Detecting SYN Flood Attack Using CSA-nets

Mohammed Alahmadi

1 Department of Software Engineering, College of Computer Science and Engineering
University of Jeddah, Jeddah 21493, Saudi Arabia
2 School of Computing, Newcastle University Science Square, Newcastle upon Tyne,
NE4 5TG, United Kingdom

Abstract. Distributed Denial of Service (DDoS) attacks pose a persistent threat to network security by interrupting server functions. One common DDoS attack is the SYN-flood attack, which targets the three-way handshake process of TCP protocol. This technique overwhelms a system by sending a vast number of SYN messages, thereby exhausting its computational and communicative resources. A visual simulation for this scenario offers deeper insights into the intricacies of the TCP-SYN-flood attack. This paper presents a novel approach that combines TCP protocol anomaly detection with visual analysis through Communication Structured Acyclic nets (CSA-nets). The strategy provides a clear visualisation of attack behaviours, granting a deeper understanding of DDoS patterns and their underlying causes. A new concept of TCCSA-nets is introduced. TCCSA-nets allow elaborating on the system's performance and emphasizing the system's operations in real-time. This approach allows for the classification of messages as abnormal if their duration exceeds a predetermined time limit. Messages within this time frame are considered normal communication. The effectiveness of this approach was tested on public datasets, demonstrating its capability in detecting SYN-flood attacks.

Keywords: Formal model, modelling, visualising, analysing, cybersecurity, protocols, threshold detection.

1 Introduction

Communication Structured Acyclic Nets (CSA-nets) [1, 2, 3, 4, 5] is a formal and mathematical model designed to analyse and visualise complex evolving systems (CES). It aids in planning, bug testing, and monitoring system efficiency. CSA-nets depict CES as distinct, acyclic nets connected by buffer places. These buffer places facilitate connections between subsystem components through both synchronous and asynchronous communications. CSA-nets embed numerous properties in their formalisation, such as causality, concurrency, and synchronisation, which can be invaluable in analysing protocols. Furthermore, they can model and analyse network communication protocols, aiding in the detection and understanding of abnormal network behaviours. The application of CSA-nets for modelling network protocols and detecting abnormalities during connections remains underutilised. However, this mathematical model could assist researchers and practitioners in discerning the root causes of such anomalies, integrating the study and modelling of these behaviours with detection methodologies.

Distributed Denial of Service (DDoS) attacks disrupt regular internet traffic by overwhelming servers with vast amounts of data, leading to resource depletion [6]. The diverse nature of malicious packets in these attacks complicates their analysis [7]. Although DDoS attacks frequently target the network layer, they can also exploit other layers using methods such as ICMP, SYN and UDP flooding [8]. Attackers can manipulate the network layer by altering the IP packet header, flooding servers with irregular packets. Tracing the origin of these attacks is challenging due to the widespread use of IP spoofing. As a result, attackers leverage attributes like packet size, rate, bit rate, and arrival time to drain server resources [9]. The Transmission Control Protocol (TCP) an essential internet protocol initiates connections through a three-way handshake involving SYN and ACK messages [10].

TCP is widely used across the internet for various services, leveraging flags such as SYN, ACK, and RST to manage connection status and data transport. SYN-flooding attacks can be direct or involve IP address spoofing, with the most potent form being a distributed attack utilizing multiple

David C. Wyld et al. (Eds): SEAPP, Fuzzy, SIGEM, CSEA, MLTEC -2023
pp. 01-15, 2023. CS & IT - CSCP 2023 DOI: 10.5121/csit.2023.132301

zombie computers to inundate the target. Attackers can exploit this by continuously sending SYN packets with fake IP addresses, leading to network saturation and server unresponsiveness [11]. A TCP-SYN-flood attack, a subtype of DDoS attack, capitalizes on the TCP handshake process to overwhelm the targeted server, rendering it inoperable. In this assault, TCP connection requests flood in faster than the server can process, inducing network congestion. Such an attack not only disrupts server services but also disrupts and compromises the security of the client-server communication channel. It is hard to detect and can arise without prior security alerts. Moreover, it is seen as permissible since it does not directly exploit network vulnerabilities or misuse resources [12].

This paper introduces a novel method for analysing and visualising cybersecurity behaviours using CSA-nets. It models clients and servers as acyclic nets, emphasising their communication via the three-way handshake. A new algorithm is introduced to discern communication between these nets. Essentially, the algorithm tracks packets that complete the communication sequence successfully, identifying any abnormal packets that fail in the process. By using the capabilities of CSA-nets, this approach offers deep insights into anomalous TCP activity, facilitating the detection of malicious activities and elucidating their root causes in the interactions between clients and servers.

2 Related work

A Distributed Denial of Service (DDoS) attack is a widespread network assault aimed at overwhelming computational resources and bandwidth, hindering the ability of legitimate users to access services. This attack typically involves substantial packet flooding, making it a more extensive version of a denial of service (DoS) attack. Over the years, numerous techniques for detecting and mitigating DDoS attacks have been developed, such as those targeting TCP flood attacks. However, these methods often prove insufficient, as they are based on the number of requests from a single source to the target server, failing to identify attacks from a singular link. Moreover, attackers can effortlessly change their IP addresses, thus bypassing blacklisting.

However, different techniques and approaches are used to detect and prevent such kinds of attacks, including machine learning (ML) and formal methods. In this context, we shed light on both fields in detecting abnormalities in TCP. Regarding ML, researchers [13] proposed a machine learning approach to identify DDoS attacks. This approach involves two main steps: feature extraction and model detection. The feature extraction process serves to remove superfluous features, isolating the most critical features of DDoS attack traffic. These features are then used as inputs in the model detection phase, which uses the random forest algorithm to train the attack detection model. Their experimental results suggest that the ML-based DDoS attack detection method yields a high detection rate for common DDoS attacks. In [14] argued that DDoS attacks from local networks pose greater threats than external ones due to detection complexities. Using a spacecraft simulator's real-time telemetry software, they analyzed both benign and malicious packets. Their detection method was designed for TCP and HTTP Flood DDoS attacks. They observed that the PSH&ACK flags, initially set low during regular data transfers, surged during attacks. Consequently, they introduced two algorithms. The primary one detects TCP floods by counting the PSH&ACK flags. Then, if this count surpasses a predefined limit in a specific duration, it indicates an attack. A recent study [15] investigated machine learning (ML) algorithms for real-time DDoS attack detection, addressing prevalent attack types like UDP flood, ICMP ping flood and TCP-SYN-flood. A classification model was crafted based on ML, trained to distinguish benign from malicious network traffic. Decision Tree (DT), Random Forest (RF), and K-Nearest Neighbors (KNN) algorithms were found to be effective in detecting attack traffic. However, the KNN approach was resource-demanding due to its distance calculation mechanism, leading to potential detection lags. In contrast, DT emerged as a more streamlined option, making it the favored classification model. A study [16] centered on detecting DoS attacks within the transmission control protocol (TCP) three-way handshake. The researchers introduced a detection and prevention

mechanism for the TCP-SYN-flood attack through the use of an adaptive threshold. This threshold was calculated via the 'Adaptive threshold algorithm'. The outcomes highlight the proficiency of the proposed approach in identifying and preventing TCP-SYN-flood attacks by leveraging an adaptive thresholding technique. Moreover, a study [17] investigated ML and data mining algorithms for detecting DDoS attacks, particularly TCP-SYN-flood attacks, using the CAIDA dataset. The decision stump and the One-R (OR) algorithm were highlighted for their accuracy, with performance metrics supporting their effectiveness.

On the other side, in the field of modeling, several studies have been undertaken. One such study [18] introduced a Petri net-based model to differentiate between DDoS attacks and legitimate communications, both on the server and client sides. When deployed on the server side, this model endeavors to reduce time complexity. It aims to identify and discard malicious packets, while allowing valid communications. Packets are categorised using a confidence index, specifically designed to prioritise against untrusted network attacks. Legitimate communications receive high index values and are positioned in a high-priority queue, whereas malicious communications are allocated to a lower-priority queue. This strategy enhances the detection and filtering of malicious attacks, leading to improved server efficiency. In addition, Structured Occurrence Nets (SONs) are another mathematical and graphical modelling tool used to detect these types of behaviours. SONs were used to identify DNS tunneling attacks, a technique leveraged by adversaries to extract data from multiple accounts [19]. In this model, each packet is represented as an individual occurrence net. If a token securely transits from the start to the finish of the SON, the packet is recognized as legitimate. Conversely, if it does not, it is deemed indicative of a DNS attack.

However, despite the progress in machine learning, there is still a need for deeper insights into abnormal behaviors and the role of modeling techniques in detection. Modeling can assist investigators and decision-makers in understanding the causes and effects by visualizing such communication and providing insights into abnormal behaviors. Thus, combining the strengths of both methodologies can provide a more comprehensive and nuanced understanding of abnormal behaviours and enhance detection capabilities. By leveraging the predictive power of machine learning and the structural insights from modeling, researchers can achieve a collaborative effect, leading to more robust and reliable solutions.

3 Preliminaries

3.1 Communication Structured Acyclic Nets (CSA-NET)S

Communication Structured Acyclic Nets (CSA-NET)S [2, 20] are sets of occurrence nets linked with each other through unique elements. These unique elements are referred to as *buffer places*, which are capable of modeling both asynchronous and synchronous communication[1]. (CSA-NET)S are designed to capture information about either: (i) the interaction between actual/expected behaviors; or (ii) the collected evidence to be analyzed. (CSA-NET)S also have the advantage of representing different depictions of the actions of dynamic, evolving systems [1]. The benefit of (CSA-NET)S lies in their structure, which reduces complexity compared to analogous representations and provides a direct means of modeling emerging structures. Events serve as the means to link these ANs with each other through these buffer places. Graphically, Figure 1 represents an occurrence net that comprises three main components: Places (P), represented by circles; Events (E), depicted as rectangles; and Flow Relations (F), illustrated by direct arcs. For any specified node x, the collection of input nodes is represented by •x, and x• denotes the output nodes. That is, the initial places in an AN has no inputs nodes, whereas the final places do not have outputs nodes.

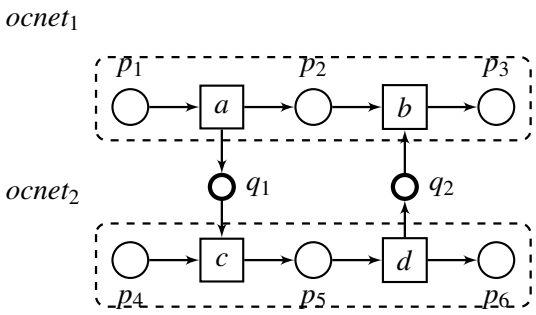

Fig. 1: Communication Structured Occurrence nets (CSO-NET).

3.2 Transmission Control Protocol (TCP)

The Transmission Control Protocol (TCP) is a protocol used to allow exchanging data between devices by transferring packets over the Internet and ensuring the reliable delivery of these packets. TCP is widely used in network communications that require data delivery where delivery errors are completely rejected. TCP creates a connection between the sender and the receiver before transmitting data and ensures remains active during the communication. This makes TCP connection-oriented protocol. For this task, TCP employs a strict strategy referred to as a three-way handshake. TCP also employs error detection strategies to ensure that the data has been received correctly. This includes setting the connection timeout duration, activating the checksum field, and receiving and sending acknowledgements. Moreover, it is a full duplex connection that is established using a well-known mechanism referred to as a three-way handshake. Generally, this mechanism allows both the sender and receiver to synchronize (SYN) and acknowledge (ACK) each other. As its name implies, it consists of three consecutive steps as follows. First, the client sends a SYN message to the receiver (server) requesting to connect. Second, the receiver responds with both SYN and ACK. The SYN in this case means that the server is ready to connect and the ACK confirms receiving the sender's previous SYN message. Finally, the client sends ACK to the server confirming receiving its previous message and the connection is now established.

SYN Flood Attack TCP faces several cyberattack as SYN flood. In particular, SYN flood attacks the three-way handshake mechanism during the establishment of a TCP connection. Technically, SYN flood is a type of DDoS that exploits the three-way handshake to consume the server's resources. It involves sending repeated SYN request packets to the server ports using fake IPs. These requests appear to be legitimate and the server is deceived and try to respond to these request (SYN- ACK) which, in turn, wastes its resources. The challenge lies in the difficulty of the TCP protocol in detecting such situations. This is because attackers typically follow the normal process of sending SYN requests to the server, which causes the server to wait and resulting in what is known as a 'half-open connection. This occurs when the third step of the three-Way handshake incuding sending final ACK to the server fails or if the host closes the connection without acknowledging the other.

SYN flood attack can be carried out in three ways, Direct SYN Flood Attack, SYN Spoofed Attack and DDoS SYN attack. In direct SYN, attackers sends a massive amount of SYN messages from the same source IP address, while in Spoofed SYN, it involves sending a massive amount of SYN messages from different Spoofed IP addresses. This is used to avoid being discovered. Moreover, in DDoS SYN attack which is the most dangerous attack, the victim server receives SYN packets simultaneously from several infected computers under the control of the attacker. This combination of hijacked machines is called a botnet.

4 Coloured communication structured acyclic nets (ccsa-nets)

A coloured communication structured acyclic net consists of several disjoint acyclic nets that can communicate through special buffer places. These buffer places allow instantaneous transfer of tokens and can involve a cycle only when synchronous communication is being implemented. For brevity, some technical definitions are omitted or described informally.

Definition 1 (Coloured Acyclic net). *A* coloured acyclic net *is a tuple* $acnet = (P, T, F, col, ex, gd)$, *where* P *and* T *are disjoint finite sets of* places *and* transitions *respectively, and* $F \subseteq (P \times T) \cup (T \times P)$ *is the* flow relation *such that:*

- P *is nonempty and* F *is acyclic.*
- *For every* $t \in T$, *there are* $p, q \in P$ *such that* pFt *and* tFq.
- col *is a mapping assigning nonempty finite set of colours to every place.*
- ex *is an arc expression function that assigns an arc expression to each arc* $x = (p,t)$ *or* $x = (t,p)$ *so that* $Type[ex(x)] = col(p)$.
- gd *is a mapping assigning a boolean guard to each transition. (In this case guards are very simple and ensure that all the input tokens used in transition firing are of the same colour and all the tokens produced have the same colour too.)*

Definition 2 (Coloured Communication Structured Acyclic net). *A* coloured communication structured acyclic net (or CCSA-net) *is a tuple*

$$ccsan = (cacnet_1, \ldots, cacnet_n, Q, W, col', ex') \quad (n \geq 1)$$

such that:

- $cacnet_1, \ldots, cacnet_n$ *are colured acyclic nets.*
- $csan = (acnet_1, \ldots, acnet_n, Q, W)$ *is a well-formed* CSA*-net (as defined in [5]), where each* $acnet_i$ *is an acyclic net obtained from* $cacnet_i$ *after deleting the last three components.*
- col' *is a mapping assigning nonempty finite set of colours to every buffer place in* Q.
- ex' *is an arc expression function that assigns an arc expression to each arc* $x = (q,t)$ *or* $x = (t,q)$, *where* $q \in Q$, *so that* $Type[ex(x)] = col(q)$.

The execution semantics of CCSA-nets follows the standard coloured net rules as well as the execution rules of CSA-nets [5] and is omitted in this paper. Markings of a CCSA-net assign sets of suitable coloured tokens to the places of *csan* as well as the buffer places. The execution rule is basically the same as for CSA-nets assuming that all the input tokens used in transition firing are of the same colour and all the tokens produced have the same colour too. Intuitively, this means that the executions corresponding to different colours do not interfere with each other (see Figures 4-10), and so coloured tokens ensure non-overlapping of interacting tokens (e.g., packets), allowing differentiation based on unique features to prevent interference [5]. In the CCSA-net case, a well-formed step sequence means that no place or buffer place is filled by the same coloured token more than once in any given step sequence. This mechanism allows for synchronising transitions from different acyclic nets (e.g., clients and server), which then aids in modelling the behaviours of the three-way handshake process. *ccsan* is 'well-formed' and so its executions (scenarios) allow to represent well-defined causal relationships and properties. These properties can assist in detecting and recording the entire causal history and so help in detecting abnormal behaviours that might occur during the three-way handshake process.

Type your text

Table 1: Detailed description of client and server acyclic nets

Content	meaning
c_1	Client is ready to initiate a connection SYN
snd-SYN	Client sends request to server to establish connection
c_2	Client waits for response from Server
rcv-SYN-ACK	Client receives response from Server ACK
c_3	Client is ready to send SYN/ACK to Server
snd-ACK	Client sends SYN/ACK to Server
c_4	Client is ready to establish connection and push data to Server
s_1	Server is ready to initiate connection with Client
rcv-SYN	Server receives request from Client to establish connection
s_2	Server is ready to send ACK to Client
snd-SYN-ACK	Server sends ACK to Client
s_3	Server waits for response from Client
rcv-ACK	Server receives SYN/ACK from Client
s_4	Server is ready to establish connection and push data to Client
x	represent the expression on the arc, which can be a function or operator.

5 Analysing three-way handshake by CCSA-net

The CCSA-net visualisation framework is designed to aid in the analysis of cybercrime investigations. It uses acyclic nets to represent different subsystems as individual acyclic nets and connects them through buffer places to model both asynchronous and synchronous communication between subsystems. In this section, we will analyse the three-way handshake using CCSA-net representation.

5.1 Structure of TCP model

Structurally, the three-way handshake process can be represented by two different acyclic nets linked by buffer places. Figure 2 demonstrates the three-way handshake process between clients and the server using a CCSA-net, which contains separate acyclic nets representing clients (the upper one) and the server (the lower one). The places/states are represented by circles, showing the current status of the process (e.g., SYN, SYN-ACK or ACK), while transitions — represented by squares — are responsible for transferring tokens from one place/states to another. Moreover, buffer places q_1, q_2 and q_3 — represented by bold circles — are responsible for transferring tokens between different acyclic nets synchronously or asynchronously. Expressions can be displayed on arcs (e.g., x and y in Figure 1), and these expressions can be functions or operators written in ML (a programming language). This adds additional constraints on the arcs, e.g., determining the amount of traffic the server can process at a time or the time limit the server waits for an acknowledgment as we used in classifier model in Figure 11. Table 1 provides a detailed description of the Client and Server nodes. Initially, tokens representing different clients can be placed in c_1 and be moved by firing transitions (from place to another place by firing a transition). Each token has different colour to distinguish between different clients during communication. Table 2 gives detailed description of the features of each token. We can analyse the handshake process based on the model's structural properties to ensure its soundness. The CCSA-net structural properties such as causality can aid our understanding of how packets are transmitted between Client and Server through three-way handshake communication. This can provide administrators and investigators with insights into the causes of DDoS attacks and help them understand how to prevent future attacks. This mechanism

allows recording the causality between executed transitions which helps in detecting and tracing the process of such communication. The investigators can benefit from extracting the causality of fired transitions to visually understand abnormal behaviours during the connection before delving into the cause and effect of such behaviour. That is, the three-way handshake can be effectively represented (structurally) using a properly structured CCSA-net.

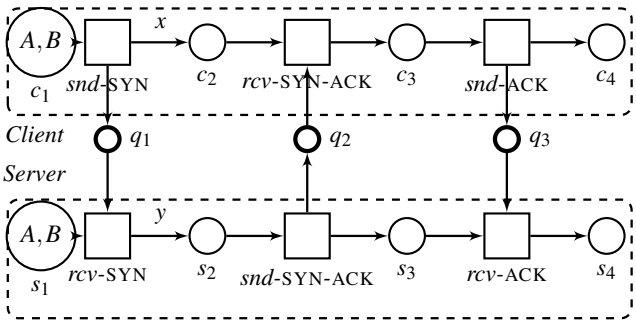

Fig. 2: CCSA-net model for communication between clients and server showing two clients (tokens) ready to establish connection with server.

5.2 Behaviour of TCP model

To analyse the three-way handshake behaviourally, we will leverage the built-in features and formal properties inherent in the semantics of CCSA-nets [5]. These can be used to analyse communication behaviour and detect abnormal situations that might arise during a connection. Such properties can be discovered by the reachability analysis, which verifies the traceability of tokens from the initial marking, and well-formedness, guaranteeing a clear representation of causality. Both these mechanisms allow us to model and trace the standard behaviours of the TCP protocol and to pinpoint abnormal behaviours, such as SYN-flood attacks. Note that well-formedness is a fundamental consistency criterion for CSA-nets and also CCSA-nets, and it essentially guarantees a clear representation of causality in the behaviours of the net. The CSA-net in Figure 2 is well-formed and so it ensures a clear representation for each token during the connection, assisting in detecting specific abnormal behaviours in communication between clients and servers.

5.3 Reachability

Reachability analysis is a verification technique used to determine, e.g., whether a specific place can be reached by following a sequence of steps from the initial marking. This property is essential for examining the behavioural characteristics of a system. Essentially, reachability concerns the system's capability to transition from its initial marking to any designated state. Figure 3 presents the reachability graph for the CCSA-net model illustrating a three-way handshake between Client and Server. The graph also shows Client executes all potential actions progressing

Table 2: Detailed description of colour sets of token

colour set	description
IP-source	contains the IP address of the source machine
IP-destination	contains the IP address of the destination machine
source-port	contains the source port number of the source machine
destination-port	contains the destination port number of the destination machine

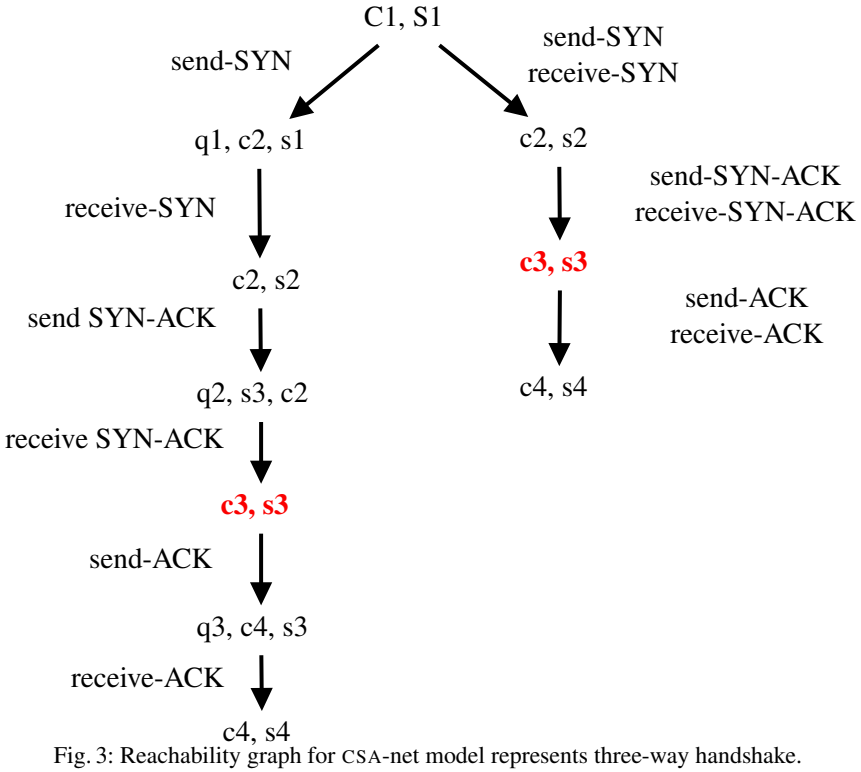

Fig. 3: Reachability graph for CSA-net model represents three-way handshake.

through various steps, asynchronous communication on the left and synchronous communication on the right. Specifically, it demonstrates that all places are accessible from the initial marking. Thus, the reachability analysis confirms that the proposed model guarantees that all places are accessible from the initial marking. Moreover, the reachability analysis helps to understand system behaviour and trace potential steps, especially within the context of larger systems.

6 TCP three-way handshake and CCSA-nets

In the previous section, we analysed three-way handshake based on the structure and behavioural properties of CSA-nets, discussing certain properties of this model. This section will demonstrate and utilize the behaviours (steps) of three-way handshake by using CCSA-net to detect abnormal behaviours (SYN-flood).

6.1 Normal behaviour

The TCP protocol employs a three-way handshake to establish a reliable connection between two devices. This procedure is captured in three principal steps using the CCSA-net. As depicted in Figure 2, tokens A and B reside at c_1. For example, token A will be consumed during the *snd*-SYN event, the client sends a SYN token, transmitting token A to c_2 and q_1, as illustrated in Figure 4. Upon receiving this, the serve's *rcv*-SYN action is activated, transferring token A from s_1 to s_2 (as shown in Figure 5). This progression signifies the client's anticipation of the server's response to make the initial SYN communication. In the second handshake phase, with token A positioned at s_2, the SYN-ACK action is triggered, sending token A to s_3 and q_2 (as shown in Figure 6). Subsequently, the client acknowledges the server's SYN-ACK action through the *rcv*-SYN-ACK event, moving token A to c_3, as captured in Figure 7. In the final handshake step, the client responds via the *snd*-ACK action, shifting token A from c_3 to c_4 and q_3 (as represented in Figure 8). The server's subsequent *rcv*-ACK action in Figure 9 finalises the handshake, ensuring an established connection between both entities.Note that both tokens A and B can be moved at the same time without interfering with each other. This is because CCSA-net can consume more than

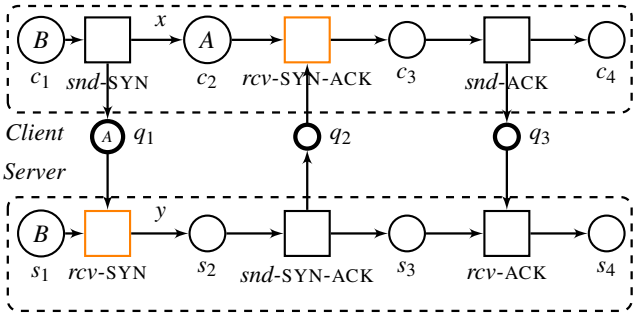

Fig. 4: Server received client's request prepares to respond to this request.

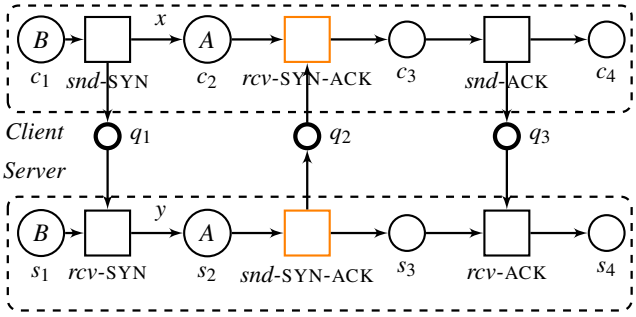

Fig. 5: Second handshake SYN-ACK using *snd*-SYN-ACK and *rcv*-SYN-ACK.

one token simultaneously. That is, we rely on the features inherent in the semantics of CCSA-net to model and analyze such communication between servers and clients.

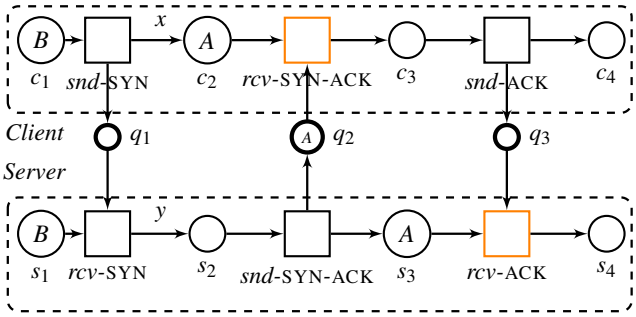

Fig. 6: Client received server's request and preparing to response for server's request.

6.2 Abnormal behaviour

In the abnormal behaviour of the SYN-flood attack, attackers manipulate the standard behaviour to inundate a server, thereby making detection increasingly challenging. As depicted in Figure 10, the server fails to receive the third handshake response (ACK) from the client. Even though the *snd*-ACK acction is enabled, token A remains frozen at c_3 and s_3. This strategy overwhelms the server with a surge of SYN requests, leading to numerous half-open connections and consequently making the server unresponsive.

7 Timed CCSA-nets

This section introduces the method for integrating timing information into CCSA-nets. This integration allows TCCSA-nets to elaborate on the system's performance, capturing the system's

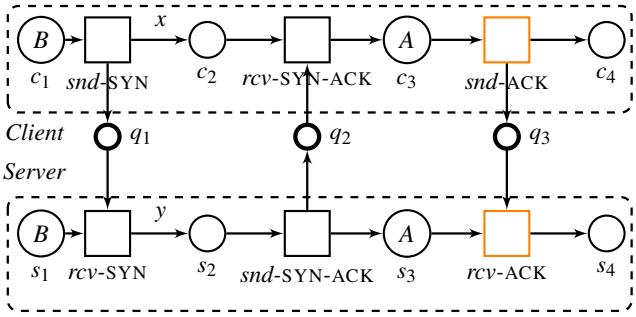

Fig. 7: Connection for third-handshake using *snd*-ACK and *rcv*-ACK events.

operations in real-time. Moreover, with this timing feature, CCSA-nets become suitable for modeling systems where correctness depends on event timing. In contrast to CCSA-nets, this timed variant offers deeper insights into behaviours, such as the average system runtime and adherence to deadlines for real-time processes. The primary distinction between timed and untimed CCSA-nets lies in their token structure as tokens now include additional values denoting time (TCCSA-net model incorporates a global time that captures the model's execution duration). This inclusion of time can aid in the evolution of the TCCSA-net model being studied. The token time quantifies the execution duration attributed to the token itself. Figure 11 provides an illustrative example of a TCCSA-net. The timestamp for each token in the initial markings is set to zero. Token time is denoted using the @ symbol. For instance, @0 indicates that the token time is zero, signifying that the token remains in its initial marking. The transition $send-\text{SYN}$ is active and ready to fire. Once the event $send-\text{SYN}$ is activated, the token time commences and begins to increment. The formal notation for TCCSA-nets is provided in Definition 3.

Definition 3 (Timed Coloured Communication Structured Acyclic net). *A* timed coloured communication structured acyclic net (or TCCSA-net) *is a tuple*

$$ccsan = (cacnet_1, \ldots, cacnet_n, Q, W, col', ex') \quad (n \geq 1)$$

such that:

- $cacnet_1, \ldots, cacnet_n$ *are coloured acyclic nets.*
- $csan = (acnet_1, \ldots, acnet_n, Q, W)$ *is a well-formed* CSA*-net (as defined in [5]), where each* $acnet_i$ *is an acyclic net obtained from* $cacnet_i$ *after deleting the last three components.*
- col' *is a mapping assigning nonempty finite set of colours to every buffer place in* Q*. Each colour set can be timed or untimed.*
- ex' *is an arc expression function that assigns an arc expression to each arc* $x = (q,t)$ *or* $x = (t,q)$*, where* $q \in Q$*, so that* $Type[ex(x)] = col(q)$*.*

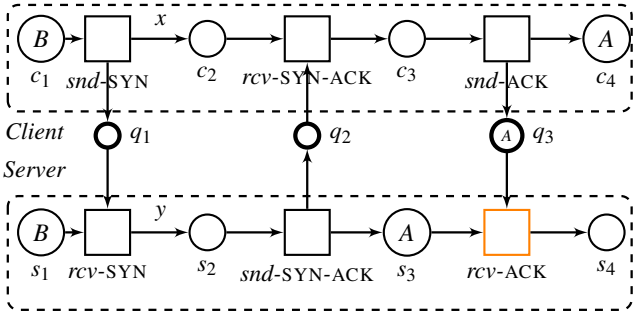

Fig. 8: Server received client's request and preparing to start reliable connection with client.

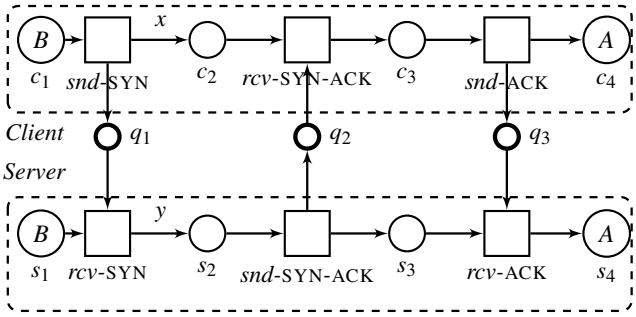

Fig. 9: Both client and server are ready to push data between each other.

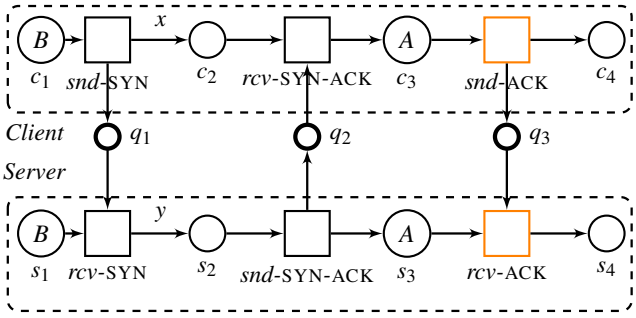

Fig. 10: Abnormal behaviour as token A is frozen in c_3.

That is, Definition 1 is extended by introducing a timed colour set for the token, enabling it to carry a time value during execution. The colour set can be either timed or untimed. All places associated with a timed colour set are termed 'timed places', and the arcs connected to these places are named accordingly. These arcs can carry an expression function or operator, expressed in ML (programming language). This introduces further constraints on the arcs. In ML, transitions might also feature a time delay description, expressed as a type TIME. These concepts are derived from the domain of coloured Petri nets. Thus, the colour set of the token **t** can be described as a vector composed of six types, detailed as follows:

$$(IP_{src}, IP_{dst}, Port_{src}, Port_{dst}, Flag, T_t).$$

Time values are continuously updated during the model's execution (i.e., transition firing). At any given point, we can evaluate the status of a token by inspecting its values. Once the token's execution is finished, the value T_t represents the duration the token has spent within the model (e.g., completing a three-way handshake). Algorithm 1 summarises the concept of initialising timing for CCSA-nets tokens. Intuitively, time increments with the firing of a token at each place, denoted as @++, which represents the duration taken for the token to transition from its current place to the next. Delays, however, can occur within the process. For example, a server might take 4 seconds to respond to a client request. Moreover, we allow events/transitions to incorporate delay expressions, represented as expressions of the TIME type. Essentially, the cumulative time within the model includes all timestamps resulting from the evaluation of the time delay inscription of the transition. For instance, if the initial event (*snd-syn*) occurs at a timestamp of zero (the initial marking at the beginning of the model's execution), and the time specified for this event is set to 5 seconds (e.g., the duration needed to delay or consume the token in the *snd-syn* event), it indicates that the token takes (0 + 5) seconds to transition to the subsequent place (*rcv-syn*). Consequently, we compute the cumulative time throughout each token's execution to capture its behaviour. Our aim is to identify any anomalies by detecting tokens that exceed the threshold specified for each

communication. In the TCCSA-net model, the delay() function signifies the additional, *unknown* duration that events might consume. Type your text

Algorithm 1: TCCSA-nets

Input : CCSA-net
Output: TCCSA-net

Initialization
Initialize the model global time to zero
for ***each token*** i ***in the initial marking*** **do**
    Initialize the token colour set and set $T[i]$ to zero
end
Start the model execution and begin incrementing the global timing
do
    for ***each token in the initial marking*** **do**
        Once the token is fired in the first event
        Start the token time $T[i]$
        do
            Increment $T[i]$ by 1 time unit //$T[i]++$
        while *Token did not reach the final marking*;
    end
while CCSA-*net is executing*;

8 Classification and frozen tokens

The extension of TCCSA-net proves advantageous for identifying unusual behaviours, such as those exhibited in SYN-flood attacks. It uses the timing attribute of tokens to detect abnormal patterns. Specifically, we rely on a predefined threshold to classify tokens. These tokens are identified using the flow ID/colour set, which includes IP source, IP destination, Source port, Destination port, Flags, and Timestamp. The algorithm assesses each token/packet independently, determining whether it is indicative of abnormal behaviour or a standard communication. For instance, as illustrated in Figure 11, tokens can be classified as abnormal if their time values exceed the predefined threshold set for the classifier place.

Formally, in TCCSA-nets, transitions can only be enabled if their input tokens are present in the preceding places. We rely on their properties, combined with time, to detect any abnormal activities that might occur during communication. For example, the classifier will send the token to *rcv*-ACK when the server acknowledges it. In other words, the *rcv*-ACK event on the server side will only be enabled when all its inputs are in place to fire *rcv*-ACK and complete the three-way handshake. If not, an *alarm* event is triggered if the token's time exceeds the threshold, indicating a potential SYN-flood attack. The threshold, denoted as τ, is assumed to be 30 seconds, which represents the maximum duration the server will wait to receive an *rcv-ack* from the client, completing the three-way handshake. It is important to note that the server's waiting duration can vary based on the criteria set by the network administrator. The procedure for identifying abnormal behaviour is depicted in Algorithm 2. The algorithm starts by defining the maximum allowable time τ. Every communication is portrayed as a token. The server's wait time is computed using Eq. (1). The status $S(i)$ of token i at the 'classifier' place is expressed as:

$$S(i) = \tau - (T(i) + C(i)) \tag{1}$$

where τ is a predefined threshold representing the maximum allowable time the server waits to *rcv-ack* from the client, $C(i)$ denotes the time taken by the server to process the token, and $T(i)$ signifies the total time the token spends inside the model, which encompasses the cumulative time for sending and receiving both SYN and SYN-ACK events. Specifically, this equation evaluates the token's time relative to the maximum allowable duration for the process (the three-way handshake, in our case). In other words, it computes the difference between the threshold and the combined time needed to complete all events inside the model (sending and receiving SYN and SYN-ACK), as well as the time the server takes to process the token received from the client. Note that this approach can be further extended and refined by examining various communication statuses and packet properties, such as response averages, traffic bandwidths, or server capacity, which is a topic for future work and could improve detection of such behaviors. In our case, tokens will be classified as:

$$\text{Token} = \begin{cases} Normal & if\ S \geq 0 \\ Abnormal & if\ S < 0 \end{cases}$$

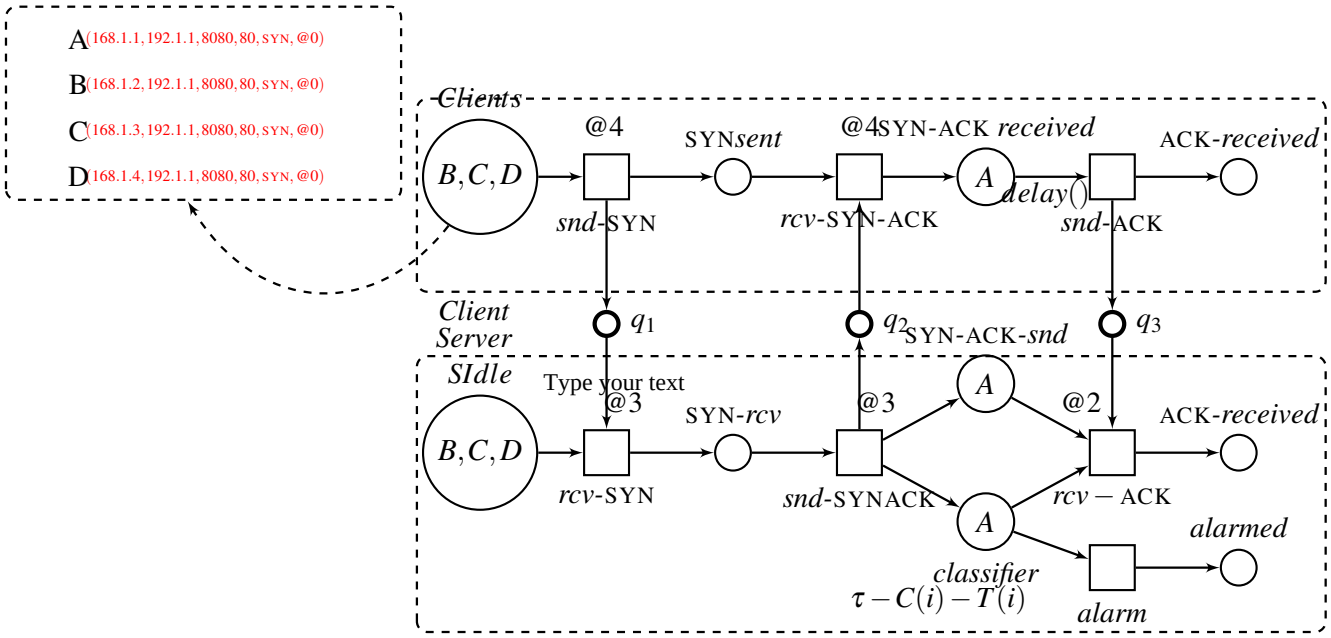

Fig. 11: CSA-net model after applying classification for normal and abnormal behaviours.

That is, detecting a SYN-flood attack can be modeled as a classification problem that differentiates between the network flow states of attack' and normal' [15]. In this context, we rely on our new TCCSA-net extension to discern and differentiate normal from abnormal behaviors. Specifically, servers operate within designated time frames to process client requests, using these durations to distinguish normal requests from abnormal ones. Through markings, tokens operating within the timeframe established by server administrators are considered normal. Consider Figure 11. Assume that the cumulative time required for a token inside the model to finalize the first and second handshake is the aggregate of all event times: @4 + @3 + @3 + @4 seconds, totaling 14 seconds. Furthermore, the server requires @2 seconds set on the *rcv*-ACK event to process the token, with a predefined threshold set at 30 seconds. As per Eq.(1), the token status is computed as $S(i) = 30 - (2 + 14)$. Since this is positive, the token is categorized as normal and proceeds to the *rcv*-ACK event. It is crucial to understand that this approach is employed to demonstrate that

TCCSA-net can be a promising tool to analyze and visualize the three-way handshake in order to detect abnormal activities that could occur during communication.

Algorithm 2: DDoS attack detection

Input : N tokens from m clients representing communications packets
τ //Time threshold
Output: Classification of token either normal or abnormal

Start
integer $i := 1$
do
  if *token i reached classifier condition* **then**
    Compute the token status $S(i)$ as per Eq. (1)
    if $S(i) \leq 0$ **then**
      Token is classified as normal
    end
    else
      Token is classified as abnormal
    end
  end
  $i := i + 1$
while $i \leq N$;

9 Experiment and Results

The proposed algorithm was tested using Python on a Mac PC equipped with an Intel Core i7 CPU and 16 GB of memory. The code is publicly available at [21]. We utilized the CICIDS 2019 dataset [22] comprising over a million tokens, each representing a packet described by 27 conditional features. These tokens were categorized into two groups: normal and abnormal. The algorithm's performance was assessed through a confusion matrix, which displayed an impressive accuracy rate of 89%. The high True Positive (TP) and True Negative (TN) rates contrasted with a low False Negative (FN) rate and zero False Positives (FP). However, an error rate of 11% was noted, primarily due to missing values in the data. After preprocessing to address these missing values, the error rate decreased by 1%, leaving a residual 10% error, which could be attributed to other unaddressed TCP flags. Our approach uniquely integrates the analysis and visualisation of client-server communication with the detection of anomalous packets. This allows network administrators to gain insights into potential DDoS attacks, aiding in the formulation of future prevention measures.

10 Conclusions

This paper presented a novel approach for modeling SYN-flood TCP DDoS attacks using TCCSA-nets. The approach includes a detection algorithm that distinguishes between normal and attack communications, with a specific focus on TCP-SYN-flood flag attacks. Our approach extends the advantages of CSA-net's concept through the use of the timing feature, which enables the algorithm to determine whether a packet is normal or not. One of the noteworthy features of this approach is its ability to provide and visualize detailed information on TCP DDoS attacks, which can be utilized further to prevent the attack. The algorithm was tested on publicly available data, and the results were impressive. Specifically, the algorithm achieved a 90% discrimination accuracy between normal and attack communications. Moving forward, we aim to improve this approach

by considering additional possible scenarios that cause TCP attacks. Moreover, we plan to enhance this model to operate in real-time situations for real-time detection. An add-in version of the algorithm will also be developed to enable easy deployment and autonomous operation.

References

[1] Koutny, M., Randell, B.: Structured occurrence nets: A formalism for aiding system failure prevention and analysis techniques. Fundamenta Informaticae 97(1-2), 41–91 (2009)

[2] Randell, B.: Occurrence nets then and now: the path to structured occurrence nets. In: International Conference on Application and Theory of Petri Nets and Concurrency. pp. 1–16. Springer (2011)

[3] Almutairi, N., Koutny, M.: Verification of communication structured acyclic nets using SAT. CEUR Workshop Proceedings, vol. 2907, pp. 175–194. CEUR-WS.org (2021)

[4] Alshammari, T.: Towards Automatic Extraction of Events for SON Modelling. CEUR Workshop Proceedings 3170, 188–201 (2022)

[5] Alahmadi, M.: Parameterised CSA-nets (2023)

[6] Kumari, P., Jain, A.K.: A comprehensive study of DDoS attacks over iot network and their countermeasures. Computers & Security p. 103096 (2023)

[7] Lent, D.M.B., Novaes, M.P., Carvalho, L.F., Lloret, J., Rodrigues, J.J., Proença, M.L.: A gated recurrent unit deep learning model to detect and mitigate distributed denial of service and portscan attacks. IEEE Access 10, 73229–73242 (2022)

[8] Jangjou, M., Sohrabi, M.K.: A comprehensive survey on security challenges in different network layers in cloud computing. Archives of Computational Methods in Engineering pp. 1–22 (2022)

[9] Kautish, S., Reyana, A., Vidyarthi, A.: Sdmta: Attack detection and mitigation mechanism for DDoS vulnerabilities in hybrid cloud environment. IEEE Transactions on Industrial Informatics (2022)

[10] Bauer, S., Jaeger, B., Reimann, M., Fromm, J., Carle, G.: Towards the classification of TCP throughput changes. In: NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium. pp. 1–7. IEEE (2022)

[11] Alibrahim, T.S., Hendaoui, S.: DDoS attacks prevention in cloud computing through transport control protocol TCP using round-trip-time rtt. IJCSNS 22(1), 276 (2022)

[12] Toyeer-E-Ferdoush, Rahman, H., Hasan, M.: A convenient way to mitigate DDoS TCP-SYN-flood attack. Journal of Discrete Mathematical Sciences and Cryptography 25(7), 2069–2077 (2022)

[13] Pei, J., Chen, Y., Ji, W.: A DDoS attack detection method based on machine learning. In: Journal of Physics: Conference Series. vol. 1237, p. 032040. IOP Publishing (2019)

[14] Shaaban, A.R., Abdelwaness, E., Hussein, M.: TCP and http flood ddos attack analysis and detection for space ground network. In: 2019 IEEE International Conference on Vehicular Electronics and Safety (ICVES). pp. 1–6. IEEE (2019)

[15] Rajesh, S., Clement, M., SB, S., SH, A.S., Johnson, J.: Real-time DDoS attack detection based on machine learning algorithms. Available at SSRN 3974241 (2021)

[16] Ramkumar, B., Subbulakshmi, T.: Tcp syn flood attack detection and prevention system using adaptive thresholding method. In: ITM Web of Conferences. vol. 37, p. 01016. EDP Sciences (2021)

[17] Sumathi, S., Rajesh, R.: Comparative study on TCP-SYN-flood DDoS attack detection: A machine learning algorithm based approach. WSEAS Transactions on Systems and Control 16, 584–591 (2021)