# Simulating Application Behavior for Network Monitoring and Security

Murugaraj Odiathevar, Kim Chung Yup

Research and Business Foundation, Sungkyunkwan University 2066, Seobu-Ro, Jangan-Gu, Suwon-Si, Gyeonggi-Do, Republic of Korea, 16419 muru.raj@skku.edu, yup.kim@skku.edu

Abstract—Existing network simulations often rely on simplistic models that send packets at random intervals, failing to capture the critical role of application-level behavior. This paper presents a statistical approach that extracts and models application behavior using probability density functions to generate realistic network simulations. By convolving learned application patterns, the framework produces dynamic, scalable traffic representations that closely mimic real-world networks. The method enables rigorous testing of network monitoring tools and anomaly detection systems by dynamically adjusting application behavior. It is lightweight, capable of running multiple emulated applications on a single machine, and scalable for analyzing large networks where real data collection is impractical. To encourage adoption and further testing, the full code is provided as open-source, allowing researchers and practitioners to replicate and extend the framework for diverse network environments.

Index Terms—Network Simulation, Network Analysis, Network Monitoring, Network Data, Statistics, Network Security

#### I. INTRODUCTION

Network simulation plays a crucial role in the design, analysis, and evaluation of computer networks by enabling researchers and engineers to study network behavior without requiring physical infrastructure. By providing a controlled environment for testing, simulation tools help assess network performance, optimize configurations, and detect potential issues before real-world deployment. Popular network simulators, such as Cisco Packet Tracer and NetCracker, offer functionalities for modeling network operations, defining traffic characteristics, and generating statistical reports, making them invaluable for both academic research and industry applications.

Despite these advantages, most network simulation tools primarily focus on modeling network-layer interactions rather than accurately replicating application-level behavior. Traditional simulation approaches typically generate synthetic traffic patterns based on packet-level characteristics without considering the higher-layer dynamics that arise from real application usage. This limitation reduces the realism of simulated network environments, particularly in the context of monitoring, anomaly detection, and security analysis.

To address this gap, this work proposes a fully statistical approach that models network behavior by first simulating application-layer interactions and deriving the corresponding network traffic from them. Unlike conventional methods that rely on predefined traffic models or require captured real-world data, this approach probabilistically generates network traffic based on application behavior, ensuring flexibility, scalability, and reproducibility. Furthermore, it does not require complex hardware setups, making it accessible for both large-scale simulations and lightweight testing environments.

The overarching goal of this work is to generate realistic packet captures (PCAPs) that mimic actual application network flows. By leveraging probability distributions to model application behavior, the generated traffic can be used for network monitoring, security analysis, and anomaly detection research. The following sections provide a detailed discussion of existing network simulation methodologies, their limitations, and how the proposed statistical framework advances the field.

## II. LITERATURE REVIEW

Network simulation is a widely used technique for evaluating communication protocols, security mechanisms, and performance characteristics in a controlled environment. It provides scalability and cost-effectiveness while eliminating the need for physical infrastructure. However, traditional network simulations often lack the realism observed in testbeds that employ real devices and applications [1]. Various network simulation tools exist, each offering distinct capabilities, making selection and implementation a challenge for users [2]. While many frameworks focus on protocol-level interactions, they often overlook application-specific behavior, which plays a crucial role in real-world network performance.

Several simulation frameworks have been developed to bridge the gap between abstract network models and real-world deployments. For instance, the *Network Simulation Bridge (NSB)* facilitates integration between real applications and network simulators, ensuring scalability while minimizing performance overhead [3]. Similarly, the *Network Research Simulator (NRS)* provides a flexible programming interface to conduct experiments on network robustness, including simulations of large-scale attacks and predictive analytics using AI [4]. Despite their contributions, these approaches do not fundamentally model application-level behavior but rather provide simulation support for network-layer functionalities.

Efforts have been made to incorporate real application traffic into network simulations. The use of captured applicationspecific traffic flows allows for the modeling of application performance under different network conditions [5]. Studies on web traffic classification have demonstrated that application-level traffic exhibits distinct statistical properties that can be leveraged for simulation and anomaly detection [6]. However, such approaches rely on empirical data collection rather than a fully statistical modeling approach, making them less adaptable to varying conditions. Furthermore, solutions like Android client emulation [7] provide a means to simulate user interactions but require resource-intensive setups.

A key limitation of existing methodologies is the lack of a true application-layer modeling approach. While application-level traffic data has been correlated with network-level traffic for improved insight [8], most studies still treat applications as static sources of predefined traffic patterns rather than dynamic entities that influence network behavior. Probabilistic models generated from historical network traffic have been proposed to emulate application statistics on virtualized environments [9], but these models do not fully replicate the underlying application behavior. Similarly, simulation tools for enterprise multicast [10] focus on protocol efficiency rather than the behavioral intricacies of applications.

The challenge of integrating real application behavior into network simulations has been long recognized. Existing simulators like NS3 offer extensive functionality for protocol testing, but their credibility is often questioned due to the lack of real-world application interactions [11]. Other approaches attempt to introduce real applications into simulated networks [12], but they require hardware-intensive setups and are not always easily replicable. While existing network simulation techniques provide insights into network operations, they do not fundamentally simulate how applications influence network behavior [3].

Given these limitations, this paper proposes a fully statistical approach that replicates application behavior and builds network behavior from it. Unlike traditional methods, which either simulate network traffic without understanding application behavior or require empirical traffic captures, this approach models application behavior probabilistically using statistical distributions, allowing it to dynamically generate network traffic patterns. This methodology is easily replicable, as it does not depend on complex data collection processes or specialized hardware. Additionally, the statistical nature of the approach makes it *lightweight and adaptable*, enabling its use across different network conditions without significant overhead. By focusing on application-layer behavior as the foundation of network simulation, this work advances the realism and usability of network simulation tools in monitoring, anomaly detection, and performance analysis.

#### III. METHODOLOGY

PCAP (Packet Capture) files are data files used to record and store network packet data captured over a network. They are generated by packet sniffing tools such as Wireshark, tcpdump, or other network monitoring utilities. These files provide a detailed snapshot of network traffic, capturing individual packets along with their metadata, such as source and destination IP addresses, port numbers, timestamps, and payload data and for network analysis, troubleshooting, and security purposes, such as identifying network issues, analyzing communication protocols, and detecting malicious activities. Typical PCAP analysis includes examining packet headers, identifying protocols in use (e.g., TCP, UDP, HTTP), and reconstructing data flows to understand application behavior or detect anomalies. Due to their detailed nature, PCAP files are essential for network engineers, cybersecurity experts, and researchers. Tools such as Scapy [13] library in Python provides a powerful framework for packet crafting, network traffic analysis, testing and with other libraries it can be used to simulate application behaviour. Other tools include the libpcap (Linux/Unix) [14], WinPcap (Windows) [15] libraries.

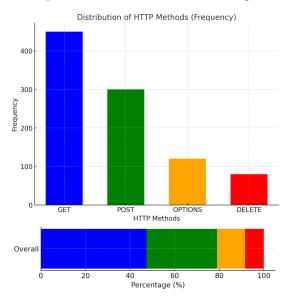
The first step is to monitor the interface to extract pcap data and determine the application behaviour over a period. This can be a few minutes, an hour, a day or a week based on the available memory and patterns. Most applications differ in patterns in weekday vs weekends or holidays, daytime vs night time as they depend on users. It also depends on how much detail is required in modeling the network. Capturing changes by the minute or even hour in most cases is not necessary. Most purposes of network monitoring are to detect when the network can be overloaded through a missed router configuration or a malfunctioning application, and the focus of this paper is on the latter. Applications exhibit distinct network traffic patterns based on their functionalities and user interactions. Understanding these patterns is crucial for effective network management and security. Most applications use the traditional client-server communication (Request-Response). Other types include Push-Based Communications [16], Pub-Sub model [17], fire and forget [18] or broadcast models [19].

Regardless of the method of communication, each application will have a few methods. In the example of "Request-Response" type communication - 'GET', 'POST', 'OPTIONS', 'DELETE'. Aggregating the type for each application, over a period of time, a histogram can be determined. An example of HTTP methods distribution are shown in figure 1. These proportion can be easily obtained after aggregating the pcap files for each application over a period of time. The distribution can then be modeled using a Uniform distribution.

The next step is to determine the volume of data or the payload transmitted over the network for each type of connection for each application. The distribution of the payload can vary in both size and shape. For example, web traffic in http can follow heavy-tailed distribution such as Pareto distribution, file downloads may follow log-normal distribution and VOIP can follow exponential distribution. These patterns have been used for traffic classification [20], [21].

For the simulation, the absolute payload sizes can be modeled using a scaled and shifted Beta distribution. Equation 1 contains four changeable parameters to provide flexibility for different types of absolute payload sizes for each packet. Once the shape of the payload is determined after observing the data, the parameters a, b,  $\alpha$  and  $\beta$  can be determined.

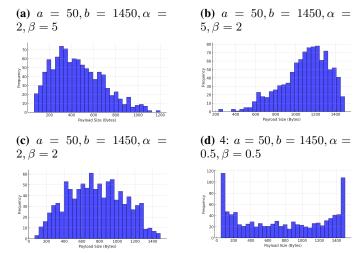
Fig. 1: Distribution of methods (Example)



Figures 2 give four possible shapes for the payload sizes and they can be scaled using the a - minimum payload size observed and a+b - maximum payload size observed. The advantages of using Beta distribution are as follows. (i) can capture skewness (small packets are common, large packets are rare) and variability. (ii) It retains the flexible shape characteristics of the Beta distribution while allowing the data range to shift and scale. (iii)Ensures the output stays within the specified range [a, a+b]. (iv) Many real-world network metrics are bounded and can be effectively modeled using this transformation [22].

$$X \sim a + b \cdot \text{Beta}(\alpha, \beta)$$
 (1)

Fig. 2: Distribution of Payloads (Example)



The next important factor to consider is the maximum number of connections which is determined by the maximum number of users for each application over a period of time. This number can be determined based on the number of unique IP addresses in the data. With these information, a configuration file for each application can be developed as shown in table III. The "Type" in the table is for HTTP requests and a few other TCP packets. These categories are changeable depending on the applications and their protocols.

**TABLE I:** Method Distribution and Connections

Type, M	Probability
GET	0.52
POST	0.12
OPTIONS	0.23
TCP	0.02
TEARDOWN	0.11
Total Connections, N	25
a	1
b	739
$\alpha$	5
β	24

The distribution of the response codes [200, 304, 302, 401, 400, 500, 404, 303] can be modeled using uniform distribution after aggregating the data in a similar fashion to 1,  $W_{response\_code}$ . A list of server ips addresses can be randomly determined for the applications and another much longer list of ip addresses can be generated for client ips. Server port numbers can also be defined likewise. Here is the algorithm.

In our simulation, a 100 application configurations were defined of which 50 applications were randomly selected in each loop. For each type of connection M, a separate function is defined to generate a payload using the Beta distribution. The algorithm can be further refined to generate duplicate packets with r% probability. In our network duplicate packets were detected around 6% of the time. A long session function can be added on top of the methods to generate multiple packets from the same connection M. The simulation runs for a duration  $T_{time}$  but due to the heavy processing and lower computational resources, the code might take longer to run which results in the simulation for the applications starting one after another. Though parallel processing would help, it is still challenging to simulate 50 processes at once. Hence, the time stamp is adjusted for application data in that loop to begin at  $T_{base}$ ; when the first simulation in that "for-loop"

Once the simulation is complete, the pcap can be aggregated using Netflow methodologies [23] and the network statistics can be gathered. Features such as number of concurrent flows per client or server, number of response codes in categories of 200, 400 or 500, packet per second, bytes per second, number of request packets, number of response packets and many more can be determined to analyse the network pattern. These features can be used to build machine learning models for network monitoring and anomaly detection.

## IV. CONCLUSION AND FUTURE WORK

To conclude, the advantages of the simulation methodology are as follows.

Algorithm 1 Simulate Network Traffic and Generate PCAP

```
Require: server_ips, client_ips, app_config, server_ports, T_{\text{time}}
Ensure: output_pcap
 1: Initialize response_codes
 2: Compute response_code_pool using W_{\text{response\_code}}
 3: T \leftarrow 0
 4: while T < T_{\text{time}} do
         Randomly select k applications \{A_{i_1}, \ldots, A_{i_k}\}, k =
 5:
    50
         for application A_i do
 6:
             Retrieve P_{\text{type}}, N_{\text{connections}} from app_traffic_config
 7:
             Generate inter-arrival times \Delta t \sim \text{Exponential}(\lambda)
 8:
             for j \leftarrow 1 to N_{\text{connections}} do
 9:
                 Select M_j \sim P_{\text{type}}
10:
11:
                 Randomly
                                                                assign
    client_ip, server_ip, server_port
                 Generate packets:
12:
                     Handshake: SYN, SYN-ACK, ACK
13:
14:
                     Data Exchange: Response code
15:
                     Payload: Simulate M_i using X
                     Teardown: FIN, FIN-ACK, ACK
16:
                 Update T \leftarrow T + \Delta t
17:
             end for
18:
         end for
19:
20:
         Adjust timestamps to base time T_{\text{base}}
21: end while
22: Sort all_packets by timestamps
    Write packets to PCAP: wrpcap(output_pcap, all_packets)
    return output pcap
```

- 1) Realistic Variation: Through random distributions (exponential inter-arrivals, Beta-based payload sizes), random packet duplication, and method-based flows, the simulation yields complex and lifelike PCAPs.
- Application Profiles: The configuration dictionary ensures each simulated application can have unique traffic characteristics.
- Extensibility: Additional methods (like PUT, HEAD, or custom protocols) can be added with minimal changes.
   The approach is modular enough to integrate new ideas.
- 4) Easy to implement: Through pure statistical distribution and modifying pcap timestamps, large network data can be simulated on relatively small hardware.

The above algorithm implements simulations for TCP connections and it can be complimented with other protocols in future work. Furthermore, the parameters  $a, b, \alpha$  and  $\beta$  and the weights for the uniform distribution can also be determined using machine learning. The simulation can be tested with live network data. Code available on github for open source testing: https://github.com/muru-raj10/AppBehaviourNetworkSim

## REFERENCES

 J. Gomez, E. F. Kfoury, J. Crichigno, and G. Srivastava, "A survey on network simulators, emulators, and testbeds used for research and education," *Computer Networks*, vol. 237, p. 110054, 2023.

- [2] C. Smera and J. Sandeep, "Networks simulation: Research based implementation using tools and approaches," in 2022 IEEE 3rd Global Conference for Advancement in Technology (GCAT). IEEE, 2022.
- [3] H. S. Kuttivelil, S. Sreenivasamurthy, L. Krishnaswamy, N. Bhatia, and K. Obraczka, "Network simulation bridge: bridging applications to network simulators," in *Proceedings of 19th ACM Int'l Symposium on QoS and Security for Wireless and Mobile Networks*, 2023.
- [4] J. L. Marzo, D. Martinez, S. Bergillos, and E. Calle, "Network research simulator an abstract model formulation," in 2022 18th International Conference on the Design of Reliable Communication Networks (DRCN). IEEE, 2022, pp. 1–4.
- [5] D. J. Zacks, T. Szigeti, T. Peleg, D. Tedaldi, and V. V. Pendhar, "Systems and methods for application traffic simulation using captured flows," Mar. 9 2021, uS Patent 10,944,641.
- [6] M. Karayaka, A. Bayer, S. Balkı, E. Anarim, and M. Koca, "Application based network traffic dataset and spid analysis," in 30th Signal Processing and Communications Applications Conference. IEEE, 2022.
- [7] S. N. Hetu, V. S. Hamishagi, and L.-S. Peh, "Similitude: Interfacing a traffic simulator and network simulator with emulated android clients," in 2014 IEEE 80th Vehicular Technology Conference (VTC2014-Fall). IEEE, 2014, pp. 1–7.
- [8] S. Sharma and N. Bhatia, "Correlating network level and application level traffic," Jun. 15 2021, uS Patent 11,038,803.
- [9] T. Ganapathi, S. Raghunath, S. Gal, K. Chandrayana, X. Che, and A. Karapetov, "Data driven emulation of application performance on simulated wireless networks," Jan. 28 2020, uS Patent 10,548,034.
- [10] W. Wu, W. Cao, T. Yan, and Z. Wang, "Ip network multicast technology and application simulation," in 2022 6th International Conference on Wireless Communications and Applications (ICWCAPP). IEEE, 2022.
- [11] S. Rampfl, "Network simulation and its limitations," in Proceeding zum seminar future internet (FI), Innovative Internet Technologien und Mobilkommunikation (IITM) und autonomous communication networks (ACN), vol. 57. Citeseer, 2013.
- [12] L. Mészáros, A. Varga, and M. Kirsche, "Inet framework," Recent Advances in Network Simulation: The OMNeT++ Environment and its Ecosystem, pp. 55–106, 2019.
- [13] R. Rohith, M. Moharir, G. Shobha et al., "Scapy-a powerful interactive packet manipulation program," in 2018 international conference on networking, embedded and wireless systems (ICNEWS). IEEE, 2018.
- [14] L. M. Garcia, "Programming with libpcap-sniffing the network from our own application," *Hakin9-Computer Security Magazine*, vol. 2, 2008.
- [15] A. Xiaoguang and L. Xiaofan, "Packet capture and protocol analysis based on winpcap," in 2016 International Conference on Robots & Intelligent System (ICRIS). IEEE, 2016, pp. 272–275.
- [16] R. C. Sofia and P. M. Mendes, "An overview on push-based communication models for information-centric networking," *Future Internet*, vol. 11, no. 3, p. 74, 2019.
- [17] A.-M. Kermarrec and P. Triantafillou, "XI peer-to-peer pub/sub systems," ACM Computing Surveys (CSUR), vol. 46, no. 2, pp. 1–45, 2013.
- [18] S. Subramaniam and G. H. Loh, "Fire-and-forget: Load/store scheduling with no store queue at all," in 2006 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'06). IEEE, 2006.
- [19] M. Braverman and R. Oshman, "On information complexity in the broadcast model," in *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing*, 2015, pp. 355–364.
- [20] M. Finsterbusch, C. Richter, É. Rocha, J.-A. Muller, and K. Hanssgen, "A survey of payload-based traffic classification approaches," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 2, 2013.
- [21] J. Zhang, Y. Xiang, W. Zhou, and Y. Wang, "Unsupervised traffic classification using flow statistical properties and ip packet payload," *Journal of Computer and System Sciences*, vol. 79, no. 5, 2013.
- [22] S. Gupta and V. Kapoor, Fundamentals of mathematical statistics. Sultan Chand & Sons, 2020.
- [23] R. Hofstede, P. Čeleda, B. Trammell, I. Drago, R. Sadre, A. Sperotto, and A. Pras, "Flow monitoring explained: From packet capture to data analysis with netflow and ipfix," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, pp. 2037–2064, 2014.