza veya örüntü ile karşılaştıran uygulamalardır.

DPI tekniğini kullanan bu uygulamalar sayesinde iyi bilinen kötü amaçlı yazılım imzalarının yanı sıra ağ akışı ile bağlantılı olarak saldırı sırası, saldırganın izlediği yol ve kullandığı teknikleri içeren saldırı deseninin tespiti gerçekleştirilebilir [2]. Tanıma sürecinin tamamlanmasından sonraki aşama, inceleme sonuçlarının eyleme dönüştürülmesidir. Ağ güvenliğinin öncelikli olmadığı sistemlerde log kayıtlarının tutulması yeterliyken ağ güvenliğinin öncelikli olduğu sistemlerde log kayıtlarının tutulmasının yanı sıra ağdan düşürme gibi eylemler de gerçekleştirilebilir [3]. IDS'lerin aksine saldırı önleme sistemleri (Intrusion Prevention System-IPS), saldırıyı durdurma veya önleme eylemini gerçekleştirme üzerine kurgulanır. Günümüzdeki siber saldırılara karşı geliştirilen sistemler, genellikle bütünleşiktir ve saldırı tespit ve önleme sistemleri (Intrusion Detection and Prevention System-IDPS) olarak bilinir.

İki uç istasyon arasındaki iletişim üzerinde kontrolün sağlanması, gerçek zamanlı olarak hem paket başlığı hem de paket yükü üzerinde bilgi sahibi olma yeteneği ile doğrudan ilişkilidir. Yalnızca paket başlığı ve meta veriler üzerinde gerçekleştirilen inceleme; paket yükü üzerinde de denetim gerektiren içerik filtreleme, IDS, IPS, veri kaybı önleme sistemleri (Data Loss Prevention-DLP) ve dolandırıcılık tespiti gibi uygulamalar için yetersizdir [6]. Bu doğrultuda DPI, SPI ve MPI tekniklerine göre ağ trafiğinin daha güvenilir bir biçimde izlenmesini ve analiz edilmesini sağlar. Ayrıca, günümüzde ağ trafiğinin neredeyse tamamı şifreli olduğu için kötü niyetli kişiler, şifreli trafiği kendilerini gizlemek için kullanarak şifreli kanallar üzerinden saldırılarını gerçekleştirebilir [7]. Bu sebeple, siber saldırıların tespitinde paket başlığının yanı sıra, paket yükünün de incelemeye dâhil edilmesi gerekir.

Modern iletişim sistemleri ve ağları üzerinde trafiğin izlenmesi ve analiz edilmesi, doğruluğun sağlanması ve büyük verilerin gerçek zamanlı olarak etkili bir şekilde işlenmesi gibi bazı zorluklar içerir. Özellikle hücresel ağlardaki ağ trafiği modeli, cihaz hareketliliği ve ağ heterojenliği gibi çeşitli faktörler nedeniyle karmaşık davranışlar gösterir. Bahsi geçen zorluklar; büyük veri olarak adlandırılan yapıdan değerli bilgilerin elde edilmesi sürecinde karşılaşılan verinin hacmi, hızı, doğruluğu ve çeşitliliği gibi faktörlerle ilişkilidir [1], [8]. Ağ karmaşıklığının artması sonucu oluşan heterojen yapıdaki bu ağ trafiğinin toplanması ve değerlendirilmesi, büyük miktardaki verilerin çevrimiçi olarak analiz edilmesi için verimli mekanizmalar gerektirir. Bu verimli mekanizmalar, birden fazla tekniğin bir araya getirilmesi ile oluşturulan güçlü ve hibrit sistemlerdir. Bu sebeple DPI tekniğinin, ağ trafiğinin analizinde kullanılan diğer tekniklerle birlikte incelenmesi önemlidir. DPI tekniğinin temel prensibi, paket yükünün incelenmesidir. Bu prensip göz önünde bulundurularak DPI ve DPI ile ilişkili yöntemler, Şekil 1'de gösterilmiştir [3], [9], [10].



***Şekil 1**. DPI ile ilişkili teknikler*

II. LİTERATÜR TARAMASI

Otomat tabanlı yaklaşımlarda veri yükündeki her baytın işlenmesi, bir veya birkaç hafıza erişimi gerektirdiğinden DPI'deki eşleştirme süreci, hesaplama açısından yoğun ve zaman alıcı bir iştir. Bu durum, DPI sürecini uzatabilir. Bu nedenle, genel DPI performansı büyük ölçüde örüntü eşleştirme verimliliğine, yani sonlu durum makinesinin (FSM) etkinliğine bağlıdır [3]. Nondeterministic finite

automata (NFA) ve deterministic finite automata (DFA) çoklu örüntü eşleştirme problemleri için kullanılan FSM'lerdir. İfade gücü olarak iki FSM de eşdeğerdir. Ancak DFA, yoğun bellek kullanımı NFA ise yoğun hesaplama gerektirir. Bu alandaki mevcut araştırmaların çoğu, depolama ve performans arasında bir denge sağlamaya yöneliktir [22]-[24].

Teorik bilgisayar bilimi ve biçimsel dil teorisinde RE; bir arama kalıbını, yani dilleri veya bir dizi dizgiyi tanımlar. RE'deki her sembol ya düzenli bir ASCII karakteridir ya da bazı özel anlamları temsil eden bir meta karakterdir. RE, bir dizi dizgiyi temsil edebilirken bir dizgi, yalnızca kendisini temsil edebilir [3]. RE'ler güçlü ve esnek tanımlama kabiliyeti sayesinde birçok açık kaynaklı ve ticari DPI uygulamasında yaygın olarak kullanılır [25]-[27]. Örüntü eşleştirme işleminde girdi olarak RE'lerin kullanıldığı süreçler için bellek açısından verimli mimarilerin oluşturulmasına odaklanan bazı çalışmaların [28]-[30] yanı sıra, 31 numaralı çalışmada örüntü eşleştirme doğruluğunun ve hızının iyileştirilmesi amaçlanmıştır. Ayrıca, ağ paketlerini arabelleklerde depolamayı ve akışları yeniden birleştirmeyi gerektirmeyen ve sırasız paketleri işleyebilen RE tabanlı bir DPI mimarisi önerilmiştir. En basit türde RE'ler olan dizgiler, sabit boyutlu örüntülerdir. Karmaşık RE'lerden daha hızlı bir eşleştirme gücüne sahip olan otomat tabanlı AC [32] algoritması, uygulanmasının kolay olması sebebiyle dizgi eşleştirme için yaygın olarak kullanılır. AC algoritmasının desen eşleştirme sürecinde uygulanması, önbellek alanını daha büyük durum geçiş tabloları için kullanışsız hale getirir. Bu durumun bir sonucu olarak büyük boyuttaki örüntü veri kümeleri için eşleştirme hızı düşer. Bellek gereksinimini azaltıp önbellek kullanımını etkin bir biçimde gerçekleştirmek için geçiş tablosunun sıkıştırılması, AC algoritmasının araştırma konularındandır. Bazı iyileştirilmiş AC algoritmaları [33]-[36], otomatların depolanmasında gerekli olan hafıza alanını azaltmaya odaklanırken 37 numaralı çalışmada, desen eşleştirme sürecinde bellek erişim sayısının ve enerji tüketiminin azaltılması amacıyla değişken adımlı AC-DFA algoritması önerilmiştir.

Paket yükündeki her baytı tek tek işleyen otomat tabanlı örüntü eşleştirme algoritmalarının aksine sezgisel tabanlı eşleştirme algoritmalarının temel prensibi, eşleştirme işlemini hızlandırmak için bazı sezgileri kullanarak mümkün olduğunca fazla sayıda paket yükü karakteri atlamaktır. Tekli örüntü eşleştirme algoritması Boyer-More (BM) [20] ve çoklu örüntü eşleştirme algoritması WM [21] sezgisel tabanlı eşleme algoritmalarıdır. BM, duruma bağlı atlamalar yaparak arama modelinin performansını iyileştirir. Kontrolü sağdan sola doğru gerçekleştiren BM, *Good-Suffix* (hedef metnin ve örüntünün eşleşen eki) ve *Bad-Character* (hedef metnin ve örüntünün eşleşmeyen karakteri) kurallarına göre kaymalar gerçekleştirir. Bilinen saldırı örüntülerinin tespiti amacıyla BM algoritmasını uygulayan birçok çalışma [38]-[41] mevcuttur. IDS'lerin performansını ve verimliliğini iyileştirmek amacıyla karakter karşılaştırma sayısını azaltmaya odaklanan çalışmalar [42]-[44] ise BM algoritmasının iyileştirilmiş versiyonu olarak sunulmuştur. BM algoritmasının performansını sınırlayan önemli bir husus, paralel olarak birden fazla örüntüyü işleyememesidir. Bu doğrultuda, BM algoritmasının bir uzantısı olarak geliştirilen WM algoritması, birden fazla örüntüyü aynı anda işleme özelliğine sahiptir. IDS'lerin performansını ve verimliliğini iyileştirmek amacıyla gereksiz eşleştirme denemelerinin sayısını azaltarak kullanılan CPU döngüsü sayısını azaltmaya odaklanan [45]-[47] numaralı çalışmalar, WM algoritmasının iyileştirilmiş versiyonu olarak sunulmuştur. Ayrıca, kaydırma mesafesinin kısa olmasına sebep olan kısa örüntülerden kaynaklanan performans kayıplarının azaltılması amacıyla WM algoritmasının iyileştirilmiş versiyonu olarak sunulan [48] numaralı çalışmada, tüm desenlerin uzunluklarına göre farklı desen setlerine bölünmesi ve bu setlerin sırayla işlenmesi ile performansı düşüren kısa desenlerin etkisinin azaltılması hedeflenmiştir. Hash tabanlı algoritmalar, paket yükünün içeriğini karakterler üzerinden örüntü ile karşılaştırmak yerine hash değerlerinin bir karşılaştırmasını yapar. RK [19] hash tabanlı bir örüntü eşleştirme algoritmasıdır. Bu algoritma hem örüntü hem de paket yükünün örüntü ile aynı uzunluklu alt dizgileri için hash hesaplamaları gerçekleştirir ve her örüntü için bu işlemi tekrar eder. Hesaplama karmaşıklığından kaynaklanan ek yükler, örüntü eşleştirme performansı üzerinde olumsuzluk teşkil eder.

Bu çalışmada önerilen algoritma, tek seferde taranan bayt sayısını artırarak eşleştirme sürecini hızlandırmayı amaçlar. Bu yönüyle blok tabanlıdır ve birden fazla örüntüyü aynı anda işleme özelliğine sahiptir. Bu algoritmada ilk m karakteri aynı olan örüntüler, diğer örüntülerden farklılaştıkları minimum uzunluklarına göre bir hash tablosunda tutulur. Daha sonra, bu uzunluklar kullanılarak hash tablosundan

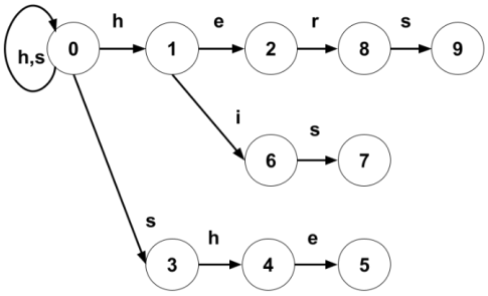
örüntülere direkt erişim sağlanır. Hash tablosu sayesinde, WM algoritmasında olduğu gibi ilk m karakteri aynı olan örüntüler üzerinde gerçekleştirilen doğrusal arama işleminden kaynaklanan performans kayıpları yaşanmaz. Bu doğrultuda, önerilen algoritma hem sezgisel hem de hash tabanlı olması yönüyle hibrit bir yapıya sahiptir. Ek olarak, WM algoritmasında bir örüntü tespit işlemi gerçekleştirildikten sonra paket yükü üzerinde kaydırma mesafesi her durumda 1'dir. Önerilen algoritmada ise kaydırma mesafesi örüntü içerisinde başka bir örüntünün tamamının ya da bir kısmının mevcut olup olmadığına göre belirlenir. Bu doğrultuda, önerilen algoritmanın eşleştirme performansı kaydırma mesafesi ile doğru orantılı olarak artar.

III. YÖNTEM

Bu bölümde, önerilen algoritma ile performansları karşılaştırılan AC, WM ve RK algoritmaları gözden geçirilmiş, önerilen algoritmanın mimarisi detaylı bir şekilde sunulmuştur.

A. AHO-CORASICK ALGORİTMASI

AC, otomat tabanlı bir örüntü eşleştirme algoritmasıdır ve örüntü eşleştirme işleminde girdi olarak önceden tanımlanmış imzalardan oluşan bir grup dizgiyi kabul eder. Örüntü tespitinin gerçekleştirilebilmesi amacıyla paket yükü AC otomatı üzerinde gezdirilir. Bu otomat; ilerleme, başarısızlık ve çıkış olarak adlandırılan üç fonksiyondan oluşur. AC otomatını oluşturan bu fonksiyonlar Şekil 2'de gösterilmiştir [32]. AC otomatında *g* ilerleme fonksiyonu, *f* başarısızlık fonksiyonu ve *o* çıkış fonksiyonu olarak temsil edilmiştir.



a. İlerleme fonksiyonu

i	1	2	3	4	5	6	7	8	9
f(i)	0	0	0	1	2	0	3	0	3

b. Başarısızlık fonksiyonu

i	output(i)
2	{he}
5	{she,he}
7	{his}
9	{hers}

c. Çıkış fonksiyonu

Şekil 2. *AC Algoritması fonksiyonları*

0 numaralı durum, otomat için başlangıç durumunu gösterir. İlerleme fonksiyonu *g*, bir durum-karakter ikilisini bir duruma eşler. Eşlenen bu durum, başarılı bir geçiş veya başarısızlık geçişi sonucu elde edilebilir. Örnek olarak Şekil 2'de $g(0,h)=1$ başarılı bir geçişi temsil ederken $g(1,\sigma)$ fonksiyonunda "e" veya "i" dışında herhangi bir karakter için başarısızlık geçişi mevcuttur. Başarısızlık fonksiyonu *f* bir durum-karakter ikilisinin bir duruma eşlenememesi halinde kullanılır. Mevcut karakter ile gidilebilecek herhangi bir durum yoktur ve başarısızlık geçişi ile elde edilen karakter-durum ikilisi doğru bir eşleme için kullanılır. Örneğin, Şekil 2'de görüldüğü gibi durum 4'ten "e" karakteri ile durum 5'e geçiş

mevcuttur. Ancak paket yükünde mevcut karakter olarak "e" bulunmuyorsa $g(4,e)$ geçişi gerçekleştirilemez ve $f(4)=1$ başarısızlık fonksiyonu kullanılarak başarısızlık geçişi gerçekleştirilir. Artık durum 1'e geçilmiştir ve durum 1'den diğer durumlara olan geçişler yeni bir eşleştirme için değerlendirilir. Çıkış fonksiyonu *o*, bir durumu bir veya daha fazla örüntüye eşler. Yani, mevcut durumda sonlanan tüm örüntülerin indisleri çıkış fonksiyonu kullanılarak saklanır. Örneğin, AC otomatı üzerinde gezdirilen paket yükü için mevcut durum 5 ise iki örüntü tespit edilmiştir. Bu durum $o(5)=\{he, she\}$ fonksiyonu ile belirlenir.

n değerinin taranan metnin boyunu, *m* değerinin tüm örüntülerdeki toplam karakter sayısını ve *k* değerinin toplam örüntü sayısını temsil ettiği kabul edilirse; AC algoritması, tüm örüntüleri $O(n + m + z)$ zamanında tespit eder. Burada *z* değeri, taranan metin içerisinde bulunan örüntülerin toplam sayısıdır.

B. RABIN-KARP ALGORİTMASI

RK, hash tabanlı bir örüntü eşleştirme algoritmasıdır ve paket yükünün içeriğini karakterler üzerinden örüntü ile karşılaştırmak yerine hash değerlerinin bir karşılaştırmasını yapar. *m* uzunluğundaki her örüntü için bir hash değeri hesaplanır. Aynı zamanda, incelenen m uzunluğundaki alt dizginin de hash değeri hesaplanır. Örüntünün paket yükü üzerinde kaydırılması sırasında herhangi bir örüntü-alt dizgi eşleşmesi tespit edilirse doğrulama için örüntü ve alt dizgi bayt bazında karşılaştırılır.
n değerinin taranan paket yükünün boyunu, *m* değerinin örüntünün boyunu ifade ettiği varsayılırsa ve $n > m$ ise RK algoritmasının en iyi ve ortalama çalışma süresi $O(n+m)$'dir. En kötü çalışma süresi ise m uzunluğundaki örüntünün ve paket yükünden elde edilen *m* uzunluğundaki tüm alt dizgilerin eşit hash değerlerine sahip oldukları durumda ortaya çıkar ve $O(nm)$ olarak kabul edilir. AAA örüntüsü ve AAAAAAA metni için hesaplanan hash değerleri birbirine eşittir ve RK algoritması için en kötü çalışma süresinin geçerli olduğu durumu örnekler.

C. WU-MANBER ALGORİTMASI

BM algoritmasının bir uzantısı olarak geliştirilen sezgisel tabanlı WM algoritması, ön işleme ve tarama olarak adlandırılan iki aşamadan oluşur. Ön işleme aşamasında, eşleştirme penceresinin uzunluğu *m*, en kısa örüntünün uzunluğuna göre belirlenir ve daha sonra desen eşleştirme sürecinde kullanılan SHIFT, HASH ve PREFIX tabloları oluşturulur. Eşleştirme penceresinin kaydırılması, örüntü eşleştirme sürecini hızlandıran *Bad Character* kaydırma yaklaşımını temel alır. Ancak, burada kaydırma hesaplamaları tek bir karakter yerine karakter bloğu kullanılarak gerçekleştirilir. Karakter bloğunun uzunluğu genellikle 2 veya 3 olarak kabul edilir.

m değerinin eşleştirme penceresinin uzunluğunu, *B* değerinin karakter bloğunun uzunluğunu, *X* değerinin mevcut eşleştirme penceresinin içerisindeki karakter bloğunu temsil ettiği varsayılmış ve $T=t_1,t_2,\ldots..,t_n$ paket yükü, *n* paket yükünün uzunluğu ve $P=\{p_1,p_2,\ldots..,p_n\}$ örüntü kümesi olarak kabul edilmiştir.

Her örüntünün ilk *m* karakterinden *B* boyutunda karakter blokları elde edilir ve her bloğun kaydırma mesafesi SHIFT tablosunda saklanır. Hash hesaplaması sonucunda *X* değerinin, SHIFT tablosunda i. indiste bulunduğu kabul edilirse SHIFT tablosu içerisindeki kaydırma mesafesi eşitlik (1) vasıtasıyla hesaplanır.

$$SHIFT[i] = \begin{cases} m - B + 1, X\ herhangi\ bir\ \text{örüntü}\ ile\ eşleşmiyorsa \\ \min\{m - q \backslash X[k] = P_j[q - B + k], 1 \le k \le B, P_j \in P\} \end{cases} \quad (1)$$

Eşitlik (1)'deki *q* değeri, *X* karakter bloğunun konumunun en sağındaki yeri temsil eder.

HASH tablosu, eşleştirme penceresi içerisinde aynı son eke sahip örüntüleri bulunduran liste girişlerini saklar. Her örüntünün ilk *m* karakterinin son *B* boyutlu karakter bloğunun hash değeri hesaplanır. Aynı hash değerine sahip tüm örüntüler aynı listeye alınır ve listenin girişi HASH tablosunda saklanır.

PREFIX tablosunun oluşumu HASH tablosunun oluşumu ile benzerdir. HASH tablosundan farklı olarak, hash değerinin hesaplanmasında örüntülerin ilk *m* karakterinin son eki yerine ön eki kullanılır.

WM algoritmasında örüntü arama işlemi aşağıdaki gibidir:

1. Eşleştirme penceresi, *T*'nin başlangıcına yerleştirilir. Paket yükü işaretçisi t_p, eşleştirme penceresinin son ek karakter bloğunu gösterecek şekilde konumlandırılır.
2. Eğer t_p işaretçisinin değeri paket yükünün son elemanını gösteren işaretçi değerinden büyükse arama işlemi sonlanır. Aksi durumda, eşleştirme penceresinin son ek karakter bloğunun hash değeri hesaplanır.
3. Elde edilen hash değeri, karakter bloğunun kaydırma mesafesini saklayan SHIFT tablosunda, arama işleminin gerçekleştirilmesinde kullanılır. Karakter bloğunun kaydırma mesafesinin 0 olması durumu, eşleştirme penceresinin bazı örüntülerin ilk *m* karakteri olduğu anlamına gelir ve 4. adıma gidilir. Kaydırma mesafesinin 0'dan büyük olduğu durumlarda ise t_p değeri kaydırma mesafesi kadar artırılır ve 2. adıma gidilir.
4. HASH tablosunda arama yapmak için SHIFT tablosunda da arama işleminin gerçekleştirilmesinde kullanılan hash değeri kullanılır. HASH tablosu üzerinde arama işleminin gerçekleştirilmesiyle aynı son ek karakter bloğuna sahip örüntülerin listesinin ilk elemanını gösteren işaretçi tespit edilir.
5. Mevcut eşleştirme penceresindeki *T*'nin ön ek karakter bloğunun hash değeri hesaplanır.
6. Listedeki her örüntü için önek karakter bloğunun hash değeri hesaplanır. Bu değer, 5. adımda hesaplanan hash değerine eşit ise örüntü ve paket yükü bayt bazında karşılaştırılır. Herhangi bir eşleşme mevcut ise eşleşme kaydedilir. t_p değeri bir artırılır ve 2. adıma gidilir.

N değerinin taranan metnin boyu, *P* değerinin örüntü sayısı ve *m* değerinin bir örüntünün boyutu olduğu varsayılırsa $M=mP$ tüm örüntülerin boyutu olur. $N \geq M$ ise herhangi bir örüntünün *B* boyutlu alt dizgisi bir kez işleme alınır ve bu sebeple SHIFT tablosu $O(M)$ zamanında oluşturulur. Tarama süresi iki duruma bağlıdır. Birinci durumda kaydırma mesafesi 0'dan büyüktür. Bu durumda, yalnızca bir kaydırma uygulanır ve ek bir iş yükü gerekmez. İkinci ve daha karmaşık durum, kaydırma mesafesinin 0 olduğu durumdur. Bu durumda, örüntüler üzerinde doğrusal arama gerçekleştirilir.

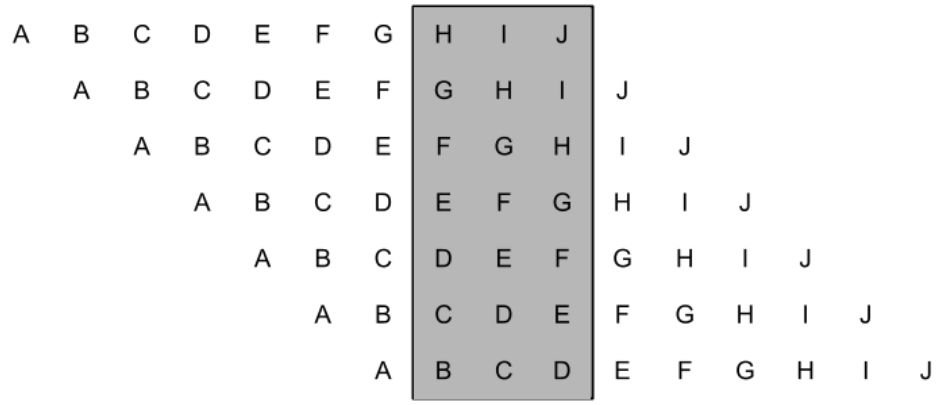
En fazla örüntü sayısı ($P=M/m$) kadar dizgi i. indiste bir kaydırma mesafesi oluşturur. *B* boyutundaki tüm olası alt dizgilerin sayısı en az $2M$'dir. Rastgele *B* boyutunda bir dizginin i. indiste bir kaydırma mesafesini oluşturma olasılığı, $0 \leq i \leq m-B+1$, $\leq 1/2m$'dir ve bir hash fonksiyonu $O(B)$ zamanında hesaplanır. Bu doğrultuda, kaydırma mesafesinin 0'dan farklı olduğu durumda toplam iş $O(BN/m)$ zamanında yürütülür. SHIFT tablosunda bir eşleşme olmadığı sürece, 0 kaydırma değeri için de iş miktarı $O(B)$ olur. Ancak bir eşleşmenin olduğu durumda, aynı son ek ve ön ek hash hesaplamalarına sahip örüntüler üzerinde doğrusal bir arama gerçekleştirilir. Bu arama işlemi, bir örüntü için $O(m)$ zamanında gerçekleştirilir.

D. ÖNERİLEN ALGORİTMA

Çok adımlı örüntü eşleştirme algoritmaları bir ağ paketinin birden çok baytının tek seferde taranmasına izin verir. Bu durum, örüntü eşleştirme performansını artırır. Önerilen blok tabanlı algoritma, tek seferde taranan bayt sayısını artırarak örüntü eşleştirme işlemlerini hızlandırmayı amaçlar. Bu algoritma, mevcut tek adımlı ya da çok adımlı örüntü eşleştirme algoritmalarına benzer şekilde, birden fazla örüntüyü aynı anda işleme özelliğine sahiptir.

Önerilen algoritma, örüntü eşleştirme işleminde bir grup dizgiyi girdi olarak kabul eder ve eşleştirme işlemini hızlandırmak için önceden tanımlanmış kuralları uygulayarak mümkün olduğunca fazla sayıda paket yükü karakteri atlamayı hedefler. Bu doğrultuda, bütün karakterleri tek tek kontrol eden doğrusal aramadan daha iyi bir sonuç elde edilmesi amaçlanmıştır. Örüntü eşleştirme sırasında; bir pencere, incelenecek karakterleri içerir ve eşleştirme sonucu, pencerenin kaydırılması gereken bir sonraki konumu belirtir.

Algoritma, ön işleme ve tarama olarak adlandırılan iki aşamadan oluşur. Ön işleme aşamasında, eşleştirme penceresinin uzunluğunu temsil eden *m* değeri belirlenir ve örüntülerin ilk *m* baytı için bir OFSET tablosu oluşturulur. Her örüntünün ilk *m* baytından *k* bayt alt dizgileri elde edilir ve her dizginin kaydırma mesafesi, OFSET tablosunda saklanır. Şekil 3'te bir *k* bayt alt dizginin bir *m* bayt örüntüden nasıl çıkarıldığı gösterilmektedir. Burada *k* uzunluğu 3, *m* uzunluğu ise 10 olarak kabul edilmiştir. *k* uzunluğunun 3'ten küçük bir değer seçilmesi durumunda, üretilen alt dizgi sayısı için gereken bellek ihtiyacı fazla olacaktır. *k* uzunluğunun büyük seçilmesi durumu ise hesaplama karmaşıklığını artırır. Dolayısıyla bu çalışmadaki *k* uzunluğu 3 olarak belirlenmiştir. OFSET tablosu oluşturulurken her örüntünün ilk *m* uzunluğu kullanılmıştır. Bu sebeple *m* uzunluğunun en kısa örüntüden küçük olmaması gerekir. Bu çalışmada kullanılan en kısa örüntünün uzunluğu 10 olduğu için *m* değeri 10 olarak belirlenmiştir.



***Şekil 3**. Alt dizgi çıkarımı*

Hash hesaplaması sonucunda mevcut eşleştirme penceresinin içerisindeki X değerinin OFSET tablosunda hash. indiste bulunduğu varsayılırsa OFSET tablosu içerisindeki kaydırma mesafesi, eşitlik (1) vasıtasıyla hesaplanır.

$$OFSET[i] = \begin{cases} m-k+1, X\ karakter\ bloğu\ herhangi\ bir\ \text{örüntünün}\ alt\ dizgisi\ değil \\ m-k-\max\left(OFSET[hash]\right) \end{cases} \quad (1)$$

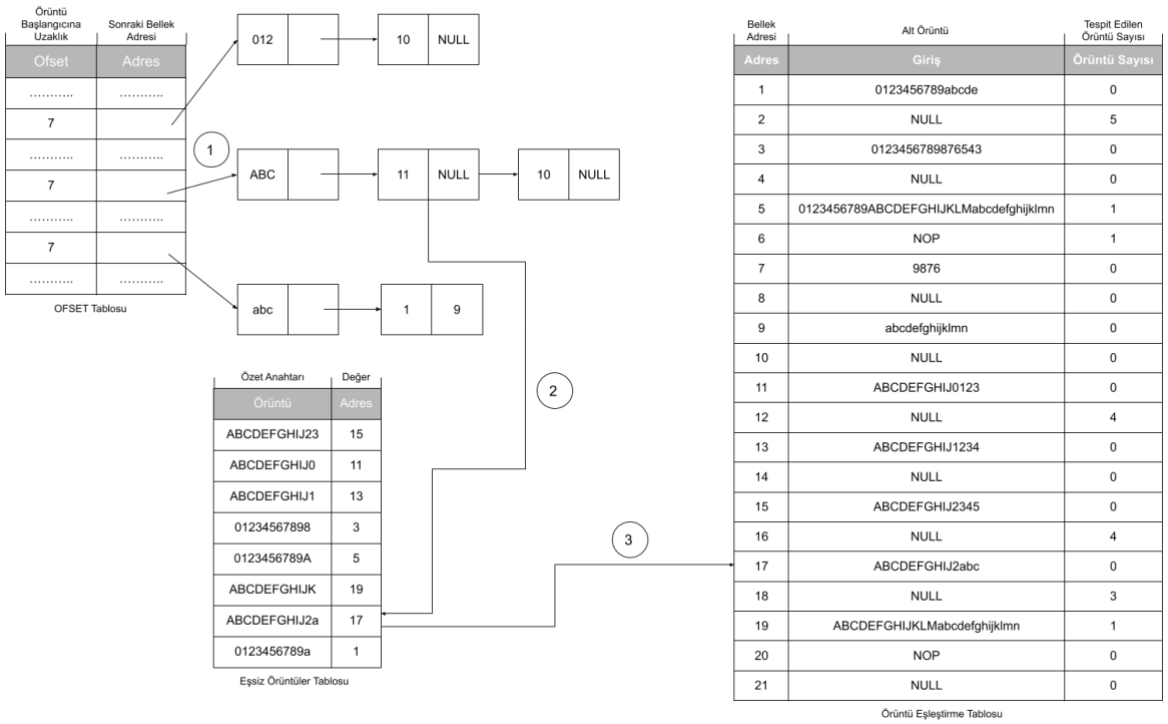
Kaydırma mesafesinin 7'den küçük olduğu durumlarda ağ paketi üzerinde kaydırma mesafesi kadar atlama gerçekleştirilir. Bu mesafe 7 iken örüntü eşleştirme işlemi gerçekleştirilir. Eşleştirme işlemi başarısız olursa ağ paketi üzerinde yalnızca 1 bayt atlanır. Başarılı bir eşleştirme olursa kaydırma mesafesi en iyi durumda, tespit edilen örüntü uzunluğu+(*m*-*k*+1) kadar; en kötü durumda ise 1 olur. En iyi durum, örüntü içerisinde başka bir örüntünün tamamının ya da bir kısmının mevcut olmadığı durumdur. En kötü durumda ise örüntünün ikinci baytından itibaren başka bir örüntünün alt örüntüsü mevcuttur.

OFSET tablosunda her *k* bayt alt dizge için ofset ve adres olmak üzere iki tanımlayıcı özellik tutulur. Ofset değeri, alt dizginin başlangıç konumunun örüntünün başlangıç konumuna olan uzaklığını ifade eder. Adres değeri ise her örüntünün ilk *m* baytının aynı son *k* bayt alt dizgisine sahip örüntülerin listesinin ilk elemanını gösteren bir işaretçi tutar. Bu listenin her bir elemanı, aynı ilk *l* bayt alt dizgilerine sahip örüntülerin listesinin ilk elemanını gösteren bir işaretçi tutar. Bu liste ise örüntülerin eşsiz olmasını sağlayan indis değeriyle birlikte tam eşleştirme işleminin gerçekleştirildiği örüntü eşleştirme tablosuna erişimin sağlanması için bir işaretçi tutar. Bu işaretçinin NULL değerine sahip olması durumunda, aynı indis değerine sahip birden fazla örüntünün mevcut olduğu anlaşılır ve eşsiz örüntüler tablosu, örüntü eşleştirme tablosuna erişim sağlanması için kullanılır.

Eşsiz örüntüler tablosu basit bir hash tablosudur ve örüntüler, eşsiz oldukları yani diğer örüntülerden farklılaştıkları minimum uzunlukları kullanılarak bu tabloya yerleştirilir. ABCDEFGHIJ1234 ve ABCDEFGHIJ0123 örnek örüntüleri için eşsiz örüntüler tablosuna ABCDEFGHIJ1 ve ABCDEFGHIJ0 alt örüntüleri yerleştirilir. Burada örüntülerin eşsiz olmalarını sağlayan minimum bayt sayısı iki örüntü için de 11'dir. Bu tablo, hash değeri olarak örüntü eşleştirme tablosuna erişim sağlayan bir işaretçi tutar.

Tam eşleştirme işlemi, örüntü eşleştirme tablosu kullanılarak gerçekleştirilir. Bu tabloda her örüntünün alt örüntüleri ardışık bellek adreslerinde tutulur. Bu sayede eşleştirme işlemi için bir sonraki adresin bulunması, basit bir artırma operasyonu ile gerçekleştirilir. Bir örüntü, başka bir örüntüyü içeriyorsa içerdiği örüntünün bitiş noktasına kadar olan kısım, bir alt örüntü oluşturur ve bu alt örüntü tabloya yerleştirilir. Örüntünün geri kalan kısmı için ise başka örüntüyü içerme durumu kontrol edilir ve oluşturulan alt örüntüler ardışık bellek adreslerine yerleştirilir. Alt örüntüler tek bir örüntüyü içerebileceği gibi birden fazla örüntüyü de içerebilir. Bu sebeple, mevcut indiste tespit edilen örüntü sayısı için örüntü eşleştirme tablosunda bir alan tutulur. Bu durum, algoritmaya birden fazla örüntüyü aynı anda işleme özelliği kazandırır ve yeni örüntü eşleştirme süreci, tespit edilen örüntüden sonraki baytlar üzerinde gerçekleştirilir. Ancak, bir örüntü başka bir örüntünün içerisinde alt örüntü oluşturacak şekilde konumlanmış ise örüntünün kalan kısmının tespit edilebilmesi için bir sonraki örüntü tespit işlemi, bu örüntünün diğer örüntü içerisindeki başlangıç indisinden itibaren gerçekleştirilir. Örüntü eşleştirme tablosunda, her örüntünün son alt örüntü bloğunu tutan bellek alanından bir sonraki alanda herhangi bir içerik saklanmaz. Örüntü eşleştirme tablosunda bu bellek alanına gelindiğinde örüntü tespitinin tamamlandığı anlaşılır ve yeni örüntü tespit işlemi için indis değeri belirlenir. Bu adresteki örüntü sayısı alanı, sonraki örüntü tespit işlemi için başlangıç indisi değerini tutar. Bu sayede, önceki aşamada tespiti tamamlanmayan örüntüler için eşleştirme süreci devam eder.

ABCDEFGHIJ2abc örüntüsü için örüntü tespit süreci Şekil 4'te gösterilmiştir. Örnek tespit sürecinde *k* değeri 3, *m* değeri 10 ve *l* değeri 2 olarak seçilmiştir. Öncelikle, örüntünün ilk *m* baytının son *k* elemanını içeren alt dizgi için hash hesaplama işlemi gerçekleştirilir ve bu alt dizginin OFSET tablosunda bulunduğu adres alanı belirlenir. Adres alanının belirlenmesinden sonraki adım, örüntünün ilk *l* elemanını içeren alt dizgi için hash hesaplamasının gerçekleştirilmesidir. Bu hash hesaplaması ile aynı ilk *l* bayta sahip örüntü listesinin ilk elemanına ulaşılır. Sonraki adım, doğru eşleştirmenin gerçekleştirilebilmesi için listedeki elemanlar üzerinde gezilmesidir. Mevcut elemanda indis alanı 11 ve örüntü eşleştirme tablosuna erişim sağlanması için gerekli işaretçi alanı NULL değerine sahiptir. Bu durumda, 11 indis değerine sahip birden fazla örüntü mevcuttur ve bu indis değeri kullanılarak eşsiz örüntüler tablosundaki uygun bellek adresine erişilir. Son adımda, erişilen bellek alanındaki adres değeri, örüntü eşleştirme tablosuna erişim sağlamak için kullanılır. ABCDEFGHIJ2abc örüntüsü tek bir örüntüden oluşur. Bu sebeple eşleştirme tablosunda ilk alt örüntüyü içeren adrese erişildiğinde tespit edilen örüntü sayısının 0 olduğu görülür. Bir sonraki adres alanına erişildiğinde ise alt örüntü alanının NULL olduğu görülür ve örüntü eşleştirme sürecinin tamamlandığı anlaşılır. Ayrıca, alt örüntü alanının NULL olduğu bu adreste tespit edilen örüntü sayısı alanının 3 olduğu görülür ve bu bilgi, yeni örüntünün tespit edilmesi sürecinde kullanılır. ABCDEFGHIJ2abc örüntüsünün son üç baytı, abcdefghijklmn örüntüsünün ilk 3 baytını içerir. Bu sebeple yeni eşleştirme süreci, ağ paketi içerisinde konumlanan ABCDEFGHIJ2abc örüntüsünün son üç baytından başlar.



***Şekil 4**. Örüntü tespit süreci örneği*

Rastgele B boyutunda bir dizginin i değerinde bir kaydırma mesafesini oluşturma olasılığı, $0 \leq i \leq m-B+1, \leq 1/2m$'dir ve bir hash fonksiyonu $O(B)$ zamanında hesaplanır. Bu doğrultuda, kaydırma mesafesinin 7'den küçük olduğu durumlarda toplam iş, $O(BN/m)$ zamanında yürütülür. OFSET tablosunda bir eşleşme olmadığı sürece, kaydırma mesafesinin 7 olduğu durum için de iş miktarı $O(B)$ olarak hesaplanır. Aksi durumda, yani OFSET tablosunda bir eşleşme olduğunda, aynı son ek ve ön ek hash hesaplamalarına sahip örüntüler üzerinde bir arama işlemi gerçekleştirilir. WM algoritmasında herhangi bir eşleşmenin tespit edilebilmesi için örüntüler üzerinde doğrusal arama gerçekleştirilir. Önerilen algoritmada ise arama işlemi basit bir hash tablosu kullanılarak gerçekleştirilir. Örüntüler, eşsiz oldukları minimum uzunluklara göre hash tablosuna yerleştirilir. Daha sonra, bu uzunluk kullanılarak aranan örüntüye hash tablosundan direkt erişim sağlanır. Hash tablosu, en iyi durumda $O(1)$ zamanında işlem yürütür. En kötü durumda ise işlem süresi doğrusal aramaya yaklaşır.

IV. BULGULAR

Bu bölümde öncelikle analiz sürecinde kullanılan teknolojiler hakkında bilgi verilmiş, daha sonra analizlerde kullanılan veri kümesinin nasıl oluşturulduğundan bahsedilmiştir. Son olarak; AC, WM, RK algoritmaları ve bu çalışmada önerilen algoritma kullanılarak örüntü eşleştirme testleri gerçekleştirilmiş ve elde edilen performansların karşılaştırılması sunulmuştur.

Tablo 2, oluşturulan donanım yapılandırmasını gösterir. Analizlerde kullanılan örüntü kümeleri, kötü niyetli aktiviteleri barındıran ağ paketlerinin imzalarını içeren Snort [49] kurallarından elde edilmiştir. Farklı sayıda örüntü içeren veri kümeleri içerisindeki ağ paketlerinin ortalama uzunlukları Tablo 3'te gösterilmiştir. Bu kümeler içerisindeki örüntülerin uzunlukları 10 ila 171 bayt arasında değişmektedir. Örüntü kümesi oluşturulduktan sonraki adım, bu imzaları yük kısmında barındıran ağ paketlerini üretmek ve bu ağ paketlerinden bir veri kümesi elde etmektir. Bu veri kümesi içerisindeki her ağ paketi, 256 bayt uzunluğundadır ve rastgele seçilmiş bir örüntünün rastgele bir paket yükü konumuna eklenmesiyle üretilmiştir.

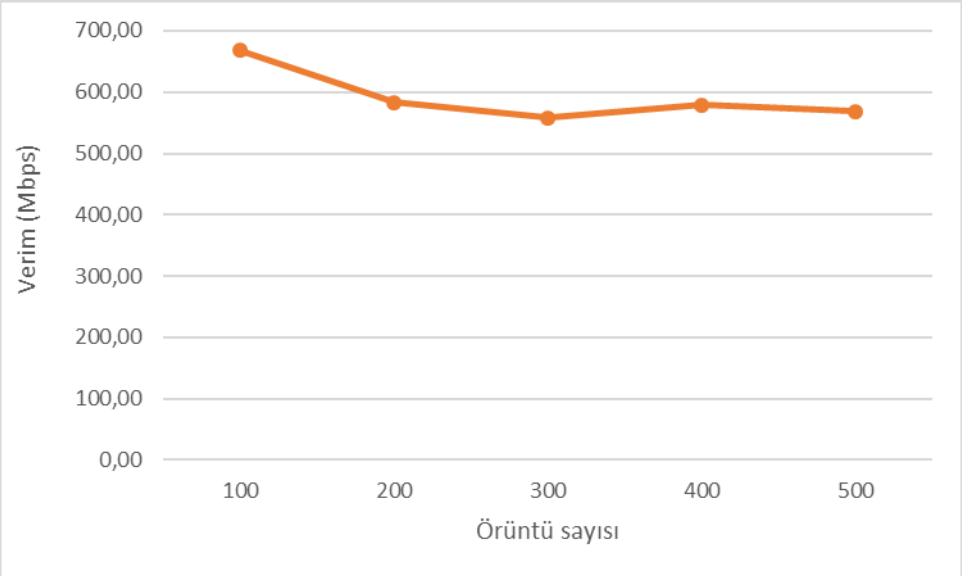
Tablo 2. *Donanım yapılandırması*

İşlemci	Intel Xeon E5-2640 v4
Bellek	32 GB DDR4-2666 4Rx4 ECC RDIMM

Tablo 3. *Paket istatistikleri*

Toplam paket sayısı	Ortalama örüntü uzunluğu
100	30,42
200	29,44
300	29,91
400	29,98
500	30,70

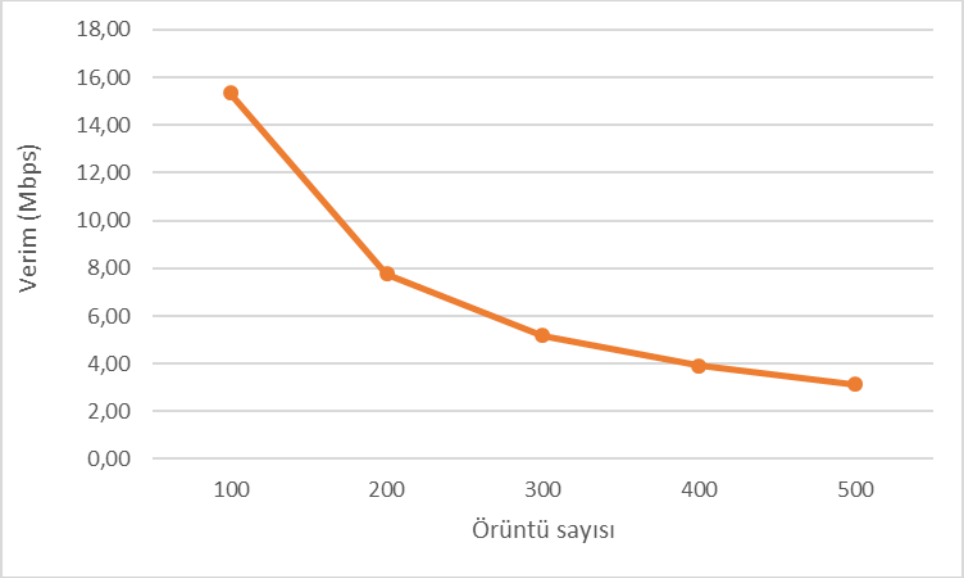
AC algoritması üzerinde gerçekleştirilen testler sonucunda elde edilen verim grafiği, Şekil 5'te verilmiştir. AC algoritmasının örüntü eşleştirme performansı incelendiğinde örüntü sayısının verim üzerindeki etkisinin oldukça kısıtlı olduğu görülmüştür. Bu etki, algoritmanın ihtiyaç duyduğu bellek alanı ve sahip olduğu erişim gecikmeleri ile ilişkilidir. İncelenecek örüntü sayısındaki artış, ihtiyaç duyulan bellek alanını da artırır. Dolayısıyla AC algoritmasının örüntü eşleştirme sürecinde uygulanması, önbellek alanını daha büyük durum geçiş tabloları için kullanışsız hale getirir. Bu durumun bir sonucu olarak büyük örüntü veri kümeleri için eşleştirme hızı düşer. CPU önbellekleri, ana bellek kaynaklarına kıyasla daha kısa erişim gecikmesi sağlar. Dolayısıyla küçük veri kümeleri için bu önbelleklere erişimin artması, AC algoritmasının daha yüksek seviyede verim elde etmesini sağlar. Bu durum, verimin düşmesinin nedenini açıklar.



Şekil 5. *AC algoritması verim grafiği*

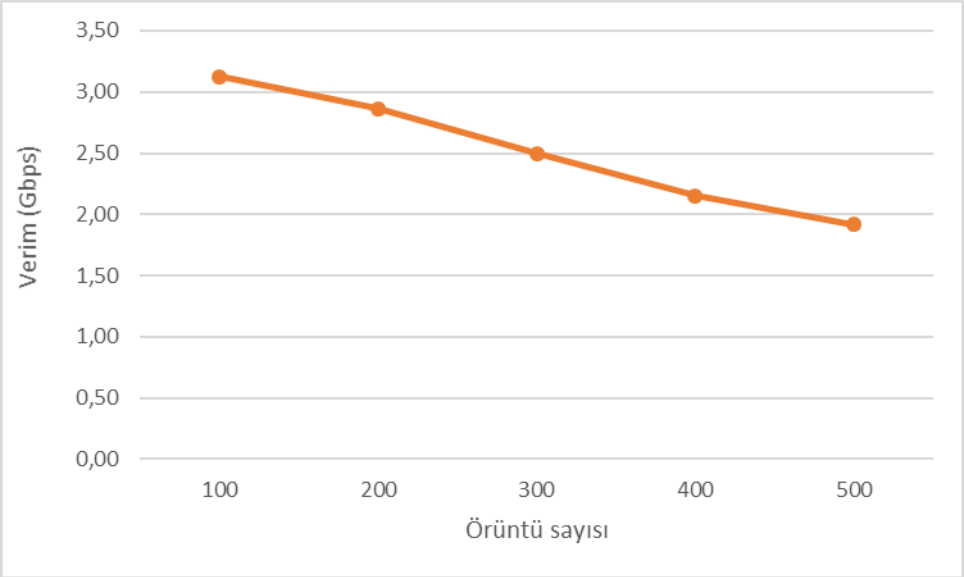
RK algoritması üzerinde gerçekleştirilen testler sonucunda elde edilen verim grafiği, Şekil 6'da gösterilmiştir. RK algoritmasının örüntü eşleştirme performansı incelendiğinde örüntü sayısının verim

üzerindeki etkisinin çok yüksek olduğu görülmüştür. Bu algoritma, paket yükünün içeriğini karakterler üzerinden örüntü ile karşılaştırmak yerine hash değerlerinin bir karşılaştırmasını yapar. Dolayısıyla *m* uzunluğuna sahip bir örüntü, ağ paketi üzerinde kaydırılırken hem örüntünün hem de ağ paketinin *m* uzunluğundaki alt dizgisinin hash değerine ihtiyaç duyulur. Bu durumun bir sonucu olarak incelenecek örüntü sayısındaki artış, RK algoritması için ek hash hesaplamaları gerektirir ve büyük örüntü veri kümeleri için eşleştirme hızı düşer.



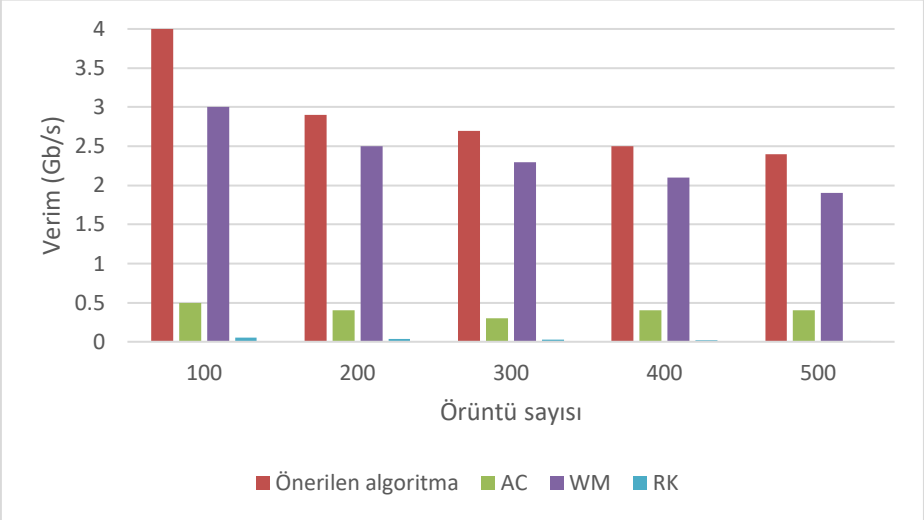
Şekil 6. *RK algoritması verim grafiği*

WM algoritması üzerinde gerçekleştirilen testler sonucunda elde edilen verim grafiği Şekil 7'de gösterilmiştir. WM algoritmasının örüntü eşleştirme performansı incelendiğinde örüntü sayısının verim üzerindeki etkisinin yüksek olduğu görülmüştür. İlk *m* baytının son *k* elemanı için aynı hash hesaplamasına sahip örüntüler, SHIFT tablosunda aynı adreste bulunur. İncelenecek örüntü sayısındaki artış, SHIFT tablosunda aynı adres alanında bulunan örüntü sayısını da artırır. Algoritma, bir eşleşme tespit edebilmek için SHIFT tablosunda aynı adres alanında bulunan ve ilk *l* eleman için aynı hash hesaplamasına sahip örüntüleri sırayla ağ paketi ile karşılaştırmak zorundadır. Bu durumun bir sonucu olarak büyük örüntü veri kümeleri için eşleştirme hızı düşer.



Şekil 7. *WM algoritması verim grafiği*

Önerilen algoritma üzerinde gerçekleştirilen testler sonucunda elde edilen verim grafiği Şekil 8'de gösterilmiştir. İlk *m* baytının son *k* elemanı için aynı hash hesaplamasına sahip örüntüler, OFSET tablosunda aynı adreste bulunur. İncelenecek örüntü sayısındaki artış, OFSET tablosundaki aynı adres alanında bulunan örüntü sayısını da artırır. Bu durumun eşleştirme performansı üzerindeki olumsuz etkisini azaltmak amacıyla arama işlemi doğrusal olarak değil, bir hash tablosu üzerinden gerçekleştirilir. Örüntüler, eşsiz oldukları minimum uzunluklara göre hash tablosuna yerleştirilir. Daha sonra, bu uzunluk kullanılarak aranan örüntüye hash tablosundan direkt erişim sağlanır. Ayrıca, örüntü eşleştirme tablosunun yapısı sayesinde bir örüntünün eşleştirilmesi sırasında diğer örüntüler için de bir eşleşme aranır. Bu durum, algoritmaya birden fazla örüntüyü aynı anda işleme özelliği kazandırır ve yeni örüntünün eşleştirilme süreci, tespit edilen örüntüden sonraki baytlar üzerinde gerçekleştirilir. Bu yaklaşımın temel amacı, mümkün olduğunca fazla sayıdaki paket yükü karakterini atlayarak eşleştirme işlemini hızlandırmaktır.



Şekil 8. *AC, WM, RK algoritmalarının ve bu çalışmada önerilen algoritmanın performanslarının karşılaştırılması*

Farklı sayıda örüntü içeren veri kümeleri kullanılarak AC, WM, RK algoritmaları ve bu çalışmada önerilen algoritma üzerinde gerçekleştirilen testler sonucunda elde edilen verimlerin karşılaştırılmasını sunan grafik, Şekil 8'de gösterilmiştir. Şekilde de görüldüğü gibi, en düşük performansa sahip algoritma RK algoritmasıdır. Bu algoritma, hem örüntü için hem de paket yükünün örüntü ile aynı uzunluklu alt dizgileri için hash hesaplamaları gerçekleştirir ve her örüntü için bu işlemi tekrar eder. Hesaplama karmaşıklığından kaynaklanan ek yükler, örüntü sayısının verim üzerindeki etkisinin çok yüksek olmasına sebep olur. En düşük performansa sahip ikinci algoritma AC'dir. Bu algoritma, blok tabanlı algoritmaların aksine her CPU döngüsünde bir paket yükü baytı işler. Bu durum algoritmanın performansı üzerinde olumsuz etki gösterir. Blok tabanlı WM algoritması ise RK ve AC'den daha yüksek bir performansa sahiptir. Bu algoritma, SHIFT tablosunda aynı adres alanında bulunan ve ilk *l* elemanı için aynı hash hesaplamasına sahip örüntüler üzerinde doğrusal arama gerçekleştirir. Ancak önerilen algoritmada, aranan örüntüye hash tablosundan direkt erişim sağlanır. Ayrıca, WM algoritmasında bir örüntü tespit işlemi gerçekleştikten sonra paket yükü üzerinde kaydırma mesafesi her durumda 1'dir. Önerilen algoritma için başarılı bir eşleştirme gerçekleşirse kaydırma mesafesi en iyi durumda, tespit edilen örüntü uzunluğu+(*m*-*k*+1) kadar olurken en kötü durumda 1 olur. Dolayısıyla önerilen algoritmanın eşleştirme performansı, kaydırma mesafesi ile doğru orantılı olarak artar.

V. SONUÇ

Bu çalışmada, tek seferde taranan bayt sayısını artırarak DPI'nın eşleştirme sürecini hızlandırmayı amaçlayan ve birden fazla örüntüyü aynı anda işleyebilen hibrit yapılı bir örüntü eşleştirme algoritması önerilmiştir. Farklı sayıda örüntü içeren veri setleri kullanılarak AC, WM, RK algoritmaları ve önerilen

algoritma üzerinde örüntü eşleştirme testleri gerçekleştirilmiş ve bu algoritmaların performansları karşılaştırılmıştır. En düşük performansa sahip RK algoritmasında hesaplama karmaşıklığından kaynaklanan ek yükler, örüntü sayısının verim üzerindeki etkisinin yüksek olmasına sebep olmuştur. Her CPU döngüsünde bir paket yükü baytı işleyen otomat tabanlı AC algoritması, en düşük performansa sahip ikinci algoritma olurken blok tabanlı WM algoritması RK ve AC algoritmalarından daha yüksek bir performans göstermiştir. WM algoritması, ilk *m* karakteri aynı olan örüntüler üzerinde doğrusal arama gerçekleştirirken önerilen algoritmada aranan örüntüye hash tablosundan direkt erişim sağlanır. Bu hash tablosunun kullanılmasıyla bütün karakterleri tek tek kontrol eden doğrusal aramaya kıyasla daha iyi bir sonucun elde edilmesi amaçlanmıştır. Ayrıca, WM algoritmasında bir örüntü tespiti gerçekleştikten sonra, paket yükü üzerindeki kaydırma mesafesi her durumda 1'dir. Öte yandan, önerilen algoritmadaki kaydırma mesafesi, örüntü içerisinde başka bir örüntünün tamamının ya da bir kısmının mevcut olup olmamasına göre belirlenir. Böylelikle önerilen algoritmanın eşleştirme performansı, kaydırma mesafesi ile doğru orantılı bir biçimde artar. Sonuç olarak, DPI eşleştirme sürecindeki en yüksek performansa sahip algoritmanın bu çalışmada önerilen algoritma olduğu saptanmıştır.

V. KAYNAKLAR

[1] M. Abbasi, A. Shahraki, and A. Taherkordi, "Deep learning for network traffic monitoring and analysis (NTMA): A survey," *Computer Communications*, vol. 170, pp. 19-41, 2021.

[2] G. A. Pimenta Rodrigues, R. de Oliveira Albuquerque, F. E. Gomes de Deus, R. T. de Sousa Jr, G. A. de Oliveira Júnior, L. J. Garcia Villalba, and T. H. Kim, "Cybersecurity and network forensics: Analysis of malicious traffic towards a honeynet with deep packet inspection," *Applied Sciences*, vol. 7, no. 10, pp. 1082, 2017.

[3] C. Xu, S. Chen, J. Su, S. M. Yiu, and L. C. Hui, "A survey on regular expression matching for deep packet inspection: Applications, algorithms, and hardware platforms," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 4, pp. 2991-3029, 2016.

[4] H. Tahaei, F. Afifi, A. Asemi, F. Zaki, and N. B. Anuar, "The rise of traffic classification in IoT networks: A survey," *Journal of Network and Computer Applications*, vol. 154, 102538, 2020.

[5] C. Parsons, *Deep Packet Inspection in Perspective: Tracing its lineage and surveillance potentials*, Kingston, Canada: Surveillance Studies Centre, Queen's University, 2008.

[6] X. de Carné de Carnavalet, and P. C. van Oorschot, "A survey and analysis of TLS interception mechanisms and motivations," *arXiv e-prints*, 2020.

[7] S. Canard, A. Diop, N. Kheir, M. Paindavoine, and M. Sabt, "BlindIDS: Market-compliant and privacy-friendly intrusion detection system over encrypted traffic," *In Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pp. 561-574, 2017.

[8] A. D'Alconzo, I. Drago, A. Morichetta, M. Mellia, and P. Casas, (2019). "A survey on big data for network traffic monitoring and analysis," *IEEE Transactions on Network and Service Management*, vol. 16, no. 3, pp. 800-813, 2019.

[9] R. Topolski, F. Press, and P. Knowledge, *NebuAd and partner ISPs: Wiretapping, forgery and browser hijacking*, Washington DC: FreePress, 2008.

[10] M. R. Shahid, G. Blanc, Z. Zhang, and H. Debar, "IoT devices recognition through network traffic analysis," *presented at 2018 IEEE international conference on big data*, pp. 5187-5192, IEEE, 2018.

[11] M. Finsterbusch, C. Richter, E. Rocha, J. A. Muller, and K. Hanssgen, "A survey of payload-based traffic classification approaches," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 2, pp. 1135-1156, 2013.

[12] L. Deri, M. Martinelli, T. Bujlow, and A. Cardigliano, "ndpi: Open-source high-speed deep packet inspection," *presented at 2014 International Wireless Communications and Mobile Computing Conference (IWCMC)*, pp. 617-622, IEEE, 2014.

[13] T. T. Nguyen, and G. Armitage, "A survey of techniques for internet traffic classification using machine learning," *IEEE communications surveys & tutorials*, vol. 10, no. 4, pp. 56-76, 2008.

[14] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors" *Communications of the ACM*, vol. 13, no. 7, pp. 422-426, 1970.

[15] D. E. Knuth, *The art of computer programming, sorting and searching*, vol. 3, Addison Wesley Longman Publishing Co. Inc., Redwood City, CA, USA, 1998.

[16] B. Fan, D. G. Andersen, M. Kaminsky, and M. D. Mitzenmacher, "Cuckoo filter: Practically better than bloom," *presented at Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*, pp. 75-88, 2014.

[17] M. Al-hisnawi, M. Ahmadi, "QCF for deep packet inspection," *IET Networks*, vol. 7, no. 5, pp. 346-352, 2018.

[18] B. Choi, J. Chae, M. Jamshed, K. Park, and D. Han, "{DFC}: Accelerating string pattern matching for network applications," *presented at 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pp. 551-565, 2016.

[19] R. M. Karp, and M. O. Rabin, "Efficient randomized pattern-matching algorithms," *IBM journal of research and development*, 31(2), 249-260, 1987.

[20] R. S. Boyer, and J. S. Moore, "A fast string searching algorithm," *Communications of the ACM*, vol. 20, no. 10, pp. 762-772, 1977.

[21] S. Wu, and U. Manber, U. *A fast algorithm for multi-pattern searching Tucson*, AZ: University of Arizona, Department of Computer Science, 1994, pp. 1-11.

[22] D. Luchaup, L. De Carli, S. Jha, and E. Bach, "Deep packet inspection with DFA-trees and parametrized language overapproximation," *presented at IEEE INFOCOM 2014-IEEE Conference on Computer Communications*, pp. 531-539, IEEE, 2014.

[23] M. Češka, V. Havlena, L. Holík, O. Lengál, and T. Vojnar, "Approximate reduction of finite automata for high-speed network intrusion detection," *presented at* I*nternational Journal on Software Tools for Technology Transfer*, vol. 22, no. 5, pp. 523-539, 2020.

[24] M. Ceška, V. Havlena, L. Holík, J. Korenek, O. Lengál, D. Matoušek, j. Matoušek, J. Semric, and T. Vojnar, "Deep packet inspection in FPGAs via approximate nondeterministic automata," *presented at 2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 109-117, IEEE, 2019.

[25] M. Roesch, "Snort: Lightweight intrusion detection for networks," *In Lisa*, vol. 99, no. 1, pp. 229-238, 1991.

[26] R. Sommer, "Bro: An open source network intrusion detection system," *Security, E-learning, E-Services, 17. DFN-Arbeitstagung über Kommunikationsnetze*, 2003.

[27] Cisco. (2022, june 6). *Cisco IOS Intrusion Prevention System (IPS)* [Online]. Available: https://www.cisco.com/c/en/us/products/security/ios intrusion-prevention-system-ips/index.html

[28] C. Yin, H. Wang, X. Yin, R. Sun, J. Wang, "Improved deep packet inspection in data stream detection," *The Journal of Supercomputing*, vol. 75, no. 8, pp. 4295-4308, 2019.

[29] R. Sun, L. Shi, C. Yin, J. Wang, "An improved method in deep packet inspection based on regular expression," *The Journal of Supercomputing*, vol. 75, no. 6, 3317-3333, 2019.

[30] S. Nagaraju, B. Shanmugham, and K. Baskaran, *High throughput token driven FSM based regex pattern matching for network intrusion detection system*, Materials Today: Proceedings, vol. 47, pp. 139-143, 2021.

[31] X. Yu, W. C. Feng, D. Yao, and M. Becchi, "O 3 FA: A scalable finite automata-based pattern-matching engine for out-of-order deep packet inspection," *presented at 2016 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, pp. 1-11, IEEE.

[32] A. V. Aho, and M. J. Corasick, "Efficient string matching: an aid to bibliographic search," *Communications of the ACM*, vol. 18, no. 6, 333-340, 1975.

[33] M. Norton, "Optimizing pattern matching for intrusion detection," Sourcefire, Inc., Columbia, MD, 2004.

[34] N. Tuck, T. Sherwood, B. Calder, and G. Varghese, "Deterministic memory-efficient string matching algorithms for intrusion detection," *In IEEE INFOCOM 2004*, vol. 4, pp. 2628-2639, IEEE.

[35] L. Tan, T. Sherwood, "A high throughput string matching architecture for intrusion detection and prevention," *presented at 32nd International Symposium on Computer Architecture (ISCA'05)*, pp. 112-122, IEEE, 2005.

[36] T. H. Lee, and N. L. Huang, "A pattern-matching scheme with high throughput performance and low memory requirement," *IEEE/ACM Transactions on Networking*, vol. 21, no. 4, 1104-1116, 2012.

[37] H. Kim, "A scalable architecture for reducing power consumption in pipelined deep packet inspection system", *Microelectronics Journal*, vol. 46, no. 10, 950-955, 2015.

[38] R. Padmashani, S. Sathyadevan, and D. Dath, "BSnort IPS better snort intrusion detection/prevention system," *presented at 2012 12th International Conference on Intelligent Systems Design and Applications (ISDA)*, pp. 46-51, IEEE, 2012.

[39] S. Gupta, "Efficient malicious domain detection using word segmentation and BM pattern matching," *presented at 2016 International Conference on Recent Advances and Innovations in Engineering (ICRAIE)*, pp. 1-6, IEEE, 2016.

[40] T. F. A. Rahman, A. G. Buja, K. Abd, and F. M. Ali, "SQL Injection Attack Scanner Using Boyer-Moore String Matching Algorithm," *J. Comput.*, vol. 12, no. 2, 183-189, 2017.

[41] Y. Otoum, and A. Nayak, "As-ids: Anomaly and signature based ids for the internet of things," *Journal of Network and Systems Management*, vol. 29, no. 3, pp. 1-26, 2021.

[42] Y. Wang, and H. Kobayashi, "An improved technology for content matching intrusion detection system," *presented at 2006 International Conference on Software in Telecommunications and Computer Networks*, pp. 238-241, IEEE, 2006.

[43] A. A. Hasan, and N. A. A. Rashid, "Hash-Boyer-Moore-Horspool string matching algorithm for intrusion detection system," *presented at In International Conference on Computer Networks and Communication Systems*, vol. 35, pp. 12-16, 2012.

[44] S. Sharma, and M. Dixit, "Single Digit Hash Boyer Moore Horspool Pattern Matching Algorithm for Intrusion Detection System," *presented at International Journal of Future Generation Communication and Networking*, vol. 9, no. 9, 169-180, 2016.

[45] Q. Zheng, "An improved multiple patterns matching algorithm for intrusion detection," *presented at 2010 IEEE International Conference on Intelligent Computing and Intelligent Systems*, vol. 2, pp. 124-127, IEEE, 2010.

[46] C. Ke-Qin, D. Lin, and W. Hui, "An improved multi-pattern matching algorithms in intrusion detection," *presented at 2013 Fifth International Conference on Measuring Technology and Mechatronics Automation*, pp. 203-205, IEEE, 2013.

[47] M. Aldwairi, K. Al-Khamaiseh, F. Alharbi, and B. Shah, "Bloom filters optimized Wu-Manber for intrusion detection," *Journal of Digital Forensics, Security and Law*, vol. 11, no. 4, 2016.

[48] B. Zhang, X. Chen, Pan, and Z. Wu, "High concurrence Wu-Manber multiple patterns matching algorithm," *presented at the 2009 International Symposium on Information Processing (ISIP 2009)*, pp. 404, 2009.

[49] Cisco. (2022, june 6). SNORT [Online]. Available: https://snort.org/