Analyzing the Impact of Malicious Network Traffic: Project Foundation

ALEXANDER M. VALENTIN, Kennesaw State University, USA

CCS Concepts: • **Networks** → **Network simulations**; **Network performance modeling**; *Network performance analysis*; *Network experimentation.*

Additional Key Words and Phrases: Distributed Denial of Service (DDoS), Botnet, Load-Balancing, Distributed Adaptive Routing, Bandwidth Exhaustion, Resource Depletion, Queuing Theory, Network Topology

ACM Reference Format:
Alexander M. Valentin. 2025. Analyzing the Impact of Malicious Network Traffic: Project Foundation. *ACM Trans. Access. Comput.* 1, 1, Article 1 (September 2025), 6 pages.

1 Project Overview

The aim of this project is to analyze the impact of malicious network traffic on network systems. This area of study is extremely relevant to the overall network system domain, especially considering the rise of DDoS attacks against large enterprise services year-over-year [3]. The results intend to explore how Distributed Denial of Service attacks (DDoS) impact server response time and the ability to serve legitimate network traffic, furthermore, how do safeguards such as load-balancing help reduce latency, improve throughput, and prevent server deterioration when under attack. This project will simulate legitimate and malicious network traffic, servers and server responses, and network performance degradation under load. Items outside the scope of this project and thus will not be included in the final simulation include the implementation of End-to-End security protocols (i.e.: TLS handshakes, firewalls, IDS/IPS).

2 System Description

2.1 System Components and Their Properties

Key entities of this simulation include the following:

- *Simulation Executive:* Env: SimPy.Environment, Network: Network, Botnet: Botnet, TrafficNetwork: LegitimateTrafficNetwork, DataCollector: DataCollector, SimulationDuration: Float
- *Data Collector:* LegitimateReqDropped: Int, LegitimateServeCount: Int, ServerThroughput: Int, ServedReqRespTimes: List[Float], MeanReqRespTime: Float, AverageQueueLength: Float, ServerUtilization: Float, QueueLengthsSeen: List[Int], SimulationStartTime: Float, SimulationEndTime: Float
- *Network:* Env: SimPy.Environment, Servers: List[NetworkServer], Router: NetworkRouter
- *Network Router:* CurrentAlgorithm: String, AuthorizedServers: List[NetworkServer]
- *Network Server:* Env: SimPy.Environment, ServerID: String, ResourceCapacity: Int, Resource: SimPy.Resource, ProcessDelay: Float
- *Botnet:* Env: SimPy.Environment, TargetNetwork: Network, AttackRate: Float, AttackStart: Float, AttackDuration: Float, MaliciousClientCount: Int, attackProcess: Process
- *LegitimateTrafficNetwork:* Env: SimPy.Environment, TargetNetwork: Network, TrafficRate: Float, TrafficStart: Float, TrafficDuration: Float, LegitimateClientCount: Int, TrafficProcess: Process

Author's Contact Information: Alexander M. Valentin, Kennesaw State University, Marietta, Georgia, USA.

2025. ACM 1936-7236/2025/9-ART1
https://doi.org/

- *Request:* RequestID: String, SourceID: String, TrafficType: String, ArrivalTime: Float, ServedTime: Float, LoadSize: Float
- *Base Network Client:* Env: SimPy.Environment, TargetNetwork: Network, ClientID: String, RequestRate: Float
- *Legitimate Network Client:* (No additional attributes)
- *Malicious Network Client:* IsEnabled: Bool

2.2 System Dynamics

Components throughout this malicious network traffic simulation will interact with one another in such a way that models targeted and legitimate network traffic flow. Upon the start of the simulation, a set of instructions defining the target network and attack rate will be passed to the botnet, whilst legitimate traffic is generated and sent to the same target network. These legitimate and malicious requests will arrive at the target network, where they will be directed to a network router, this router will then decide how to direct client requests to the available servers in the network. Upon arrival at a server, the request will be added to a server processing queue, where different processing threads will process and return the expected results to the client that requested the resources. Emergent behaviors arise over time as the system undergoes continued load from the attacking network, since the attacking network aims to exhaust resources from processing servers, this component interaction should hypothetically impact the server's ability to process legitimate requests. The data collection component will monitor the network as it undergoes this continued stress, recording metrics such as legitimate request drop rate, request response time, average server queue length, server throughput, and server utilization.

2.3 Core Models and Algorithms

In order to accurately simulate the proposed system at hand, it is imperative to properly understand and implement the core models and algorithms that drive network traffic, probability algorithms regarding resource consumption, and other computational/conceptual models that drive system networking. As outlined in Kazeem A. Adnan A. and Anish M.'s academic paper, *DDoS Attack and Detection Methods in Internet-Enabled Networks*, there are three important algorithms that will help drive core data collection logic throughout the course of this simulation, namely:

Probability of Depletion of Bandwidth Exhaustion:

$$P_B = \frac{(a^c)}{\sum_{i=0}^{e} \frac{a^i}{i!}} \tag{1}$$

Probability of Depletion of Victim Resources:

$$P_{TA} = 1 - (1 - P_B)(1 - P_M) \tag{2}$$

Probability of Successful Attack:

$$P_{wa} = \begin{cases} P = \lim_{t\to\infty} \frac{X_t}{t \times C} \\ P_{wa} = \overline{P} \end{cases} \tag{3}$$

These mathematical models will be imperative to data collection and will help provide a baseline to compare expected simulation results versus actual simulation results. In addition, network traffic routing will be modeled after a modified version of the Distributed Adaptive Routing algorithm outlined in the academic paper *Routing Algorithms: A Review* presented by Ujjwal S., Vikas K. and Shubham K., under the guidance of Dr. Jayasheela, in which the network router in the malicious traffic simulation will utilize metrices such as shortest path to networks as well as ongoing processing traffic to make request routing decisions. Furthermore, the architecture behind the attacking network will be modeled after a modified version of the centralized C&C topology outlined in Divya N., Pooja W., Sanjay S., and Deepak K's academic paper, *BOTNET: Lifecycle, Architecture and Detection Model.* While there are other models and algorithms that will be introduced to the final system, the models and algorithms outlined above are extremely imperative to the successful implementation of the final system.

2.4 Assumptions

Since the simulation is not being conducted in an actual network environment, a key assumption is that request response time will not be impacted by the network topology itself and the communication time between network components will be zero. As a result, it is imperative to clarify that the only factors impacting request response time is the time taken to process the request itself, and any additional time the request may have sat in the processing queue and awaiting queue. Furthermore, it is key to clarify that the servers in this simulation will all be identical to one another, there will be no differences in terms of the processing capabilities of any of the servers included in the target network. Another key assumption is that the attacking network will have no emergent behavior throughout the course of the simulation. The botnet in this simulation made to model an attacking network will send requests at a fixed rate to the target server, as opposed to exhibiting dynamic behavior in order to combat defensive network strategies.

3 Implementation Approach

In order to develop the malicious network traffic simulation, Python will be the primary language of the project due to my familiarity and previous experience with the language. Furthermore, Python supports numerous libraries tailored to discrete-event simulations, allowing for robust and more expansive test simulations.

The following tools, libraries, and frameworks will be utilized in order to implement all of the necessary functionality required by such a simulation:

- Development Environment: PyCharm, Git, GitHub
- Simulation Tooling: SimPy
- Data Processing and Analysis: NumPy
- Result Visualization: Matplotlib
- Testing: PyTest

Throughout the course of the simulation, metrics such as legitimate request drop rate, request response time, average server queue length, server throughput, server utilization will be collected in order to gather information about how a network responds and consequences when under attack.

4 Literature Review

The academic paper *Routing Algorithms: A Review* presented by Ujjwal S., Vikas K. and Shubham K., under the guidance of Dr. Jayasheela, provides a comprehensive breakdown of the routing algorithms that drive network traffic. As outlined in the paper, these algorithms are divided into three distinct categories based on the way that they manage traffic: Adaptive Routing Algorithms (Isolated, Centralized, Distributed), Non-Adaptive Routing Algorithms (Flooding, Random Walk), and Hybrid Routing Algorithms (Link State, Distance Vector). The routing algorithms discussed throughout the paper are imperative to the implementation of a successful malicious network traffic simulation, as this simulation aims to accurately model real-world network traffic whilst analyzing a system under load [5]. In order to successfully implement these traffic routing concepts into the network simulation, it is advantageous to modify an adaptive routing algorithm such as Distributed Adaptive Routing into a single "load-balancing" network router, to conceptually serve as a single component that dynamically directs traffic based on load and routing algorithm decisions.

Kazeem A. Adnan A. and Anish M.'s academic paper, *DDoS Attach and Detection Methods in Internet-Enabled Networks*, provides excellent insight into the taxonomy and architecture of both centralized and decentralized

DDoS attacks, as well as the algorithms used to model probabilities such as: Bandwidth exhaustion. Depletion of resources, and Successful attack probability [1]. These concepts are extremely relevant and imperative to the successful implementation of a malicious network traffic simulation, as the writing contains details pertaining to proper attack architecture that is relevant to modeling how a real-world attack is performed. In the case of this network traffic simulation, we aim to implement and potentially modify the algorithms used to determine successful attack probability, probability of bandwidth exhaustion, and probability of victim's resource depletion, to better capture how malicious traffic effects networks.

Divya N., Pooja W., Sanjay S., and Deepak K's academic paper, B*OTNET: Lifecycle, Architecture and Detection Model* dives deep into the lifecycle and deployment of a Botnet, the different design architectures and implementations of such a network, as well as different detection methods for malicious computer networks designed for Distributed Denial of Service (DDoS) attacks. The main purpose of this paper with regard to the malicious network simulation at hand is the implementation of the attacking network designed to emulate a real-world attack. According to the writing, there are five main design topologies when dealing with botnet architecture [4]:

- Star or Centralized C&C Topology: Relies on a single C&C resource to communicate with all bot agents. Each bot agent is issued new instructions directly from the central C&C point.
- Multi-Server Topology: A logical extension of the Star topology in which multiple servers are used to provide C&C instructions to bot agents, where the command systems communicate amongst each other to manage the botnet.
- Hierarchical C&C Topology: A Bot Master communicates with a central botnet C&C Server which send instructions propagating down a layered hierarchy of connected agents.
- Peer-to-Peer Topology: Each host periodically connects to its neighbor to retrieve orders from the Botmaster. The Botmaster only needs to connect to one of the Bots (peer) to send his commands all over the network.
- Unstructured or Random C&C Topology: Each bot has the ability to scan the internet in order to find another Bot.

As for the design implementation of an attacking network in the malicious network traffic simulation, we aim to utilize modified centralized C&C topology in order to model an attacking network. Due to the scope of the simulation, there is no need to implement a communication protocol to manage instructions being issued to network agents. In this simulation, attacking agents will be issued instructions directly as opposed to reliance on a protocol such as IRC, HTTP, and P2P.

Botnets Multiply and Level Up is a DDoS threat intelligence report authored by Chris Conrad and published by NETSCOUT SYSTEMS, INC. in early 2022. The report provides details regarding recent growth in the botnet landscape, the impact this technology has on organizations, and provides information about prevalent botnets. The report lists botnets such as Mirai, Meris, Dvinis, and Killnet, and lists important information such as their deployment date, notable targets, attack capabilities, attack vectors, attack surface, and node size. This information is extremely important in terms of designing the malicious network traffic simulation, as it provides a reference point in terms of the capabilities and size of real-world attack networks. According to the writing, at its height Meris was estimated to contain an estimated 250,000 compromised devices, allowing the botnet to execute attacks consisting of 2 million requests per second [2]. By having access to quantifiable metrics concerning the size and capabilities of real-world botnets that execute enterprise-level attacks, we can modify and model the size of our simulated attacking network accordingly in a way that accurately represents the surface size and capability of real-world attack networks.

William Stalling's *Queuing Analysis* is an academic report detailing the different queuing models utilized throughout computer and network analysis. Throughout the reading, Stalling goes into great detail about prevalent network topics such as Single-Server Queues, Multiserver Queues, Networks of Queues, as well as other Queuing models. This resource has been of significant assistance, as it has helped determine how to come to an accurate algorithm in terms of how to estimate average response time and utilization of servers within the simulation [6]. Based on the reading, the final implementation of servers will be a simple FIFO queue model with multiple worker threads processing requests stored in the queue, however, we can utilize algorithms outlined in Stalling's reading to come to accurate algorithms to gather metrics surrounding the network system as a whole.

5 System Diagrams

The following images are designed to give the reader a better understanding of the overall system architecture and composition of the proposed simulation; main simulation classes, relationships, key attributes and methods, inheritance and composition, and system interactions are illustrated.

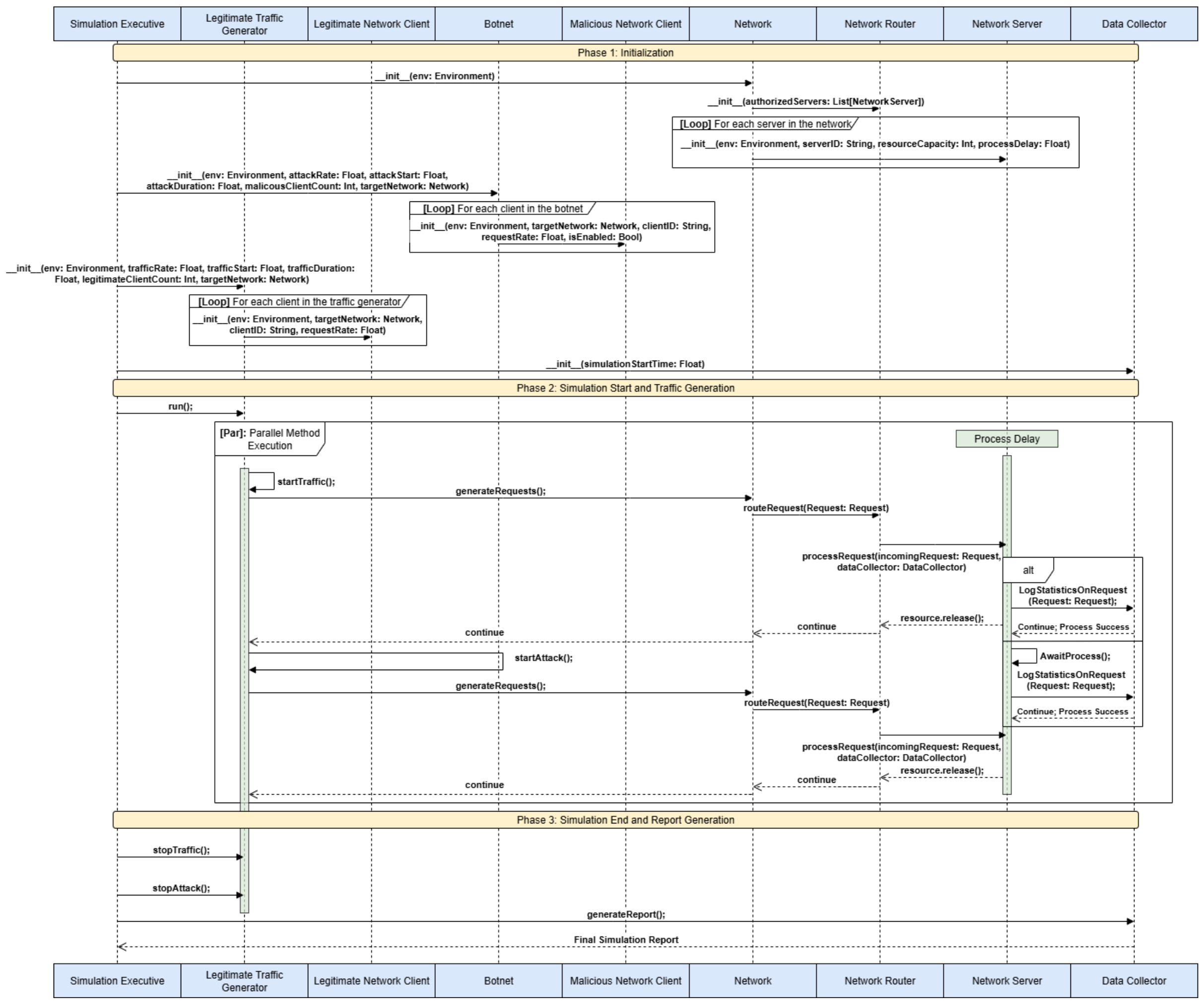
Simulation Executive
Legitimate Traffic Generator
Legitimate Network Client
Botnet
Malicious Network Client
Network
Network Router
Network Server
Data Collector
Phase 1: Initialization
__init__(env: Environment)
__init__(authorizedServers: List[NetworkServer])
[Loop] For each server in the network
__init__(env: Environment, serverID: String, resourceCapacity: Int, processDelay: Float)
__init__(env: Environment, attackRate: Float, attackStart: Float, attackDuration: Float, malicousClientCount: Int, targetNetwork: Network)
[Loop] For each client in the botnet
__init__(env: Environment, targetNetwork: Network, clientID: String, requestRate: Float, isEnabled: Bool)
__init__(env: Environment, trafficRate: Float, trafficStart: Float, trafficDuration: Float, legitimateClientCount: Int, targetNetwork: Network)
[Loop] For each client in the traffic generator
__init__(env: Environment, targetNetwork: Network, clientID: String, requestRate: Float)
__init__(simulationStartTime: Float)
Phase 2: Simulation Start and Traffic Generation
run();
[Par]: Parallel Method Execution
Process Delay
startTraffic();
generateRequests();
routeRequest(Request: Request)
processRequest(incomingRequest: Request, dataCollector: DataCollector)
alt
LogStatisticsOnRequest (Request: Request);
resource.release();
continue
Continue; Process Success
continue
startAttack();
AwaitProcess();
LogStatisticsOnRequest (Request: Request);
generateRequests();
routeRequest(Request: Request)
Continue; Process Success
processRequest(incomingRequest: Request, dataCollector: DataCollector)
resource.release();
continue
continue
Phase 3: Simulation End and Report Generation
stopTraffic();
stopAttack();
generateReport();
Final Simulation Report
Simulation Executive
Legitimate Traffic Generator
Legitimate Network Client
Botnet
Malicious Network Client
Network
Network Router
Network Server
Data Collector

Fig. 1. UML sequence diagram showing interactions within the malicious network traffic discrete-event simulation.

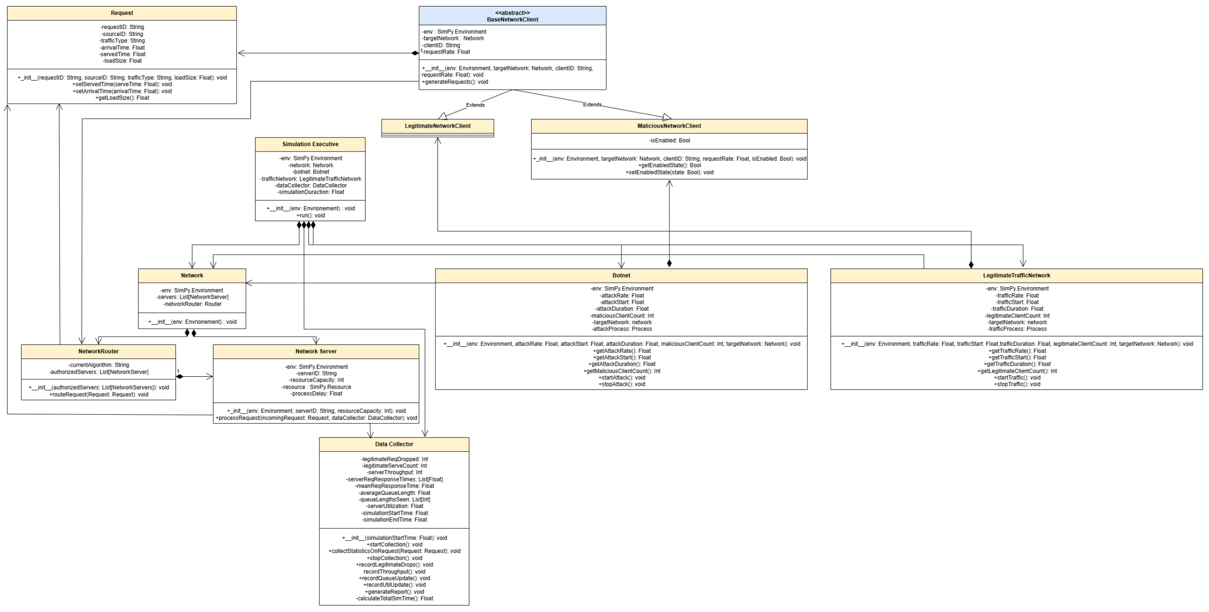
Request
-requestID: String
-sourceID: String
-trafficType: String
-arrivalTime: Float
-servedTime: Float
-loadSize: Float
+_init__(requestID: String, sourceID: String, trafficType: String, loadSize: Float): void
+setServedTime(serveTime: Float): void
+setArrivalTime(arrivalTime: Float): void
+getLoadSize(): Float
<<abstract>>
BaseNetworkClient
-env : SimPy.Environment
-targetNetwork : Network
-clientID: String
-requestRate: Float
+__init__(env: Environment, targetNetwork: Network, clientID: String, requestRate: Float): void
+generateRequests(): void
1
Extends
Extends
LegitimateNetworkClient
MaliciousNetworkClient
-isEnabled: Bool
+_init__(env: Environment, targetNetwork: Network, clientID: String, requestRate: Float, isEnabled: Bool): void
+getEnabledState(): Bool
+setEnabledState(state: Bool): void
Simulation Executive
-env: SimPy.Environment
-network: Network
-botnet: Botnet
-trafficNetwork: LegitimateTrafficNetwork
-dataCollector: DataCollector
-simulationDuraction: Float
+__init__(env: Envrionement) : void
+run(): void
Network
-env: SimPy.Environment
-servers: List[NetworkServer]
-networkRouter: Router
+__init__(env: Envrionement) : void
Botnet
-env: SimPy.Environment
-attackRate: Float
-attackStart: Float
-attackDuration: Float
-maliciousClientCount: Int
-targetNetwork: network
-attackProcess: Process
+__init__(env: Environment, attackRate: Float, attackStart: Float, attackDuration: Float, maliciousClientCount: Int, targetNetwork: Network): void
+getAttackRate(): Float
+getAttackStart(): Float
+getAttackDuration(): Float
+getMaliciousClientCount(): Int
+startAttack(): void
+stopAttack(): void
LegitimateTrafficNetwork
-env: SimPy.Environment
-trafficRate: Float
-trafficStart: Float
-trafficDuration: Float
-legitimateClientCount: Int
-targetNetwork: network
-trafficProcess: Process
+__init__(env: Environment, trafficRate: Float, trafficStart: Float,trafficDuration: Float, legitimateClientCount: Int, targetNetwork: Network): void
+getTrafficRate(): Float
+getTrafficStart(): Float
+getTrafficDuration(): Float
+getLegitimateClientCount(): Int
+startTraffic(): void
+stopTraffic(): void
NetworkRouter
-currentAlgorithm: String
-authorizedServers: List[NetworkServer]
+__init__(authorizedServers: List[NetworkServers]): void
+routeRequest(Request: Request): void
1
Network Server
-env: SimPy.Environment
-serverID: String
-resourceCapacity: Int
-resource : SimPy.Resource
-processDelay: Float
+_init__(env: Environment, serverID: String, resourceCapacity: Int): void
+processRequest(incomingRequest: Request, dataCollector: DataCollector): void
Data Collector
-legitimateReqDropped: Int
-legitimateServeCount: Int
-serverThroughput: Int
-serverReqResponseTiimes: List[Float]
-meanReqResponseTime: Float
-averageQueueLength: Float
-queueLengthsSeen: List[Int]
-serverUtilization: Float
-simulationStartTime: Float
-simulationEndTime: Float
+__init__(simulationStartTime: Float): void
+startCollection(): void
+collectStatisticsOnRequest(Request: Request): void
+stopCollection(): void
+recordLegitimateDrops(): void
recordThroughput(): void
+recordQueueUpdate(): void
+recordUtilUpdate(): void
+generateReport(): void
-calculateTotalSimTime(): Float

Fig. 2. UML class diagram showing the main network simulation classes and relationships with key attributes, methods, inheritance, and composition.

6 GitHub

The simulation implementation can be found at: https://github.com/AMVGH/NetworkDDoSSimulation

References

[1] Kazeem B. Adedeji, Adnan M. Abu-Mahfouz, and Anish M. Kurien. 2023. DDoS Attack and Detection Methods in Internet-Enabled Networks: Concept, Research Perspectives, and Challenges. *Journal of Sensor and Actuator Networks* 12, 4, Article 51 (2023). doi:10.3390/jsan12040051

[2] Chris Conrad. 2022. *Botnets Multiply and Level Up.* DDoS Threat Intelligence Report. NETSCOUT SYSTEMS, INC. Report Number: SEC04.

[3] Deepstrike.io. 2024. DDoS Attack Statistics: 2023-2024. https://deepstrike.io/blog/ddos-attack-statistics. Accessed: 2024-09-17.

[4] Divya Nagpal, Pooja Wadhwa, Sanjay Singh, and Deepak Kumar. 2023. BOTNET: Lifecycle, Architecture and Detection Model. In *Proceedings of the International Conference on Cyber Security and Cryptography.* 112–125.

[5] Ujjwal Sharma, Vikas Kumar, Shubham Kumar, and Dr. Jayasheela. 2024. Routing Algorithms: A Review. *arXiv preprint* (2024). https://arxiv.org/abs/2403.11228v1 arXiv:2403.11228v1.

[6] William Stallings. 2022. *Queuing Analysis in Computer Networks.* Pearson Education.