SYN Flood DoS Detection System Using Time Dependent Finite Automata

Noura AlDossary, Sarah AlQahtani, Reem Alzaher and Atta-ur-Rahman
12220500187@iau.edu.sa 22220500185@iau.edu.sa
32220500186@iau.edu.sa aaurrahman@iau.edu.sa
Department of Computer Science (CS), College of Computer Science and Information Technology (CCSIT), Imam Abdulrahman Bin Faisal University, P.O. Box 1982, Dammam, 31441, Saudi Arabia

Abstract
Network intrusion refers to any unauthorized penetration or activity on a computer network. This upsets the confidentiality, integrity, and availability of the network system. One of the major threats to any system's availability is a Denial-of-Service (DoS) attack, which is intended to deny a legitimate user access to resources. Therefore, due to the complexity of DoS attacks, it is increasingly important to abstract and describe these attacks in a way that will be effectively detected. The automaton theory is used in this paper to implement a SYN Flood detection system based on Time-Dependent Finite Automata (TDFA).

Keywords:
Denial of Service (DoS); Finite Automata (FA); Time-Dependent Finite Automata (TDFA); SYN Flood; Intrusion Detection System

1. Introduction

In recent years, the services provided by many sectors such as medicine, education, banking, and transportation are being replaced gradually with network-based applications. Consequently, the availability of these services is critical. However, the Internet and its services are vulnerable to attackers who aim to breach its availability. One of the major threats to any system's availability is a Denial-of-Service (DoS) attack, which is intended to prevent a legitimate user from accessing resources and services [1]. The DoS has many types, each of which targets specific components of the network to achieve a common goal, which is breaching the availability. The SYN Flood attack is one of the common DoS attacks that targets the three-way TCP communication process by flooding the connection tables [2]. Ultimately, the service to legitimate clients is blocked and the server may even malfunction or crash [3]
Due to the DoS nature, it should be detected in a timely manner, as time plays an important factor in this attack. To detect DoS attacks, many network intrusion detection systems (NIDS) have been implemented using different mechanisms. Basically, NIDS analyzes the raw network data to discover any signs of possible attacks [4]. One possible mechanism is to build NIDS based on automaton theory. Finite Automata (FA) is an abstract way of representing computations and statues of computers [5], thus, it can be utilized in security field by tracing and recording the attacks [6]. Time Dependent Finite Automata (TDFA) is a special type of FA, which is concerned about the time factor. Since the DoS is a temporal correlative, the TDFA can be used to analyze the time series of network packets to detect any suspicious motion, change, or development in the network. Hence, make the proper and timely response. In this paper, the concept of TDFA is used to build the SYN Flood DoS Detection System.

The remaining part of this work is organized as follow: Section 2 contains a review of related literature. Section 3 contains background information about Deterministic Finite Automata (DFA), Time-Dependent Finite Automata (TDFA), Denial of Service (DoS) attacks, and finally how to represent DoS using TDFA. Section 4 explains the proposed system architecture and its components, while Section 6 contains the conclusion and recommendations emanating from this work.

2. Literature Review

Recently, applying the concept of Finite Automata (FA) in the information security field has been proven to perform well in detection systems. However, this concept lacks the interest of researchers and professionals. Here are some related papers that focus on applying FA in the security sector. Authors in [7] have built a network-based intrusion detection system (NIDS) to detect Denial of Service (DoS) attack. This system is a misuse detection system, and

Manuscript received June 5, 2023
Manuscript revised June 20, 2023
https://doi.org/10.22937/IJCSNS.2023.23.6.18

it is designed based on the Time Deterministic Finite Automata (TDFA) approach. The authors stated that the system can recognize seven DoS attacks in which their signatures are known. For testing purposes, they have used eight datasets to detect only five attacks. In conclusion, the system has shown a good result in detecting all attacks except SYN Flood. A similar study [6] has proposed a misuse detection model using Adaptive Time-dependent Finite Automata (ATFA) to recognize the attacks' signatures. The model preprocesses the network data and extracts the temporal sequence that is used later by ATFA. Eventually, the model performs well in detecting DoS and Probing attacks.

In addition, a comprehensive study [8] has discussed different pattern matching techniques using Finite Automata (FA) in NIDS. They are compared based on various parameters, such as the used FA, whether it is deterministic or not, space complexity, time complexity, etc. Also, a possible solution has been demonstrated to overcome memory and time inefficiency. Also, authors in [9] have worked on creating a pattern-matching engine called O3FA. This engine is based on deterministic finite automata combined with suffix and prefix FA. The main purpose of this engine is to be able to detect the packets before reordering them. This makes the system invulnerable to DoS attack.

In [10] has described various network attacks using Deterministic Finite Automata (DFA). Authors have represented these attacks using state transition diagram to detect whether the system status is normal or not. Basically, the DFA model is used to trace the packet flow through different states. It is responsible of analyzing the data to catch any intrusion behavior. This study shows two DFA models representing two kinds of DoS attacks, which are SYN flooding and IP spoofing. Another study [11] has built models to represent social-technical attacks using timed automata. As well as transform the attack trees, which is a way of attacks representation into timed automata models. For illustration, an IPTV case study is elaborated using a timed automata model.

Almseidin et. al. [12] have proposed an innovative mechanism to detect multi-step attacks (e.g., Denial of Service attack). These kinds of attacks occur after many steps, and they are usually harmful and target the victim's security without being detected. The proposed method is built using two concepts: fuzzy systems and automaton theory. Eventually, the system successfully achieves a detection rate of 97.84%. Although Finite Automata is an ideal method for representing and dealing with security attacks, the scientific community lacks relevant research. In this study, an intrusion detection model is build based on Time Dependent Finite Automata (TDFA) to detect Denial of Service (DoS) attack, particularly SYN Flood attack.

3. Background

This section provides the background regarding the application of the finite automata.

3.1 Deterministic Finite Automata

The deterministic finite automaton (DFA) is the simplest version of the finite automata in the Chomsky hierarchy of formal grammars [13]. Theoretically, a DFA is an abstract computational model used to represent modern computers. Just as a computer changes states of processes and generates some outputs given certain inputs, so does a DFA. DFAs are constructed to identify member strings of a specified language. Specifically, they identify those languages that belong to the class of regular languages [14]. DFAs hold two important properties. First, they represent a finite number of states. Second, they are deterministic meaning that on each input there is only one state to which the automaton can transition from its current state (could be same state or new one).

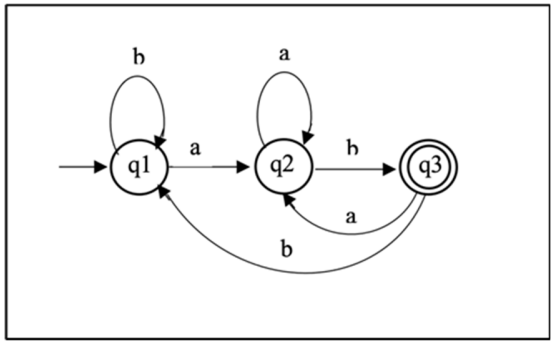

Fig. 1. A Representation of DFA Using State-Transition Diagram

A state-transition diagram is used to represent DFAs. It is a structure containing a set of states and transitions between them. Normally, final states are double circled, which accepts the entirety of an input string. In addition, a transition must be defined for

every symbol in the language alphabet. Figure 1 shows an example of a DFA that accepts strings ending with "ab" over the alphabet {a,b}. In this example, q1 is the start state and q3 is the final accepted state. If the DFA is in state q1 and gets an input "a", then it transits to state q2. On the other hand, if the DFA is in state 2 and gets an input "a", it remains in the same state. The correct pattern of an input string (i.e., "abab") will be accepted in q3, the final state. An analysis of Fig 1 shows that any input disrupting the pattern "ab" takes the DFA to its starting state. Moreover, the illustrated example has only one final state, which is not a requirement for DFA in general. A DFA can have multiple final states.

3.2 Time-Dependent Finite Automata

A time-dependent finite automaton (TDFA) is very similar to DFAs. When it comes to modeling real-time systems, a DFA might not be sufficient. Hence, it is crucial to extend the classical finite state machines with some capabilities to be able to reason about time constraints.

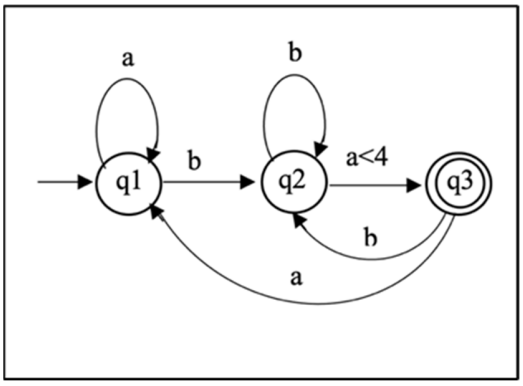

Fig. 2. A Representation of TDFA Using State-Transition Diagram

Figure 2 demonstrates an example TDFA. It is observed that this machine recognizes the pattern "b,a < $". In other words, the "a" must occur within four seconds of the initial "b". All transitions shown without the four second times are default transitions. If the desired input does not occur within the required time restraints, then the default transition directs the TDFA back to the appropriate state where it continues monitoring.

3.3 DoS Attack (SYN-Flood)

A DoS attack, also known as a denial-of-service attack, it a sort of attack on a networking structure that prevents a server from providing services to its clients. By flooding a server with massive packets of invalid data to send requests with a forged or faked IP address. Moreover, there are many different types of attacks that can be used against a server to slow it down, so it is unable to serve any requests [15]. There are several types of DoS attacks such as Ping of Death, TCP SYN Flood, and Smurf. The TCP three-way handshake process is performed during the establishment of a TCP connection between a client and a server. First, clients start sending SYN packets to the server with a random sequence number. Then, the server replays with a SYN-ACK packet that includes a random sequence number as well as an ACK number that acknowledges the client's sequence number. Finally, the client sends an ACK packet by acknowledging the server's sequence number. After the connection is established, the client and server can communicate and share messages as shown in Figure 3 [16].

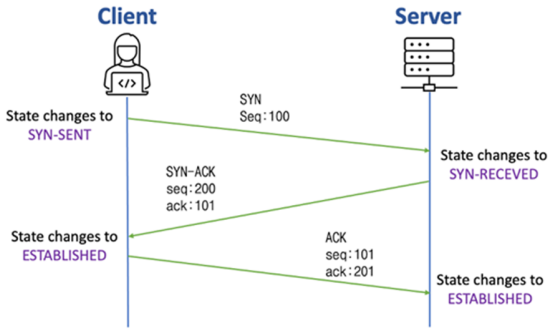

Fig. 3. TCP Three-Way Handshake Process

A SYN Flood is the most common type of Dos attack in which the attacker sends many SYN requests to the target system. It is ineffective against most modern networks. But, once the SYN has been received, the server must allocate resources before the ACK has been received. If a client does not send an ACK packet, the allocated memory cannot be recycled or utilized by other clients before the timeout. If a server receives a significant number of SYN packets from many other clients but no SYN-ACK packets, the server's resource limit is exceeded, and the server becomes unavailable to clients [17].

3.4 DoS using TDFA

The TDFA models are a preferred option for demonstrating DoS attack because of their inherent simplicity. In another way, a DoS attack consists of a specific sequence of network packets that could disable a specific target. However, the incremental conditions of an attack are represented by TDFA states. A TDFA's final state represents the points at which the attack has been successfully completed [7].

4. Proposed System Architecture

Now, in the upcoming subsections, we will discuss the overall architecture of our proposed system. Four main components make up our proposed system as shown in Figure 4: (1) Data Source Component, (2) Picket Filtration, (3) Token Generator, (4) TDFA.

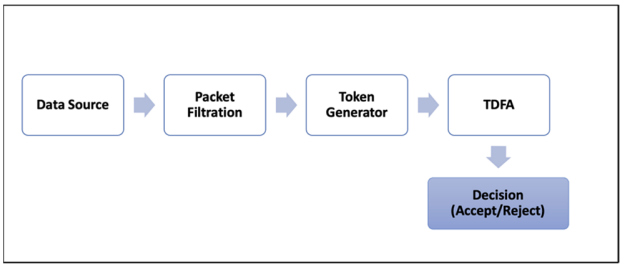

Fig. 4. Proposed System Architecture

4.1 Data Source

Before proceeding, it is important to note that our system detects SYN Flood attack, which occurs in TCP network traffic. It is an industry standard network protocol for communication [18]. Our proposed IDS collects the traffic from different data sources, primarily real-time traffic, and historical data. The best level of protection is offered by real-time monitoring because it may prevent ongoing attacks. Yet, adding historical datasets allows the site security officer (SSO) to check whether an attack might have occurred while the IDS was down for maintenance. Also, with the use of pre-recorded datasets, SSO can test and tune the system for newer attacks. Next, it is important to highlight the significant impact of TDFA in our proposed system. Earlier, we discussed how TDFA serves as a logical choice for representing DDoS attacks. Yet, the additional advantage of TDFAs for site security officer is that they allow the storage of an attack's-based signature. Also, the use of time-based information in attack signatures enhances the accuracy of detecting DoS attacks.

4.2 Packet Filtration

It is well known that network packets carry a lot of information (i.e., header length, sequence number, checksum, etc.), most of which is not necessary for the objectives of all networks based IDSs. Since our aim is a SYN Flood DoS attack, we consider the packet data fields relating to such attacks. For instance, source and destination IP addresses in addition to a variety of flag fields (i.e., ACK and SYN). The role of the packet filtration unit is to process important network information for the successive components of the system. As discussed in the data source section, data is originated from either a real time data source or a historical data source. The Tcpdump [19] tool is used to analyze network activity and produce a record of a specific node in ASCII text format. The following are the fields parsed by PFC: (1) Timestamp, (2) Source IP address, (3) Destination IP address, (4) Destination port number, (5) SYN-flag, (6) ACK-flag.

In the stored data source, data is usually located in a log file and may or may not be in ASCII format, but in binary format. Since the PFC processes packet information in ASCII format only, any binary information must be converted to ASCII text format before it can be processed. The end output is a bound message which comprises specific network event information.

4.3 Token Generator

After network event data is filtrated by the packet filtration unit, the data is passed to the token generator. The token generator is then responsible for transforming the messages, each of which represents a specific network event, into unique tokens. The information contained in a single message may cause the token generator to generate a "sequence" of tokens. These preconfigured tokens, each of which is a string of one or more ASCII characters, provide a language that our system uses to recognize DoS attempts. The token generator provides all the required event details to the TDFA to decide. After reading messages from the packet filtration unit, the token generator determines the sequence number of the TCP connection, source IP address, and packets and time-interval between SYN requests to generate an event that is sent to TDFA.

4.4 TDFA

There is a distinct core component in most intrusion detection systems that is responsible for identifying attacks. In this architecture, TDFA is designed to detect DDoS attacks. With this approach, the TDFA can determine whether a host is at danger of attack by analyzing data included in tokens. The following TDFA in Figure 5 designed to detect SYN Flood attack by examining network packet attributes. As shown, there are three states in the TDFA q0 safe, q1 critical, and q2 DoS attack. The q0 Safe is the initial state. This state represents the first line of defense in this system. Furthermore, the q0 state performs three checks on the generated event. Start by checking the source IP address, then move to checking the number of SYN requests that were sent from the source address. After that, q0 calculates the interval time between SYN requests. In the case of a higher number of SYN requests than 10 requests, q0 will jump to q1 Critical. In the next state, q1 will check the interval time that was calculated by q0. If the interval time is greater than 10 seconds, this is an indicator of a DoS attack. q1 will move to the final state q2.

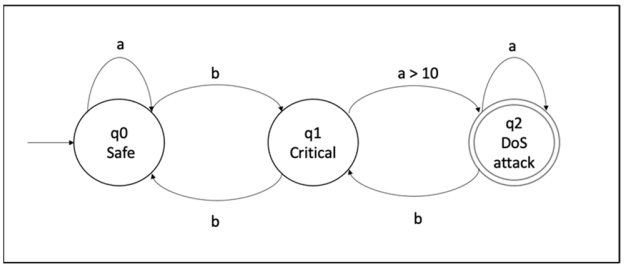

Figure 5. TDFA for SYN Flood Detection

For proof of concept, python code has been implemented to design the proposed TDFA to detect DoS attacks for full code see Appendix A. First, TDFA states and inputs should be provided by the user to create a new TDFA. Then, as shown in Fig 6, initial and final states must be specified to ensure the correctness of the created TDFA.

After that, the code provides two options to recreate the TDFA or start testing a DoS attack as shown in Fig. 7. If the second option is selected, it will ask you to enter the time interval between SYN requests that were sent by outsiders. Then, the code starts analyzing all the required input data to make the decision if this case is a DoS attack or not. If the time interval between SYN requests exceeds 10 seconds, the system will treat this as a true DoS attack, and a security alert should be sent to system administrator as a detection action. Otherwise, if the time interval is less than 10 seconds, the event will go back to critical status.

```
 TDFA
Enter TDFA statesq0 q1 q2
STATES : ['q0', 'q1', 'q2']
Enter TDFA inputa b
ALPHABET : ['a', 'b']
Enter transitions for state q0. If required, use 'REJECT'.
CURRENT STATE : q0      INPUT ALPHABET : a      NEXT STATE : q0
CURRENT STATE : q0      INPUT ALPHABET : b      NEXT STATE : q1
Enter transitions for state q1. If required, use 'REJECT'.
CURRENT STATE : q1      INPUT ALPHABET : a      NEXT STATE : q2
CURRENT STATE : q1      INPUT ALPHABET : b      NEXT STATE : q0
Enter transitions for state q2. If required, use 'REJECT'.
CURRENT STATE : q2      INPUT ALPHABET : a      NEXT STATE : q2
CURRENT STATE : q2      INPUT ALPHABET : b      NEXT STATE : q1
Enter the Initial_State: q0
Enter the Final_State: q2
```

Fig. 6. TDFA Code Implementation – 1

```
Enter Choice:
1. Recreate TDFA
2. Test DoS Attack
Enter Your choice : 2
Enter the value of a as time interval for DoS attack : 15
DoS attack, send alert to system admin

Enter Choice:
1. Recreate TDFA
2. Test DoS Attack
Enter Your choice : 2
Enter the value of a as time interval for DoS attack : 5
It is not a DoS attack
```

Fig. 7. TDFA Code Implementation – 2

5. Conclusion

The DoS attack is one of the most dangerous and serious attacks that affect network availability. Detecting the attack properly and in timely manner will help significantly in reducing the damage it causes. In this paper, a proposed detection system based on Time Dependent Finite Automata (TDFA) to detect potential SYN Flood attacks. The system components are explained in detail with a code implementation of TDFA component. For future work, the system can be widened to detect multiple network attacks by the help of TDFA concept [20-50].

References

[1] T. Penttinen, "Distributed Denial-of-Service Attacks in the Internet," 2005.

[2] M. Bogdanoski, T. Shuminoski, and A. Risteski, "Analysis of the SYN Flood DoS Attack," Computer Network and Information Security, vol. 8, pp. 1–11, 2013, doi: 10.5815/ijcnis.2013.08.01.

[3] R. Tandon, "A Survey of Distributed Denial of Service Attacks and Defenses," Aug. 2020, Accessed: May 18, 2022. [Online]. Available: http://arxiv.org/abs/2008.01345

[4] J. W. Branch, "EXTENDED AUTOMATA-BASED APPROACHES TO INTRUSION DETECTION," 2003.

[5] A. A. Sharipbay, Z. S. Saukhanova, G. B. Shakhmetova, and N. S. Saukhanov, "Application of finite automata in cryptography," Jun. 2019. doi: 10.1145/3330431.3330452.

[6] Z. F. Han, J. P. Zou, H. Jin, Y. P. Yang, and J. H. Sun, "Intrusion detection using adaptive time-dependent finite automata," in Proceedings of 2004 International Conference on Machine Learning and Cybernetics, 2004, vol. 5, pp. 3040–3045. doi: 10.1109/icmlc.2004.1378554.

[7] J. W. Branch, A. Bivens, C. Y. Chan, T. K. Lee, and B. K. Szymanski, "Denial of Service Intrusion Detection Using Time Dependent Deterministic Finite Automata," 2002.

[8] P. M. Rathod, N. Marathe, and A. v. Vidhate, "A survey on Finite Automata based pattern matching techniques for network Intrusion Detection System (NIDS)," Jan. 2015. doi: 10.1109/ICAECC.2014.7002456.

[9] X. Yu, W. C. Feng, D. Yao, and M. Becchi, "O3FA: A scalable finite automata-based pattern-matching engine for out-of-order deep packet inspection," in ANCS 2016 - Proceedings of the 2016 Symposium on Architectures for Networking and Communications Systems, Mar. 2016, pp. 1–11. doi: 10.1145/2881025.2881034.

[10] Q. W. Shang, K. Cao, and F. Wang, "The study on network attacks based on automaton theory," in Procedia Engineering, 2011, vol. 23, pp. 653–658. doi: 10.1016/j.proeng.2011.11.2561.

[11] N. David et al., "Modelling social-technical attacks with timed automata," in MIST 2015 - Proceedings of the 7th ACM CCS International Workshop on Managing Insider Security Threats, co-located with CCS 2015, Oct. 2015, pp. 21–28. doi: 10.1145/2808783.2808787.

[12] M. Almseidin, I. Piller, M. Al-Kasassbeh, and S. Kovacs, "Fuzzy automaton as a detection mechanism for the multi-step attack," International Journal on Advanced Science, Engineering and Information Technology, vol. 9, no. 2, pp. 575–586, 2019, doi: 10.18517/ijaseit.9.2.7591.

[13] G. Kim, "the Relationship Between the Chomsky Hierarchy and Automata," pp. 1–10, 2019.

[14] P. Grachev, I. Lobanov, I. Smetannikov, and A. Filchenkov, "Neural network for synthesizing deterministic finite automata," Procedia Computer Science, vol. 119, pp. 73–82, 2017, doi: 10.1016/j.procs.2017.11.162.

[15] K. M. Elleithy, D. Blagovic, W. K. Cheng, P. Sideleau, A. " Et, and W. Cheng, "Denial of Service Attack Techniques: Analysis, Implementation and Comparison," 2005. [Online]. Available: http://digitalcommons.sacredheart.edu/computersci_fac

[16] F. H. Hsu, Y. L. Hwang, C. Y. Tsai, W. T. Cai, C. H. Lee, and K. W. Chang, "TRAP: A Three-way handshake server for TCP connection establishment," Applied Sciences (Switzerland), vol. 6, no. 11, Nov. 2016, doi: 10.3390/app6110358.

[17] S. Deore and A. Patil, "Survey Denial of Service classification and attack with Protect Mechanism for TCP SYN Flooding Attacks Atul Patil," International Research Journal of Engineering and Technology, 2016, doi: 10.1109/TC.2003.1176986.

[18] M. W. Jeter, Network Programming, vol. 1. 2018. doi: 10.1201/9780203749333-6.

[19] "tcpdump(1) man page | TCPDUMP & LIBPCAP." https://www.tcpdump.org/manpages/tcpdump.1.html (accessed May 18, 2022).

[20] Fahd Alhaidari, Nouran Abu Shaib, Maram Alsafi, Haneen Alharbi, Majd Alawami, Reem Aljindan, Atta-ur Rahman, Rachid Zagrouba, "ZeVigilante: Detecting Zero-Day Malware Using Machine Learning and Sandboxing Analysis Techniques", Computational Intelligence and Neuroscience, vol. 2022, Article ID 1615528, 15 pages, 2022. https://doi.org/10.1155/2022/1615528.

[21] M. Jamal, N.A. Zafar, A. Rahman, D. Musleh, M. Gollapalli, S. Chabani, "Modeling and Verification of Aircraft Takeoff Through Novel Quantum Nets," Computers, Materials and Continua, vol. 72, no. 2, pp. 3331-3348, 2022.

[22] A. Rahman, M. Mahmud, T. Iqbal, L. Saraireh, H. Kholidy et al., "Network Anomaly Detection in 5G Networks," Mathematical Modelling of Engineering Problems, vol. 9, No. 2, pp. 397-404, 2022.

[23] F. Al-Jawad, R. Alessa, S. Alhammad, B. Ali, M. Al-Qanbar, A. Rahman, "Applications of 5G and 6G in Smart Health Services," International Journal of Computer Science and Network Security, vol. 22, no. 3, pp. 173-182, 2022.

[24] S.U. Rehman, M. Mahmud, A. Rahman, I.U. Haq, M. Safdar, "Information Security in Business: A Bibliometric Analysis of the 100 Top Cited Articles," Library Philosophy and Practice (e-journal), 5354, 2021.

[25] R. Zagrouba, A. AlAbdullatif, K. AlAjaji, N. Al-Serhani, F. Alhaidari, A. Almuhaideb, A. Rahman, "Authenblue: a new authentication protocol for the industrial internet of things," Computers, Materials & Continua, vol. 67, no.1, pp. 1103–1119, 2021.

[26] A. Rahman, S. Dash, A.K. Luhach, N. Chilamkurti, S. Baek, Y. Nam, "A Neuro-Fuzzy Approach for User Behavior Classification and Prediction", Journal of Cloud Computing, 8(17), 2019.

[27] A. Rahman, “Memetic Computing based Numerical Solution to Troesch Problem”, Journal of Intelligent and Fuzzy Systems, 37(1):1545-1554, 2019.

[28] A. Rahman, “Optimum Information Embedding in Digital Watermarking”, Journal of Intelligent and Fuzzy Systems, 37(1):553-564, 2019.

[29] A. Rahman, S. Abbas, M. Gollapalli, R. Ahmed, S. Aftab et al., “Rainfall Prediction System Using Machine Learning Fusion for Smart Cities,” Sensors, vol. 22, no. 9, pp. 1-15, 2022. https://doi.org/10.3390/s22093504.

[30] N. M. Ibrahim, D. G. I. Gabr, A. Rahman, S. Dash, A. Nayyar, “A deep learning approach to intelligent fruit identification and family classification,” Multimedia Tools and Applications, 2022. https://doi.org/10.1007/s11042-022-12942-9.

[31] M Gollapalli, A. Rahman, D. Musleh, N. Ibrahim et al., “A Neuro-Fuzzy Approach to Road Traffic Congestion Prediction,” Computers, Materials and Continua, vol. 72, no. 3, pp. 295-310, 2022.

[32] A. Rahman, K. Sultan, I. Naseer, R. Majeed, D. Musleh et.al., “Supervised Machine Learning-based Prediction of COVID-19,” Computers, Materials & Continua, vol. 69, no.1, pp. 21-34, 2021. DOI: 10.32604/cmc.2021.013453.

[33] S. M. Alotaibi, A. Rahman, M. I. Basheer and M. A. Khan, “Ensemble machine learning based identification of pediatric epilepsy,” Computers, Materials & Continua, vol. 68, no.1, pp. 149–165, 2021.

[34] G. Zaman, H. Mahdin, K. Hussain, A. Rahman, J. Abawajy and S. A. Mostafa, “An Ontological Framework for Information Extraction from Diverse Scientific Sources,” IEEE Access, vol. 9, pp. 42111-42124, 2021. doi: 10.1109/ACCESS.2021.3063181.

[35] A. Rahman, S. Dash, M. Ahmad, T. Iqbal, “Mobile Cloud Computing: A Green Perspective,” Intelligent Systems, Lecture Notes in Networks and Systems book series (LNNS, volume 185), pp. 523-533, 2021.

[36] A. Rahman, “GRBF-NN based ambient aware realtime adaptive communication in DVB-S2.” J Ambient Intell Human Comput (2020). https://doi.org/10.1007/s12652-020-02174-w.

[37] F. Alhaidari, A. Rahman, & R. Zagrouba, “Cloud of Things: architecture, applications and challenges.” J Ambient Intell Human Comput (2020). https://doi.org/10.1007/s12652-020-02448-3.

[38] A. Rahman, S. Dash, & A.K. Luhach, “Dynamic MODCOD and power allocation in DVB-S2: a hybrid intelligent approach.” Telecommun Syst, vol. 76, pp. 49–61, 2021. https://doi.org/10.1007/s11235-020-00700-x.

[39] M. Ahmad, M.A. Qadir, A. Rahman et al., “Enhanced query processing over semantic cache for cloud based relational databases.” J Ambient Intell Human Comput (2020). https://doi.org/10.1007/s12652-020-01943-x

[40] M. Mahmud, A. Rahman, M. Lee, J. Choi, “Evolutionary-based image encryption using RNA codons truth table”, Optics & Laser Technology, vol. 121:1-8, 2020.

[41] G. Zaman, H. Mahdin, K. Hussain, A. Rahman, N. Ibrahim, N.Z.M. Safar, “Digital Library of Online PDF Sources: An ETL Approach,” IJCSNS, vol. 20 (11), pp. 172-181, 2020.

[42] M. Ahmad, U. Farooq, A. Rahman, A. Alqatari, S. Dash & A.K. Luhach, “Investigating TYPE constraint for frequent pattern mining”, Journal of Discrete Mathematical Sciences and Cryptography, 22:4, 605-626, 2019.

[43] K. Sultan, I.M. Qureshi, A. Rahman, B.A. Zafar, M. Zaheer, “CSI Based Multiple Relay Selection and Transmit Power Saving Scheme for Underlay CRNs Using FRBS and Swarm Intelligence,” International Journal of Applied Metaheuristic Computing (IJAMC) 10 (3), 1-18, 2019.

[44] A. Rahman, M.I.B. Ahmed, “Virtual Clinic: A CDSS Assisted Telemedicine Framework”, Chapter 15, Telemedicine Technologies, 1st Edition. Elsevier, 2019.

[45] L. Ajmi, Hadeel, N. Alqahtani, A. Rahman and M. Mahmud, “A Novel Cybersecurity Framework for Countermeasure of SME's in Saudi Arabia,” 2019 2nd International Conference on Computer Applications & Information Security (ICCAIS), 2019, pp. 1-9, doi: 10.1109/CAIS.2019.8769470.

[46] A. Rahman, Maqsood Mahmud, Kiran Sultan, Nahier Aldhafferi, Abdullah Alqahtani, Dhiaa Abdullah, “Medical Image Watermarking for Fragility and Robustness: A Chaos, ECC and RRNS Based Approach”, Journal of Medical Imaging and Health Informatics, vol. 8(6), pp. 1192-1200, July 2018.

[47] A. Rahman, "Efficient Decision Based Spectrum Mobility Scheme for Cognitive Radio Based V2V Communication System," Journal of Communications, vol. 13, no. 9, pp. 498-504, 2018. Doi: 10.12720/jcm.13.9.498-504.

[48] A. Rahman, F.A. Alhaidari, “Querying RDF Data”, Journal of Theoretical and Applied Information Technology 26(22):7599-7614, 2018.

[49] M.Z. Muzaffar, I.M. Qureshi, A. Rahman, F.A. Alhaidari, M.A.A. Khan, “Compressed Sensing for Security and Payload Enhancement in Digital Audio Steganography”, Journal of Information Hiding and Multimedia Signal Processing, 15(6):1506-1517, Nov. 2018.

[50] A. Rahman, S.A. Alrashed, A. Abraham, “User Behavior Classification and Prediction using FRBS and Linear Regression” Journal of Information Assurance and Security, vol. 12, no. 3, pp. 86-93, 2017.

Appendix A

```
class TDFA:

    def __init__(self):
        self.Q = self.create_TDFA_states()
        self.SIGMA = self.create_TDFA_alphabet()
        self.DELTA = self.populate_TDFA()
        self.START_STATE, self.ACCEPT_STATES =
self.set_initial_state()
        self.CURRENT_STATE = None
```

```
    def set_initial_state(self):
        while(True):
            start = input("Enter the
Initial_State: ")
            accept = input("Enter the
Final_State: ").split()
            if (start in self.Q) and
(set(accept).issubset(set(self.Q))):
                return start, accept
            else:
                print("Please re
enter".format(self.Q))

    def create_TDFA_states(self):

        Q_input = input("Enter TDFA
states").split()
        print("STATES : {}".format(Q_input))
        return Q_input

    def create_TDFA_alphabet(self):
        SIGMA_input = input("Enter TDFA
input").split()
        print("ALPHABET :
{}".format(SIGMA_input))
        return SIGMA_input

    def populate_TDFA(self):
        transition_dict = {el : {el_2 : 'REJECT'
for el_2 in self.SIGMA} for el in self.Q}

        for key, dict_value in
transition_dict.items():
            print("Enter transitions for state
{}. If required, use 'REJECT'.".format(key))

            for input_alphabet, transition_state
in dict_value.items():
                transition_dict[key][input_alphab
et] = input("CURRENT STATE : {}\tINPUT ALPHABET :
{}\tNEXT STATE : ".format(key, input_alphabet))

        return transition_dict


    def run_machine(self, in_string):

        if in_string > 10:
            return False
        else:
            return True


if __name__ == "__main__":
    check = True
    print("\n TDFA")
    machine = TDFA()
    while(check):
        choice = int(input("\nEnter Choice:\n1.
Re_create TDFA\n2. Test DoS Attack \nEnter Your
choice : "))
        if (choice == 1):
            machine = TDFA()
        elif (choice == 2):
            input_string = int (input("Enter the
value of a as time interval for DoS attack : "))
            print("It is not a DoS attack" if
machine.run_machine(input_string) else "DoS
attack, send alert to system admin")
        else:
            check = False
```

Appendix A

```
class TDFA:

    def __init__(self):


        self.Q = self.create_TDFA_states()
        self.SIGMA = self.create_TDFA_alphabet()
        self.DELTA = self.populate_TDFA()
        self.START_STATE, self.ACCEPT_STATES =
self.set_initial_state()
        self.CURRENT_STATE = None

    def set_initial_state(self):
        while(True):
            start = input("Enter the Initial_State:
")
            accept = input("Enter the Final_State:
").split()
            if (start in self.Q) and
(set(accept).issubset(set(self.Q))):
                return start, accept
            else:
                print("Please re
enter".format(self.Q))

    def create_TDFA_states(self):

        Q_input = input("Enter TDFA states").split()
        print("STATES : {}".format(Q_input))
        return Q_input

    def create_TDFA_alphabet(self):
        SIGMA_input = input("Enter TDFA
input").split()
        print("ALPHABET : {}".format(SIGMA_input))
        return SIGMA_input

    def populate_TDFA(self):
        transition_dict = {el : {el_2 : 'REJECT' for
el_2 in self.SIGMA} for el in self.Q}

        for key, dict_value in
transition_dict.items():
            print("Enter transitions for state {}. If
required, use 'REJECT'.".format(key))

            for input_alphabet, transition_state in
dict_value.items():
                transition_dict[key][input_alphabet]
= input("CURRENT STATE : {}\tINPUT ALPHABET :
{}\tNEXT STATE : ".format(key, input_alphabet))

        return transition_dict


    def run_machine(self, in_string):

        if in_string > 10:
            return False
        else:
            return True


if __name__ == "__main__":
    check = True
    print("\n TDFA")
    machine = TDFA()
    while(check):
        choice = int(input("\nEnter Choice:\n1.
Re_create TDFA\n2. Test DoS Attack \nEnter Your
choice : "))
        if (choice == 1):
            machine = TDFA()
        elif (choice == 2):
            input_string = int (input("Enter the
value of a as time interval for DoS attack : "))
            print("It is not a DoS attack" if
machine.run_machine(input_string) else "DoS attack,
send alert to system admin")
        else:
            check = False
```