Application Agnostic Network Traffic Modeling for Realistic Traffic Generation

by

Oluwamayowa Ade Adeleke

A dissertation submitted to the Department of Computer Science,

College of Natural Sciences and Mathematics

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in Computer Science

Chair of Committee: Deniz Gurkan

Committee Member: Jaspal Subhlok

Committee Member: Edgar Gabriel

Committee Member: Ricardo Lent

University of Houston
December 2020

DEDICATION

Dedicated to God, and to my parents.

Oluwamayowa Ade Adeleke

December, 2020

ACKNOWLEDGMENTS

I would like to especially thank my advisor, Dr. Deniz Gurkan, throughout my Ph.D. program, you never stopped believing in me. You always mentored, encouraged, and pushed me to be a better version of myself. Thanks, Dr. Gurkan. My deep appreciation also goes to our laboratory research professor, Nicholas Bastin. Without your guidance and technical direction throughout my Ph.D., I would not have come so far on this dissertation. Both of you helped nurture what was only a simple idea into the interesting project it has become. I genuinely appreciate you.

My sincere gratitude goes to my dissertation committee members, Dr. Jaspal Subhlok, Dr. Edgar Gabriel, and Dr. Ricardo Lent. Your feedback during and after my dissertation proposal helped me to focus on what was most important.

I would like to appreciate my colleagues from the UH Netlab - Stuart Baxley and Levent Dane. It is a pleasure working with you guys. I also thank my friends who stood by me through all the highs and lows of my Ph.D. program: Ezekiel, Afis, Fela, Oscar, Tolu, Arinze, and Damilola. I cannot thank you enough.

Finally, to my family: my dad, mum, sisters, brothers-in-law, nieces, nephews. I love you. Thanks for being my biggest supporters throughout this process. This Ph.D. is for us all. My most profound appreciation also goes to my uncle Matt, aunt Abiola, Yemisi, Dolapo, and uncle Biola. Your presence and frequent calls throughout my Masters and Ph.D. helped me not to miss home too severely.

ABSTRACT

Research and testing in networking sometimes require experiments that utilize real application network traffic. However, the process of obtaining production network traffic data from industry partners for testing novel algorithms, protocols, and network functions is a significant pain point for many researchers in academia. Many industry operators are reluctant to share network traffic data with third parties to avoid violating privacy policies and avoid unintentional exposure of proprietary information to competitors. Therefore, many researchers resort to the use of synthetic traffic generators in networking experiments.

Our survey of over 7000 networking research papers revealed that most research projects exclusively use constant/maximum throughput traffic generators in their evaluation experiments. These generators do not always generate traffic that is similar to real production traffic. They often blast out packets at fixed rates or rates based on statistical distributions. Existing realistic traffic generators are rarely used, and there is no standardized evaluation system for realistic traffic generators.

Therefore, this work focuses on developing a new application-agnostic framework for producing abstract, high-fidelity models of application network traffic patterns for realistic application traffic generation in laboratory environments. The framework includes a comprehensive evaluation system for realistic traffic generation models. We evaluated the methods and algorithms applied in the framework, then we created and evaluated a new application traffic modeling method that combines clustering methods with stochastic modeling for realistic traffic modeling. The evaluation results reveal that traffic generated is similar to actual production traffic for many types of applications.

This work's outcome is vital to researchers and industry operators in computer networking, especially those involved in large scale enterprise, data-center, and internet of things (IoT) network testing. The methods presented make it easy to investigate how various changes in a network's traffic patterns and infrastructure can impact its performance. Researchers can test new protocols and algorithms with realistic traffic derived from actual applications, without violating privacy policies or replaying extra-large traffic trace files.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

ACRONYMS

ADU Application Data Unit.

CDF Cumulative distribution Function.

CSV Comma Separated Values.

DPI Deep Packet Inspection.

FTP File Transfer Protocol.

GPU Graphic Processing Unit.

HTML5 Hypertext Mark-up Language Version 5.

HTTP Hypertext Transfer Protocol.

HTTPS Hypertext Transfer Protocol Secure.

IOT Internet of Things.

IP Internet Protocol.

IPv4 Internet Protocol Version 4.

IPv6 Internet Protocol Version 6.

ISP Internet Service Provider.

KS-2-sample Kolmogorov Smirnoff Two Sample Test.

LDAP	Lightweight Directory Access Protocol.
MAC	Media Access Control.
MSS	Maximum Segment Size.
MTU	Maximum Transmit Unit.
NFV	Network Functions Virtualization.
NS2	Network Simulator Version 2.
PCA	Principal Component Analysis.
PDU	Protocol Data Unit.
RDP	Remote Desktop Protocol.
SDN	Software Defined Networking.
SSH	Secure Shell.
SVM	Support Vector Machine.
TCP	Transmission Control Protocol.
UDP	User Datagram Protocol.
VLAN	Virtual Local Area Network.
VNF	Virtual Network Function.
WAN	Wide Area Network.

GLOSSARY

Application Data Unit (ADU) Data block sent by an application client or server at a single time through an operating system call.

Application Network Traffic Network traffic packets that are sent or received due to a specific applications process(es).

Capture See Network Trace.

KS-2-sample Refers to Kolmogorov Smirnoff two sample tests for testing the equality and similarity between two empirical distributions.

Network Capture See Network Trace.

Network Trace A file that shows all of the network activity on the wire. It shows the packets flowing into or out of a network interface.

Protocol Data Unit (PDU) Data block sent by a protocol or application operating directly above the transport layer. A single PDU may be broken down into many packets while it is being sent out of an interface.

Trace See Network Trace.

Traffic model A model (often stochastic) of the traffic flows from data sources in a communication network.

PREVIOUSLY PUBLISHED MATERIAL

- Chapters 1, 2, and 4 contain revised excerpts of text from a previous publication [4]: Oluwamayowa Ade Adeleke, Nicholas Bastin, and Deniz Gurkan. “Network Testing Using a Novel Framework for Traffic Modeling and Generation”. International Conference on Computer Communications and Networks. IEEE ICCCN, 2020

- Chapter 3 contain revised excerpts of text from a submitted publication [3] : Oluwamayowa Ade Adeleke, Nicholas Bastin, and Deniz Gurkan. “A Survey of Methods and Outcomes in Network Traffic Generation”. Association for Computing Machinery (ACM) Computing Surveys (CSUR), *submitted in 2020*

- Chapter 1, contain excerpts of text from a previous publication [2] : Oluwamayowa Ade Adeleke, Echo-State networks for network traffic prediction. IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON). Oct. 2019.

Chapter 1

Introduction

The internet has become ubiquitous. Although it started as a small network with wired connections between 4 computers in 4 universities in the western part of USA, it has evolved into a massive web with over 18 billion networked devices and over 3.48 billion users as of 2017, according to the Cisco networking index [24]. The implication is that about half of the population of the world now uses internet based services on daily basis, and the numbers continue to increase every day.

This sustained increase in the internet size and utility continues to ride on the tireless work of researchers in the field of computer networks and distributed computing. Over the last decade, developments in Network Functions Virtualization (NFV) and Software Defined Networking (SDN) [81] have made it possible for academic computer networking researchers to easily create large test networks in laboratory environments. It is no longer surprising to find research experiments where virtual networks with hundreds of virtual hosts, switches and routers are instantiated on a single server [29, 75, 16]. This development has driven a big increase in research output in networking in the last few years. As a result, many new protocols, virtual functions, and hardware which have improved computer networks have been developed in the last decade.

1.1 Problem

In computer networking experiments, tests with real network traffic workloads are crucial for ensuring that the system under test can perform as expected when eventually deployed into production. Although it is often best to test new ideas with actual production traffic, the process for obtaining appropriate network traffic to use for testing novel ideas, algorithms, protocols, and network functions is a major pain point for many researchers. That is, even though researchers can easily create production scale networks in virtualized environments for experiments, it is challenging to do production-scale testing because of the difficulty in obtaining and applying real network traffic from production networks in the test experiments. There are a few reasons for this.

Firstly, privacy policies in many states and countries impose limits on the nature of network data that industry operators can share with third parties [113], including most academic researchers. This is because network traffic often contains sensitive user data that is protected by privacy laws. In addition to this, many industry operators are reluctant to share traffic data with third parties so that they do not unwittingly expose any internal information, systems, or methods to competitors, and thus lose their competitive edge.

Even when privacy policies are not an issue and researchers can obtain actual production traffic for testing, the logistical hurdles of anonymizing and scaling production traffic into a testbed with limited capacity can be daunting for many academic researchers. Furthermore, the process of anonymizing traffic by scrubbing all payload data, IP addresses, and other headers [86] sometimes yield network traces that are devoid of some important header or payload parameters that may be useful for some experiments that require real values for such parameters.

As a result of the factors above, networking researchers have to resort to alternative methods for creating traffic workloads for their experiments. One of the most popular options is the use of synthetic traffic generator applications. Traffic generators are software tools or hardware devices that put synthetically created network packets into a computer network. That is, packets in traffic generators are not generated by actual production application user processes. Thus traffic generators play an important role in computer networking research because they enable experimenters to quickly test new algorithms, protocols, and network-functions with synthetic network traffic.

Synthetic traffic workload is said to be realistic for an application or network environment if it has a decent level of resemblance to real network traffic captured from the application or network environment, in terms of metrics distributions and traffic patterns. There are many categories of synthetic network traffic generators. However, as we show in our survey in chapter 3, many network experiments rely exclusively on constant or maximum throughput traffic generators that do not produce realistic application traffic by default but only blast out packets through a network interface at fixed rates, or maximum possible throughput. Only a few experiments use model-based realistic traffic generators. Furthermore, there are no standards guidelines for how workload is generated for network experiments. This implies that researchers in networking often test new hardware, software, and protocols with synthetic traffic that is not similar to real application traffic in target production networks. This is the main research problem that this work addresses.

1.2 Solution

To tackle the problems described above, in this dissertation, we have developed a high-fidelity modeling framework of novel and existing computer application networking traffic patterns for the purpose of realistic traffic generation. We have also developed a comprehensive evaluation system for measuring the effectiveness of the traffic models.

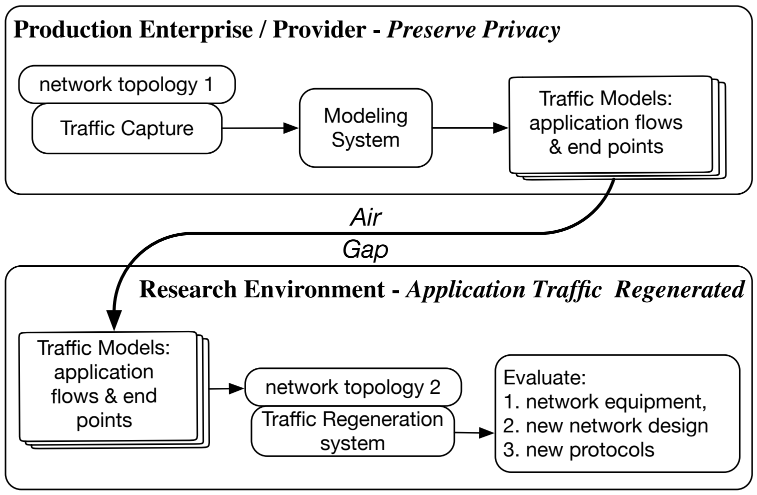

Production Enterprise / Provider - *Preserve Privacy*
network topology 1
Traffic Capture
Modeling System
Traffic Models: application flows & end points
Air Gap
Research Environment - *Application Traffic Regenerated*
Traffic Models: application flows & end points
network topology 2
Traffic Regeneration system
Evaluate:
1. network equipment,
2. new network design
3. new protocols

Fig. 1.1: Privacy preservation with the new modeling and generation framework: traffic models containing no private data, created from a production environment can be taken to a test-bed environment and used to regenerate similar traffic.

The framework can be used in any production network to create traffic models that contain no private data. These models are in simple text format, contain just enough information to enable regeneration of the traffic. They can be shared with any researcher to use in their respective test-bed environments. When the models are shared this way, the actual network traffic data remains exclusively under the control of the production network operator (Fig. 1.1).

This dissertation shows that the hierarchical modeling method based on application data exchange patterns, with parameters at application user level, connection level, and at individual PDU level performs well at generating realistic traffic for many classes of applications. Our evaluation also shows that incorporating high level application network behavior like the use of connection classes, connection pools and request bursts improves the quality of a model in generating realistic traffic.

The framework allows users to add custom modeling methods and corresponding generation methods to the system, allowing them to be evaluated alongside with existing models. The framework is designed to be portable, it can operate on any UNIX based operating system, and does not require a specific test bed environment, neither is it limited to a specific class of traffic.

The modeling system we have developed utilizes machine learning clustering methods along with statistical distributions. It is designed to be effective at modeling any arbitrary network traffic and from any type of application. We have evaluated the modeling method based on diverse sets of application traffic including web application, video streaming, secure shell (SSH), remote desktop, and many others.

This framework is useful to researchers and academics in computer network research, especially those involved in large scale enterprise, data-center, and IoT network testing. The methods presented make it easier to empirically investigate how various changes to a network's traffic patterns, and infrastructure will impact the production network performance using realistic traffic. Industry players, including network operators and equipment vendors can use the framework to demonstrate to operators that their new devices and network functions can successfully handle the specific patterns of traffic that operators have on their production networks. It also enables industry operators to be able to make more informed decisions when designing new networks and optimizing existing networks. Our solution serves as a good foundation for others to build more advanced solutions that apply machine learning to the problem of realistic traffic generation.

1.3 Summary of Contributions

Our contribution in this dissertation includes:

- Traffic modeling framework for traffic generation that uses advanced learning algorithms to process input traffic traces to provide privacy-preserving models
- A new application-agnostic traffic model generation method based on application data exchange patterns, incorporating representations for high level application behavior including

true transport layer semantics, Connection classes, connection pools, and request bursts.

- A comprehensive evaluation showing that the modeling method above is effective and application agnostic, and that the incorporation of high level application network behavior (connection classes, connection pools and request bursts) improves the quality of the model in generating realistic traffic.

- The outcomes of our extensive survey on about 100 network traffic workload generators used in computer networks research, and our analysis of their usage in over 7000 networking papers

- A set of tools and a recommended methodology to guide experimenters in selecting appropriate network traffic generators for specific experimental use-cases based on required features, and recorded metrics.

1.4 Organization of the Dissertation

In chapter 2 we provide background literature on network traffic workload generators, their evaluation, and usage scenarios that is relevant to the body of this dissertation. Afterwards in chapter 3 we share the results of our extensive survey on workload generation techniques in computer networks research as the groundwork on evaluation of traffic generation needs. Details for traffic modeling, generation, and evaluation are in chapter 4. In chapter 5 we provide the details of our evaluation experiments, and a discussion of the results showing the benefits of our new modeling approach.

Chapter 2

Literature Review

In computer network research, tests on new artifacts (algorithms, network functions and protocols) are rarely ever done in actual production environment, because of the high risk factor involved. Any mistake in the design or implementation of the new artifact can disrupt a production network and adversely impact users, administrators and even other connected networks. Hence before any new network artifact is deployed in production, it usually undergoes many stages of testing in laboratory and test bed environments. When carrying out tests in the laboratory, the type of network traffic workload used in experiments matter. The type of traffic used can positively or negatively impact experimental results. One common objective of many test processes is to have an estimate of how the new artifact will perform when deployed in a new production environment. To achieve this objective, some tests must be carried out using traffic that is typical to the target production network. However, as described in section 1.1 researchers usually still have to rely on traffic generators for testing these artifacts. Therefore, traffic generators that produces traffic workload that is realistic for the target deployment network must be used. This is why realistic traffic generators play an important role in network experimentation. The next section explores common classes of network traffic generators, providing a few examples of generators in each category.

2.1 Traffic Generators Classification

In a 2013 paper by Molnár et al. [83], the authors provided a classification for network traffic generators from the perspective of their techniques for pushing packets into the network. We have expanded on this classification in the section below. In general, traffic generators can be categorized into constant / maximum throughput generators, application-level synthetic workload generators, trace file replay systems, model-based generators, and script driven traffic generators. We describe each class below.

2.1.1 Constant or Maximum Throughput Generators (CMT)

Traffic generators in this category typically create a packet with specific header fields. The packet is then repeatedly sent out of a network interface at a constant rate or the maximum possible rate in bits per second (bps) or packets per second(pps). Popular examples of traffic generators in this category are `iperf` [128] and `netperf` [58]. Other examples include `nuttcp` [42] and `SolarwindsWANkiller` [119]. These are often the easiest to use, and are suitable for quick network throughput stress testing. A characteristic of generators in this category is that they offer little or no variation in header and payload content of the packets blasted out of the interfaces. In most traffic generators in this class, a user can specify only one flow per native run instance such as the source and destination IP address and port numbers before the packet generation process starts.

2.1.2 Application Level Synthetic Workload Generators

These generate network packet traffic for a specific type of application or protocol such as the `httperf` [84]. In some cases, researchers may launch actual application programs and run a specific set of workloads using their data exchanges to generate the traffic. This approach of workload generation is often capable of realistic variations on packets for the specific application or protocol. However, the resulting workload still consists of a limited set of application events. Some other popular examples of traffic generators in this category include `httperf` [84], `Surge` [14] and `packmime-http` [140]. Exclusive usage of these approaches may result in a skewed performance deprived of the realistic simultaneous background traffic or the application-user interaction in a typical production environment [136].

2.1.3 Trace File Replay Systems

Replay systems inject packets from a trace file into a network interface at the indicated time intervals in the capture file. In some cases, users are able to specify the speed at which they would like to replay the packets. Many researchers obtain trace files with anonymous data and empty payload contents from public data sets [109, 44, 72], and replay them on the nodes of their individual experiment topologies using tools like `TCPreplay` [130]. These replay systems can produce traffic workloads that mirror the original traffic, especially if the workload can be run on an experiment topology that is similar to the original network. In addition, most replay systems are stateless and are unable to send the packets in a manner that will be responsive to the impairments in the experiment network. For example, such a replay will continue to send out TCP packets even when the links between endpoints are down whereas a realistic TCP flow control would have limited further packet transmissions. In addition, continuous replay of the same trace file on a network will keep producing the same events periodically resulting in an unrealistic traffic pattern. Other common examples of replay systems include `TCPIVO` [41], `EAR replay` [69], `Bit-twist` [147] and `divide and conquer` [146].

2.1.4 Script Driven Traffic Generators

In recent years many new script driven traffic generators have been developed. These generators allow users to dynamically modify the full range of packet header and data content. Popular examples of generators in this category are `DPDK pktgen` [142] and `moongen` [37]. These allow users to create any type of packet, with almost any packet header value, and while also dynamically modifying the packets at run time.

2.1.5 Model-based Traffic Generators

A popular method of generating realistic traffic is the creation and transmission of packets following random distributions of their time intervals, packet sizes, etc. One example of these is the Multi-Generator (MGEN) [88] traffic generator. Other examples include D-ITG [10], and Brute [21]. These generators allow users to specify a random distribution model with parameters that may match the intended scenario of network traffic workload. With carefully selected random distributions, they can generate traffic that is statistically similar to traffic workload in specific production environments.

2.1.6 Trace Driven Model-based Traffic Generators

Some traffic generators go a step further than the purely model-based approach by allowing experimenters to supply a trace file input or log files of actual traffic from production networks. The input trace file or log file is analyzed to create a model by fitting the various traffic parameters to random distributions which are then used to generate packets. Some examples are `harpoon` [120] and `swing` [137]. Other examples include `TMIX` [141], sourcesonoff [134], Litgen [108] and reneto [45]. They generate packets that are statistically similar to actual packets seen in the corresponding input production network trace or log file.

2.2 Traffic Modeling and Realistic Traffic Generators

In networking research experiments that are designed to evaluate how a device or software would perform in a real production network environment, it is important that such experiments are carried out using network traffic that is realistic for the target production network. When the researchers are unable to obtain and replay actual production traffic due to the privacy policies described earlier in section 1.1, they often have to resort to one of the synthetic work load generation methods described in section 2.1 above. However, in this kind of experiments, traffic generators with better realistic traffic modeling methods will yield better evaluation results than constant or maximum throughput generators. Hence an understanding of network traffic modeling research is crucial in any realistic traffic generation effort.

One important characteristic of realistic traffic is the high variability in source and sink endpoints, applications, and packet header parameters. When researchers carry out tests with only simple traffic generators, the limited variation in packets seen might cause a wrong prediction of expected performance measurements, because non-realistic traffic generators will not expose new protocol or network function to entire range of possible traffic mixes that will be seen in production. Network traffic flowing through a link or an interface at any point is made up of a summation of all packets flowing between all the sources and sinks that are connected to that point of the network, with each source and sink pair typically having multiple applications that exchange network traffic. Hence, the traffic observed at any point can be taken as the aggregate of all traffic from every application running on all sources and sinks in the network. A good realistic traffic model must create good representations for how applications and endpoints on a target network exchanges network traffic.

Since the early days of computer networks, most network traffic modeling efforts have been focused on getting accurate representations for packet the inter-arrival times parameter. Initially, many efforts were focused on the use of non-self-similar distribution models for inter-packet times [145]. In the eighties and early nineties, most traffic models were based on the simple Poisson distribution [79]. The simple Poisson model was popular because of its ease of use, and because it was assumed that burstiness in observed values of inter-packet times could be smoothed by aggregating traffic from a large number of sources, in adherence with the central limit theorem. Other possion-based models were later developed in an attempt to account for the burstiness observed. Some examples of such models include the compound Poisson traffic model [57], the Markov-modulated Poisson traffic model (MMPP) [51] and the packet trains [57].

In the early 1990s, a large scale network traffic data collection and analysis effort yielded results that clearly showed evidence of self-similarity and long range dependence in real Ethernet traffic [73, 143]. However, Poisson models are not self similar in nature, making them inadequate in modeling traffic for packet generation in simulations. This was also proven experimentally by Paxson et. al. [99]. They demonstrated that packet inter-arrivals could not be adequately modeled with Poisson processes.

Since that time, many distributions have been developed that are better at modeling network traffic. A Pareto distribution was found to be suitable in modeling real network traffic inter-arrival times, even though the Pareto distribution is not inherently self-similar, as it exhibits visual self-similarity due to its heavy tail [99]. Other self similar models include fractional Brownian motion traffic model [92] and Chaotic Maps traffic models [38].

2.3 Common Realistic Traffic Generators in Research

In the application of traffic models to realistic packet generation, the typical approach is to create a system that identifies various characteristics of network traffic and create stochastic distributions to represent values observed for each of the characteristics. There are many realistic traffic generators that follow this approach [71, 10, 120, 137]. These generators fall into the category of model-based traffic generators and trace driven model-based traffic generators which we describe in sections 2.1.5 and 2.1.6 respectively. Each of them generates traffic based on a model containing distributions for differing sets of parameters. By incorporating multiple sources and sinks based on the hierarchical models, some of them are able to provide good self similarity characteristics. In this rest of this section we provide brief descriptions and references for some common realistic traffic generators in research.

D-ITG: The distributed internet traffic generator (D-ITG) [10] is a traffic generator that allows users to specify statistical distributions for network traffic parameters including inter-packet times and packet sizes. It then generates network traffic from a source, based on the distributions.

Harpoon: Harpoon [120] is a flow-level TCP traffic generator for router and network testing that extracts statistical distributions for 7 traffic parameters from router netflow logs, and then generates packets based on these statistical distributions.

TMIX: TMIX [141] generates realistic TCP application workloads in NS-2. The traffic generator consists of two applications, the first extracts connection vectors (details of Application Data Units) from the traces. The second application takes the connection vectors as input, and generates packets, either by replaying the pattern seen in the connection vectors, or by using calculated values after fitting the connection vectors to statistical distributions.

Swing: Swing [137] is a realistic and responsive network traffic generator that uses a hierarchical model to extract probabilistic distributions for many parameters from traffic traces at user level, request-response exchange level, connection level and packets level. It attempts to replicate both the traffic and observed network conditions in a laboratory. The generator is able reproduce the burstiness characteristic observed in self similar internet traffic, by recreating original network conditions in a closed generation environment.

SourcesOnOff: Sourcesonoff [134] uses multiple source and sink processes to generate packets, with each source having periodic on and off packet trains to create self similar traffic, as suggested by Jain et al. [57] and Willinger et al. [144]. The tool also allows users to select statistical distributions to be used for packet sizes and inter-arrival times in the multiple sources.

Litgen: Litgen [108] automatically extracts statistical distributions from traces for the multiple sources at session, object and packet levels to later generate traffic based on values from these distributions.

Reneto: Reneto [45] is made up of two distinct applications, a traffic generator for the OMNeT++/INET simulator. First, A trace analyzer that extracts packets to fit distributions for 13 hierarchical parameters at UDP/TCP, Flow, Session, and user levels, to create an XML file containing the model. Second, a traffic generator application to generate traffic based on the XML file.

Brute: Brute [21] allows users to write code extensions to define custom statistical models. These code extensions can then be used to generate packets in a live network.

RAMP: RAMP [71] utilizes traces to estimate end-user behavior and network conditions, it then generates a 3-level structural source model for web and FTP traffic and uses the models to generate realistic traffic.

SURGE Scalable URL Reference Generator (SURGE) [14] is a realistic traffic generator for web (HTTP) traffic. It creates a model with distributions obtained from empirical measurements for 6 parameters, and uses this model to generate packets at desired sources and sinks.

2.4 The Need for a New Realistic Traffic Generation Framework

In our survey of traffic generators and their use in research (section 3.3), we observed realistic traffic generators are not as popular as constant or maximum throughput traffic generators in terms of usage in research experiments. We believe there are many reasons for this. Some of them are complicated to use, some are too tightly bound to specific test bed environments [141, 45]. Others require that all nodes are connected with an out of band control network [137]. In addition, since network traffic typically evolves over time as newer applications are created, some of these generators have modeling methods that are no longer valid for the nature of traffic seen in typical networks today.

Our work in this research is unique, and different from all those described in section 2.3 in that our focus is on creating a framework, and not just another traffic modeling and generation methods. The framework makes it possible for users to continuously add custom modeling and generation methods for desired application network traffic in response the evolving nature of network traffic.

In addition to this, the modeling system is designed to be portable to all operating systems that support the python programming language, and it does not require a specific test-bed or out of band control network. Ease of use was a major guiding principle in the design of the framework, thus, the framework design makes it possible for a user to select from a library of ready made traffic model files for a specific application or network, and use the model to generate traffic in a test environment without having to go through the more tasking process of model creation. By decoupling the modeling system from the generation system, the framework also ensures compliance with any privacy policies, if the model files are created within a production network operator domain, and shared with third parties via other channels.

Using the framework, we also developed a new modeling method from realistic traffic generation. The modeling is based on 2-way client-server data exchanges as seen in real TCP and UDP traffic, unlike most of the realistic traffic generators where modeling is done based on 1-sided flows. Furthermore, the modeling method is unique in its hierarchical structure in that it includes a machine learning based clustering of application connections, in addition to the usual stochastic distribution based modeling seen in other method.

In the next chapter we share the details of our survey on network traffic generation methods before we go on to provide the details of our framework in the subsequent chapter.

Chapter 3

Survey of Traffic Generators

There are a plethora of tools to generate network traffic workloads with different implications on the measurement outcomes. As a result of the diverse set of options available, researchers tend to generate traffic using varying methods for even similar experimental goals. In fact, in many cases, distinct traffic generation methods report separate sets of metrics for similar traffic workloads, thus making comparisons between experiments difficult and hindering repeatability of outcomes.

A workload generation approach to a networking research experiment has to supply the behavior and the means to validate the desired characteristics. There is not any one particular traffic generation method that is capable of answering the needs of all types of experiments. Hence, a comprehensive assessment of traffic generator features and possible usage scenarios are important steps in experiment design.

In this chapter, we present a comprehensive survey of network traffic generators in academia and industry. Unlike existing traffic generator surveys [36, 66, 83], our objective is not a performance comparison, rather a determination of the behaviors. The performance of traffic generators has been studied extensively in the literature and our survey focuses on the types of variances and functionality of the available traffic generators even though it is possible that they could be run in a high performance setting with the support of hardware platforms and techniques guaranteeing a wire-speed generation capability. In fact, most generators in our analysis are software programs that are vulnerable to the limitations in the runtime environment and the hardware systems. Our goal is to analyze available characteristics and features of commonly preferred tools and then to provide a systematic methodology to pick suitable generators for the type of research goals.

We first present our survey of traffic generators and their usage in a comprehensive set of publications in the top ACM and USENIX conferences where we collect information on usage frequency of a traffic generation method of any kind. (IEEE publications were not included in our corpus because the API to pull papers from the IEEE database made it difficult to perform extensive downloads of a large number of papers.) We compile almost 100 traffic generators used in academia and industry. For each one we attempt to obtain the binaries or the source code and then study available documentation or reference papers. We investigate and then categorize each as either dormant or active based on their last-reported maintenance date. We then group the generators into classes based on how they function. Afterwards, based on the usage scenarios in papers from prestigious networking research conferences over the last 13 years (over 7000 papers), we rank them per popularity. We then pick the top 10 to carry out a closer examination of the individual features. Based on the literature analysis, key observations are as follows:

1. Most research papers have used constant/maximum throughput traffic generators. These traffic generators blast packets at a given rate with no specific application behavior or header field programmability.

2. Only a few research projects use realistic (model-based) traffic generators.

3. Traffic generators produce output metrics in a non-standard manner.

In the next sub-sections, we provide the details of our survey of traffic generators, and their usage in research and academia. We then focus on specific features and metrics provided in each of the top 10 traffic generators. We also present a methodology to help researchers in choosing appropriate traffic workload generators based on their experiment needs.

3.1 Comprehensive List of Traffic Generators

We assembled an exhaustive list of network traffic generators used across research and industry, finding 96 traffic generators created between 1995 and 2018. Our list of traffic generators was sourced from computer networking research papers (over 7000 papers in published by ACM SIGCOMM and USENIX [132]) and general internet document searches [111, 112, 127, 49].

3.1.1 List of Traffic Generators

The Table 3.1 below lists all 96 traffic generators we considered in this survey. We have included the information on licensing, software maintenance status, supported operating systems, the generation category as outlined in the taxonomy section 2.1, and the best available web link to get further information about each traffic generator. The generators in the table have been listed in descending order of popularity based on our findings in the section 3.3.

Table 3.1: Status of traffic generators in research and industry as of Jan 2019. Generators sorted on descending order of popularity per section 3.3

ID	Name	License	Status Date	Platform[5]	Category	Link (Source, Binaries, Paper)
1	iPerf2 [128]	BSD	2019-01	All	CMT[2]	`https://sourceforge.net/projects/iperf2/`
2	Netperf [58]	Free[1]	2018-06	All	CMT[2]	`https://hewlettpackard.github.io/netperf/`
3	Httperf [84]	GPLv2	2018-11	All	App level gen	`https://github.com/httperf/httperf`
4	Moongen [37]	MIT	2018-12	Unix, Linux	Script driven	`https://github.com/emmericp/MoonGen`

Table 3.1. (continued). Status of traffic generators in research and industry as of Jan 2019. Generators sorted on descending order of popularity - per section 3.3

ID	Name	License	Status Date	Platform	Category	Link (Source, Binaries, Paper)
5	Scapy [20]	GPLv2	2019-01	All	Script driven	`http://www.secdev.org/projects/scapy/`
6	Linux Pktgen [93]	GPLv1	2018-09	linux	Script driven	`https://github.com/torvalds/linux/blob/master/net/core/pktgen.c`
7	Netcat [53]		2019-01	All	Other[3]	`http://nc110.sourceforge.net/`
8	iperf3 [33]	BSD-3-Clause	2018-12	All	CMT[2]	`https://github.com/esnet/iperf`
9	TCPreplay [130]	GPLv3	2018-12	All	Traffic replay	`http://tcpreplay.appneta.com/`
10	DPDK Pktgen [142]	BSD	2019-01	Unix, Linux	Script driven	`https://pktgen-dpdk.readthedocs.io/en/latest/`
11	Harpoon [120]	GPLv2	2018-01	Unix, Linux	Trace driven	`https://github.com/jsommers/harpoon`
12	D-ITG [10]	GPLv3	2013-03	All	Model based	`http://www.grid.unina.it/software/ITG/`
13	TMIX [141]	MIT	2011-11	NS2 or NS3	Other[3]	`https://github.com/weiglemc/tmix-ns2`
14	Nuttcp [42]	GPLv2	2018-07	All	CMT	`https://www.nuttcp.net/`
15	SWING [137]	Free[1]	2008-09	Unix, Linux	Trace driven	`http://cseweb.ucsd.edu/~kvishwanath/Swing/`
16	Surge [14]	Free[1]	1998-11	All	App level gen	`http://cs-www.bu.edu/faculty/crovella/surge_1.00a.tar.gz`
17	OSNT [8]	-	2019-01	NetFPGA	Script driven	`http://osnt.org/`
18	Bit-Twist [147]	GPLv2	2012-04	All	Traffic replay	`http://bittwist.sourceforge.net/`
19	Globetraff [62]	-	2016-09	All	Trace driven	`https://github.com/lookat119/GlobeTraff`
20	Ixnetwork [65]	Commercial	-	-	-	`https://www.ixiacom.com/products/ixnetwork`
21	UDPgen [124]	-	-	-	-	`http://www.fokus.fhg.de/usr/sebastian.zander/private/udpgen`
22	Nping [91]	GPLv2	2018-03	All	CMT	`https://nmap.org/nping/`

Table 3.1. (continued). Status of traffic generators in research and industry as of Jan 2019. Generators sorted on descending order of popularity - per section 3.3

ID	Name	License	Status Date	Platform	Category	Link (Source, Binaries, Paper)
23	TRex [25]	Apache-v2	2019-01	Unix, Linux	Script driven	`https://trex-tgn.cisco.com/`
24	Ostinato [123]	GPLv3	2019-01	All	Script driven	`https://ostinato.org/`
25	Libcrafter [101]	MIT	2017-09	Unix, Linux	Script driven	`https://github.com/pellegre/libcrafter`
26	PackMime-HTTP [140]	MIT	2005-06	NS2	App level gen	`https://www.cs.odu.edu/~mweigle/research/packmime/`
27	Spirent SmartBits [122][6]	Comm-ercial	-	-	-	`https://www.spirent.com/products/testcenter`
28	Nemesis [87]	GPLv2	2003-11	All	Script driven	`http://nemesis.sourceforge.net/`
29	IPB [77]	-	-	-	-	`https://ieeexplore.ieee.org/abstract/document/693678`
30	LANforge FIRE [22][6]	Comm-ercial	-	All	-	`http://www.candelatech.com/`
31	GenSyn [50]	-	-	-	-	`https://www.researchgate.net/publication/234056921_GenSyn-a_Java_based_generator_of_synthetic_Internet_traffic_linking_user_behaviour_models_to_real_network_protocols`
32	Mtools [9]	-	-	-	-	`http://www.grid.unina.it/grid/mtools/`
33	Netspec [59]	-	1997-12	Unix, Linux	Trace driven	`http://www.ittc.ku.edu/netspec/`
34	Packet Shell [124]	-	-	-	-	`http://pksh.tecsiel.it/`
35	Skaion TGS [116]	Comm-ercial	-	-	Other	`http://www.skaion.com/`
36	Trafgen [52]	GPLv2	2019-01	Unix, Linux, Mac	App level gen	`http://netsniff-ng.org/`
37	RAMP [71]	-	-	-	Trace driven	`http://www.csie.ncku.edu.tw/~klan/data/materials/ramp.pdf`
38	BRUTE [21]	GPLv2	2016-11	Linux	CMT[2]	`https://github.com/awgn/brute`

Table 3.1. (continued). Status of traffic generators in research and industry as of Jan 2019. Generators sorted on descending order of popularity - per section 3.3

ID	Name	License	Status Date	Platform	Category	Link (Source, Binaries, Paper)
39	Breaking-Point [63]	Comm-ercial	-	-	App level gen	`https://www.ixiacom.com/products/breakingpoint-ve`
40	IP-Packet [12]	GPLv2	2003-11	Linux, FreeBSD	CMT[2]	`http://p-a-t-h.sourceforge.net/html/index.php`
41	Rude/ Crude [70]	GPLv2	2002-06	All	CMT[2]	`http://www.atm.tut.fi/rude`
42	Bruno [7]	-	-	-	Trace driven	`https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4667607`
43	Divide & Conquer [146]	-	-	-	Traffic re-play	`https://ieeexplore.ieee.org/document/1386202`
44	Byte-Blower [39]	Comm-ercial	-	-	-	`https://www.excentis.com/products/byteblower/`
45	Colosoft Packet Builder [56]	Free[1]	2016-06	Windows	CMT[2]	`http://www.colasoft.com/download/products/download_packet_builder.php`
46	EAR Re-play [69]	-	-	-	Traffic re-play	`ieeexplore.ieee.org/abstract/document/6214199/`
47	GL traffic generator [47]	Comm-ercial	-	-	-	`https://www.gl.com/traffic-generators.html`
48	HexInject [1]	BSD-2-Clause	2017-01	Linux	CMT[2]	`http://hexinject.sourceforge.net/`
49	IPGen [74]	-	2001-03	-	CMT[2]	`http://sourceforge.net/projects/ipgen/`
50	IxChariot [64]	Comm-ercial	-	-	Trace driven	`https://www.ixiacom.com/products/ixchariot`
51	PIM-SM Packet Generator [5]	-	-	-	-	`https://literature.cdn.keysight.com/litweb/pdf/5988-6560EN.pdf?id=1649878`
52	EPB [135]	Free[1]	2019-05	All (C)	Script driven	`http://m-a-z.github.io/epb/`
53	NETI@ home [114]	-	-	-	-	`http://neti.gatech.edu/`

Table 3.1. (continued). Status of traffic generators in research and industry as of Jan 2019. Generators sorted on descending order of popularity - per section 3.3

ID	Name	License	Status Date	Platform	Category	Link (Source, Binaries, Paper)
54	TTCP, Test TCP [26]	-	-	-	-	`https://www.cisco.com/c/en/us/support/docs/dial-access/asynchronous-connections/10340-ttcp.html`
55	LANTraffic [27]	Commercial	2015-11	Windows	CMT^2	`https://www.zti-communications.com/lantrafficv2/`
56	Libtins [43]	BSD	2019-01	All	Script driven	`https://github.com/mfontanini/libtins`
57	LitGen [108]	-	-	-	Trace driven	`https://www.researchgate.net/publication/220850223_LiTGen_a_Lightweight_Traffic_Generator_Application_to_P2P_and_Mail_Wireless_Traffic`
58	MGEN [89]	MIT-ish	2018-11	All	Model based.	`https://www.nrl.navy.mil/itd/ncs/products/mgen`
59	UDP Generator [125]	MIT	1999-05	Linux, Unix	CMT^2	`http://www.citi.umich.edu/projects/qbone/generator.html`
60	Network Expect [35]	-	-	Unix, Linux, Mac	CMT^2	`http://www.netexpect.org/`
61	Cat Karat [31]	Commercial	2010-01	Windows	CMT^2	`https://sites.google.com/site/catkaratpacketbuilder/`
62	NTG [149]	-	-	-	App level gen	`http://www.wseas.us/e-library/conferences/2013/Paris/CCTC/CCTC-35.pdf`
63	Fragout [121]	BSD-3-Clause	2002-04	All	CMT^2	`http://www.monkey.org/~dugsong/fragroute/`
64	GEIST [60]	BSD-2-Clause	2012-11	All	Model based	`http://kkant.net/geist/`
65	NTGM [100]	Commercial	2018-10	Windows	CMT^2	`http://pbsftwr.tripod.com/id17.html`
66	Graph-Based TG [115]^4	-	-	-	-	`http://rvs.unibe.ch/research/pub_files/SSKB10.pdf`

Table 3.1. (continued). Status of traffic generators in research and industry as of Jan 2019. Generators sorted on descending order of popularity - per section 3.3

ID	Name	License	Status Date	Platform	Category	Link (Source, Binaries, Paper)
67	Inter-networking Test TG [32]^4	-	-	-	-	`http://www.donfraysoftware.com/MITS/MITS.htm`
68	Omnicor TG [94]^4	Commercial	-	-	Model based	`https://www.omnicor.com/products/network-testing-tools`
69	Jugi's TG [78]^4	GPLv2	2010-11	Linux	CMT^2	`http://www.netlab.tkk.fi/~jmanner/jtg.html`
70	KUTE [150]	GPLv2	2007-09	Linux	CMT^2	`http://caia.swin.edu.au/genius/tools/kute/`
71	LAN-decoder32T [129]	-	-	-	-	`http://www.triticom.com/triticom/ld32/trafgen.htm`
72	packet sender [85]	GPLv2	2018-12	All	CMT^2	`https://packetsender.com/`
73	PackETH [98]	GPLv3	2017-12	All	CMT^2	`http://packeth.sourceforge.net/`
74	Mausezhan [139]	GPLv2	2011-12	Linux (C)	CMT^2	`https://github.com/uweber/mausezahn`
75	MxTraf [68]	GPLv2	-	-	-	`http://mxtraf.sourceforge.net/`
76	Solarwinds WAN killer [118]	Commercial	-	Windows	CMT^2	`https://www.solarwinds.com/topics/traffic-generator-wan-killer`
77	NSWEB [138]	-	-	NS2	-	`https://www.net.t-labs.tu-berlin.de/~joerg/`
78	NTGen [11]	-	2002-11	Linux (C/C++)	-	`http://softlab-pro-web.technion.ac.il/projects/NTGen/html/ntgen.htm`
79	STG-10G [34]	Commercial	-	-	Model based	`https://www.ecdata.com/products/stateful-traffic-generator/`
80	PacGen [96]	GPL-v2	2006-09	Linux (C)	CMT^2	`http://sourceforge.net/projects/pacgen/`
81	PlayCap [95]	GPLv3	2010-03	All	Traffic replay	`https://github.com/signal11/PlayCap`
82	Poisson TG [107]^4	-	2003-06	(C)	Model based	`http://www.spin.rice.edu/Software/poisson_gen/`
83	ProvaGEN 3.0 [124]	-	-	-	-	`http://www.provanet.com/packet_generator_tts_page.htm`

Table 3.1. (continued). Status of traffic generators in research and industry as of Jan 2019. Generators sorted on descending order of popularity - per section 3.3

ID	Name	License	Status Date	Platform	Category	Link (Source, Binaries, Paper)
84	Qosnetics TG [111][4]	-	-	-	-	`http://www.qosnetics.com/`
85	Real-Time Voice TG [110][4]	-	-	-	-	`http://www.cs.ucr.edu/~msamidi/projects.htm`
86	VOIP TG [17][4]	-	2005-11	(perl)	App level gen	`http://voiptg.sourceforge.net/`
87	Self Similar TG [67][4]	MIT	2001-04	(C)	Model based	`http://research.glenkramer.com/code/trf_gen3.shtml`
88	Sources-OnOff [134]	GPLv3, CeCILL	2013-03	Linux (C)	Model based	`http://www.recherche.enac.fr/~avaret/sourcesonoff`
89	SPAK [61] Packet Generator	-	-	-	-	`http://static.lwn.net/lwn/1998/0312/a/spak.html`
90	TCPivo [41]	-	2002-09	Linux (C)	Traffic replay	`https://www.thefengs.com/wuchang/work/tcpivo/`
91	TfGen [126]	-	1998-02	Windows	CMT[2]	`http://www.pgcgi.com/hptools/`
92	IP-traffic [97]	Commercial	2019	Windows	CMT[2]	`https://www.pds-test.co.uk/products/ip_test_measure.html`
93	Traffic Generator Tool [102]	-	-	-	-	`http://www.postel.org/tg/`
94	WRAP [90]	BSD clause	2019-01	All	Script driven	`https://github.com/Juniper/warp17`
95	Yersinia [15]	GPLv2	2017-09	All	CMT[2]	`https://github.com/tomac/yersinia`
96	YouTube Workload generator [23]	-	-	-	-	-

Table 3.1 Footnotes

1 Some traffic generators classified as free require attribution

2 CMT stands for constant or maximum throughput traffic generators (see section 2.1)

3 Other, when not listed in the pre-defined traffic generator categories in section 2.1

4 TG is used as an abbreviation for traffic generator

5 Platform refers to supported operating systems

6 Hardware traffic generators, all others are software traffic generators

3.2 Tool Availability

For each of the generators on our list we evaluated the distribution status - whether there were freely available binaries or source code, or trial variants. We found that thirteen generators required commercial licenses, and only two of those had available trial versions. This includes all of the hardware traffic generators on our list and a small number of software traffic generators. For the remaining non-commercial generators we were unable to locate source code or binaries for twenty-nine of them, leaving us with source code or binaries in full or as a free trial version for only fifty-six of the generators.

For these fifty-six generators, we obtained the latest date of development update (code change or package release) to gain a sense of maturity and development activity. Twenty-five of the generators have not been updated since 2012.

We provided a taxonomy for the generators in the section 2.1 with a reference to one of the listed categories in the Table 3.2 below. There were a few of the generators that did not fit into any of the specified categories (marked as "other" in the table). We were unable to verify the category of the rest because they were either no longer unavailable or required commercial licences that we could not obtain. The complete detail of these findings are in the Table 3.1.

Table 3.2: Classification of traffic generators per section 2.1

	Traffic Generator Category	All
1	Constant or maximum throughput generators	26
2	Application level generators	7
3	Trace file replay tools	7
4	Model-based traffic generators	11
5	Trace driven model-based traffic generators	5
6	Script driven traffic generators	11
7	Others	29
	Total	96

3.3 Traffic Generation Tool Popularity

We collected the 96 traffic generators listed in the Table 3.1 based on their usage in papers published over the last 13 years (from 2006 to 2018). We examined a total of over 7000 computer networking related papers, including 2762 papers published in various conferences and journals by the Association for Computing Machinery's (ACM) Special Interest Group on Data Communications (SIGCOMM). The ACM conferences we explored include ACM-ICN, AllThingsCellular, ANRW, APNet, CHANTS, Cnet, GreenNets, HomeNets, HotNets, HotSDN, IoTS&P, LANCOMM, MECOMM, MobiArch, NetAI, NetEcon, NSEthics, NSDR, SIGCOMM, SOSR and 43 others. The remaining 4623 papers were published in various conference proceedings and journals of the Advanced Computing Systems Association (USENIX) computer networking related conferences between the years 2006 to 2018. The conferences we explored include the ATC, APSys, CoolDC, CSET, EVT, FOCI, HotCloud, HotEdge, HotSec, IPTPS, LISA, NetDB, NSDI, ONS, OSDI, Security, SRUTI, SustainIT, SysML, WASL, WebApps and 37 others. We could not include any of the Institute of Electrical and Electronics Engineers (IEEE) papers in the analysis because the API of the IEEE digital library made it difficult to perform extensive downloads of a large number of papers. The full list of papers examined in the surveys and the corresponding journals or conferences have been povided online [132].

We conducted a detailed word search analysis of all 7000+ papers. First, we created a list of terms/phrases that uniquely describe each generator. For example, search terms for the DPDK packet generator included 'dpdk pktgen', 'pktgen dpdk', 'dpdk packet generator', 'dpdk generator', 'dpdk based packet generator' and 'dpdk based generator'. We then created n-gram indices with n = 1 to 5, from the raw text of the entire corpus of papers. We searched these indices to locate matches of the traffic generator terms/phrases. For each match, we ran a script that captured the surrounding sentences for the location of the match, which resulted in about 1800+ papers. We manually examined the sentences for each match in order to determine whether the generator was actually used, cited, or just merely mentioned in the paper. Based on the surrounding text we were also able to identify and exclude cases where the search terms in the paper was found to refer to something other than the traffic generation context. The scripts that we wrote and used for the search and analysis of papers is open source and made publicly available online [132].

The result of the analysis is a list of traffic generators and the associated lists of papers where the generators are used, cited, and mentioned. Based on the result, we rank the generators and select the top 10 based on their usage popularity in the last 13 years for further examination. The top 10 list consists of: `iperf2` [128], `netperf` [58], `httperf` [84], `moongen` [37], `scapy` [20], `linux pktgen` [93], `netcat` [46], `TCPreplay` [130], `iperf3` [33], `DPDK pktgen` [142] in descending order of usage. Fig. 3.1 gives the details of the results of this analysis, while Table 3.1 gives the complete list of traffic generators in descending order of usage. All top 10 traffic generators are open source, and they are all software traffic generators.

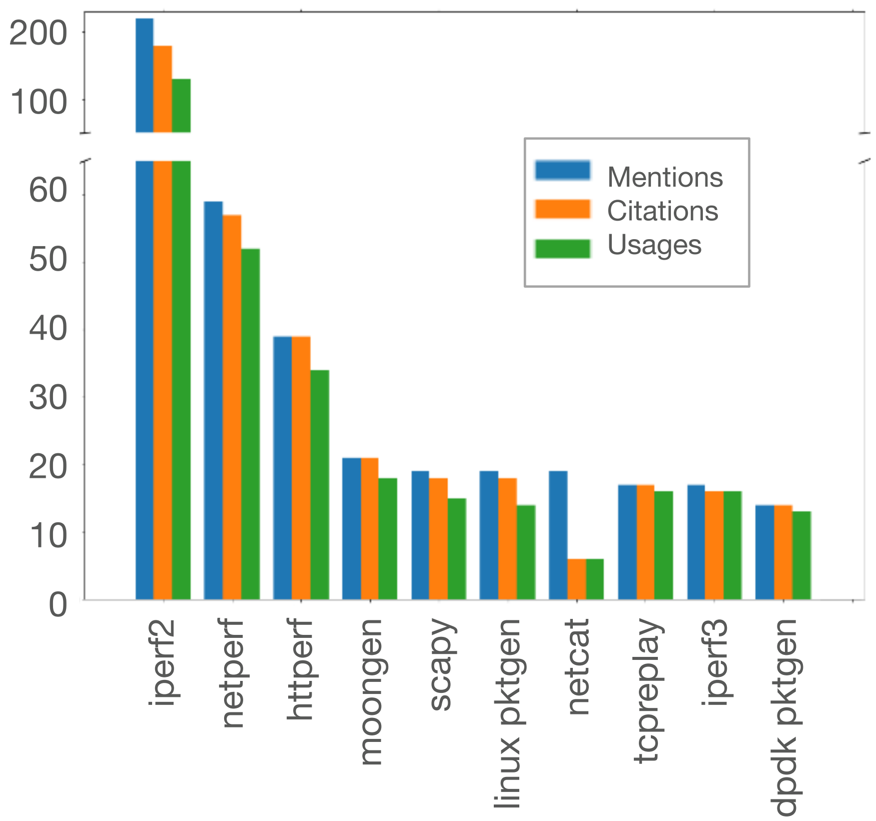


Fig. 3.1: Top traffic generators from ACM and USENIX networking-related conference publications [132]

The usage reference of each of these generators is given in Fig. 3.2 per year from 2006 to 2018. Based on the results, constant / maximum throughput traffic generators, especially `iperf2` [128], continue to dominate in terms of usage. More recent realistic traffic generators that are based on stochastic models, e.g., `swing`, `DITG`, etc., are not cited as much as the constant / maximum throughput traffic generators, even in the recent publications. Although they do not make the top 10, there are many of other realistic traffic generators in the next 10 on the list in Table 3.1.

In the later years, usage of script driven traffic generators like `moongen` [37], and `DPDK pktgen` [142] that allow extensive variations in specific header values have gained more traction, and we expect the trend to continue in the upcoming years. In addition, some of these script driven traffic generators like `trex` [25] do not feature among the top 10. However, we believe that such generators will find more utility in research in the upcoming years. When a generator is script driven, there is usually a script file that configures its running parameters with various options for programmability of the individual packet header fields per flow and some of the desired traffic characteristics. On the other hand, a typical native run of a generator such as the `iperf2` is a pre-defined flow of TCP or UDP at a user-defined throughput level.

Most constant / maximum throughput traffic generators create packets with almost no variation in header and payload contents, they usually only allow for selection of a single value for specific header fields before the beginning of their generation process. Hence they are useful for a narrow class of applications and may not reflect a true mix of network traffic in typical topologies. For example, one such generator is `iperf2`. Per each native run, `iperf2` can provide a TCP or UDP flow that is driven by a constant throughput goal. Despite these limitations, these types of generators are quite popular. In order to further verify this observation, we examined all papers where `iperf2` were used and we read through them to find out if such papers used any other traffic generation

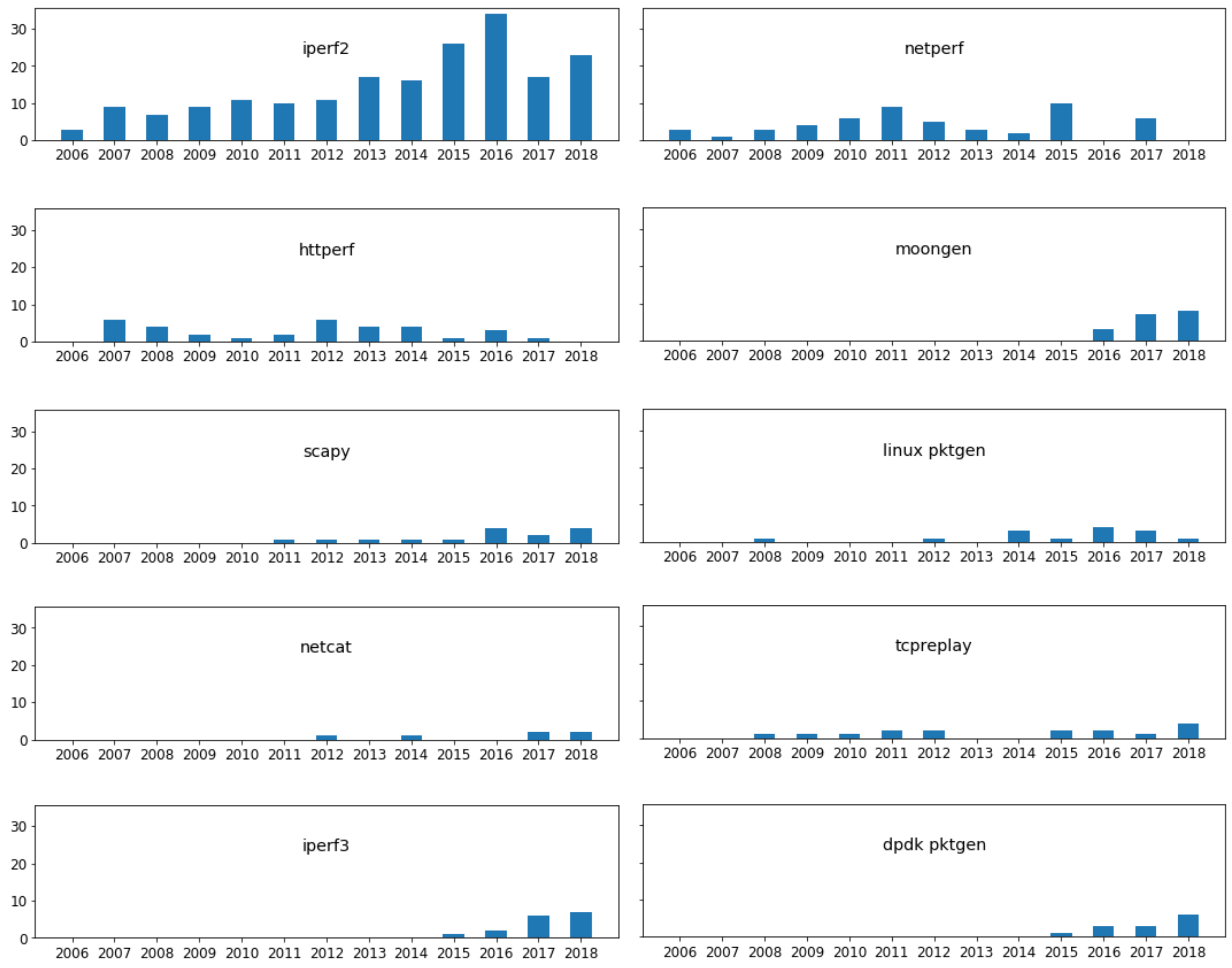


Fig. 3.2: Usage by year of top 10 generators as cited in ACM and USENIX networking-related conference publications

mechanisms in combination with `iperf2`. We counted the number of papers per year in which `iperf2` was used exclusively to plot in Fig. 3.3.

In some types of experiments that require specific traffic patterns, researchers may write appropriate wrapper scripts and native packet creators for the generation task with the desired traffic patterns. Nevertheless, there was not a significant percentage of papers referring to traffic generation without a reference to a specific generator.

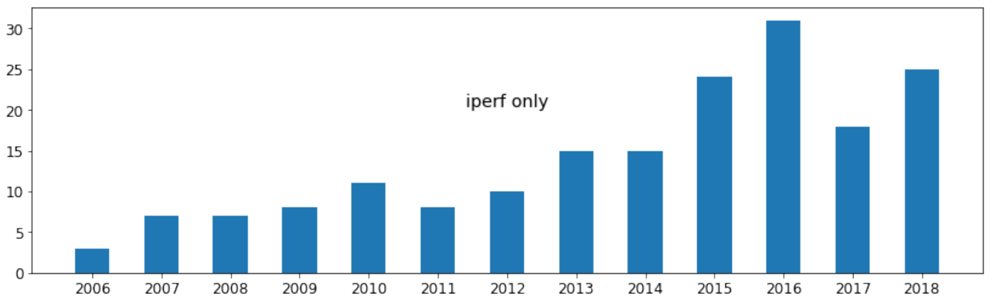


Fig. 3.3: Per year exclusive usage citations of `iperf2`.

3.3.1 Top 10 Traffic Generators

Based on the survey results described in the previous section, we give a brief description for each of the top 10 traffic generators below.

3.3.1.1 Iperf2

Iperf2 [128] is a bandwidth / throughput measurement utility. It is often used to measure the maximum achievable TCP or UDP throughput of a network, and it can also generate UDP packets at fixed throughput rates. The default output of an iperf2 test often includes the throughput in bits per seconds, total bytes sent and packet loss. Iperf supports multiple parallel connections, and can allow users to set the bandwidth, parallel streams, TCP options, number of bytes and time duration of testing.

3.3.1.2 Netperf

Netperf [58] is a network bench-marking tool developed by Hewlett Packard. It can be used to measure many different aspects of networking performance. It provides tests that measure both unidirectional throughput, and end-to-end latency. It usually operates in client-server mode, where a separate server application (netserver) is used to listen for incoming packets, while the client application (netperf) sends packets. It establishes a separate control channel between the client and the server, that is used for exchange of control information. Netperf results reports usually include throughput in bits per second, and the socket send and receive buffer sizes of the two communicating endpoints.

3.3.1.3 Httperf

The httperf [84] tool measures web server performance. With httperf, a user can generate HTTP request URIs with various http header values towards and send to any desired server address. It also allows users to specify parameters like number of times to repeat requests and number of connections to use. Its output is a report that includes metrics such as the connection rate, connection times, request rates, request sizes, reply times and reply sizes.

3.3.1.4 Moongen

Moongen [37] is a script-based high-speed traffic generator built on top of the libmoon library. Its work load generation process is typically controlled by a user-provided lua script. As a high-speed generator, and it has been tested to be able to generate up to up to 178.5 Million packets per seconds at 120 Gbit/s. It is also designed with specific features that enable it to generate packets with accurate timing injection to match expected inter packet times.

3.3.1.5 Scapy

Scapy [20] is a packet manipulation program. It is able to forge, sniff or decode packets for a wide number of protocols, and send or receive them through the network interface. It is often used as a python library in developing of other software applications that and probe, scan, analyze or attack networks. It allows a user to create any type of packet, inserting any value into the packet headers.

3.3.1.6 linux pktgen

Pktgen [93] is a module bundled with the linux kernel, that can generate packets at very high speeds. To use the traffic generator, a configuration script must first be written to prescribe packet generation parameters. It can randomize some packet headers parameters like IP and MAC addresses, provided that the random ranges are specified in the pre-start configuration script.

3.3.1.7 Netcat

Netcat [53] is a UNIX networking utility which reads and writes data into and out of network connections, using the network protocol stack on UNIX based operating systems. It is also often used as a basic network debugging tool. It allows users to write arbitrary data files into both the client and listener sides of TCP and UDP connections.

3.3.1.8 Iperf3

Iperf3 [33] is a redesign of the original iperf2. It was designed with a goal to have a simpler code base, and to have a API library that can be used from within other applications. Iperf3 essentially supports everything the original iperf2 does. In addition, it also supports a zerocopy mode, and allows an optional json output. However it is not backwards compatible with the older version and both exist as independent projects.

3.3.1.9 TCPreplay

The TCPreplay tool [130] is used for replaying network traffic from previously saved trace files. The tool was initially created to replay malicious traffic patterns for intrusion detection/prevention systems testing. It is able to resend all packets seen in an input trace file either at the same time intervals at which the packets were captured, or at a specified data rate, depending on the bandwidth of the network interface.

3.3.1.10 DPDK pktgen

DPDK Pktgen [142] is a traffic generator based on the DPDK fast packet processing framework. It is able to saturate a 10Gbps line with 64byte packets. It also has a run-time environment that allows users to supply configuration parameters and control commands for individual traffic flows. The configuration can also be written in Lua, and the configuration scripts can be used to set up repeatable experiments. Sequential packets with different header values can be generated by iterating over a set of packet headers values.

3.4 Traffic Generator Selection: Common Requirements and Features

Traffic generators have diverse sets of features and various traffic generators typically report different set of metrics. There is no single traffic generator that is better in all experiment use cases than every other one in terms of serving a research objective through these features and reported metrics. For instance, while a particular traffic generator may be good at injecting packets into a network at very high speeds, it may not provide dynamic packet length variations.

In Table 3.3 we show whether there is support for a particular feature among the top 10 traffic generators in our survey. We examined the experiments, evaluations and methodology sections of the surveyed papers for the research goals, the types of traffic workloads and their corresponding required features. We then examined the documentation, source code, man pages, help information and the associated research papers for each of the generators in the top 10 list to list the presence of a particular capability in the Table 3.3.

Table 3.4 further gives a list of the header fields in the Ethernet, IPv4, TCP and UDP protocol stacks, and provides information on how each of the traffic generators in the top 10 list supports the configuration of that header field. In some cases header fields can be set to a single value, while in other cases header fields can be set to values that vary or that are randomized during the packet generation process.

Table 3.5 presents a list of common metrics among the auto-generated reports of traffic generators. No single traffic generator reports all metrics. In fact, there are some that do not give a report at all. It is important to note that even when a generator is able to report metrics, the use of virtual interfaces in packet generation may sometimes confound the accuracy of some reported metrics. Therefore it is sometimes useful to calculate the metrics from the packet traces that are captured on the wire.

3.4.1 Methodology for Traffic Generator Selection

The Table 3.3 illustrates how there is no single generator in the top 10 list that supports every feature. There are some generators that support all of the features in addition to giving a comprehensive set of metric reports, but they usually require commercial licenses that are quite expensive [116, 65]. There is an overwhelming number of options for generator selection as shown

Table 3.3: Traffic generator selection: common features and experiment requirements

		Generator									
	Feature [1]	iperf2	net-perf	http-erf	moon-gen[2]	scapy	Linux pkt-gen[2]	netcat	iperf3	TCP-replay	DPDK pkt-gen[2]
1	set # of packets					✓	✓		✓	✓	✓
2	set total bytes	✓				✓			✓		
3	set fixed throughput	✓[3]			✓	✓	✓		✓	✓	✓
4	set randomized throughput				✓	✓					
5	set packet rate				✓	✓	✓			✓	✓
6	set time duration	✓	✓		✓	✓	✓		✓	✓	
7	send data files		✓			✓		✓	✓		
8	replay traffic traces				✓	✓				✓	
9	set fixed packet size				✓	✓		✓			✓
10	set randomized packet sizes				✓	✓	✓				✓
11	set fixed inter-packet time				✓	✓					
12	set randomized inter-packet times				✓	✓					
13	support TCP connections	✓	✓	✓		✓		✓	✓		
14	support SCTP connections		✓			✓			✓		
15	set MSS	✓				✓			✓		
16	set reporting intervals	✓	✓		✓	✓			✓	✓	✓
17	set interface				✓	✓	✓			✓	✓
18	specify IP addr. of interface	✓	✓					✓	✓		
19	set CPU affinity		✓		✓		✓		✓		✓
20	generate IP fragments				✓	✓	✓				✓
21	bi-directional generation	✓	✓	✓		✓					
22	multiple parallel connections/flows	✓	✓	✓	✓	✓	✓		✓		✓
23	arbitrary http requests			✓		✓		✓			

Table 3.3 Footnotes

1 Feature descriptions are provided in appendix A.1
2 Requires exclusive control of the network interface
3 UDP only

Table 3.4: Supported configuration of header fields for the top 10 traffic generators

	Header Field [1]	Generator									
		iperf2	net-perf	http-erf	moon-gen	scapy	Linux pkt-gen	netcat	iperf3	TCP-replay	DPDK pktgen
1	L2 source MAC				⋆	⋆	⋆			⋆	⋆
2	L2 destination MAC				⋆	⋆	⋆			⋆	⋆
3	L2 VLAN ID				⋆	⋆	✓			⋆	⋆
4	L2 ethertype				⋆	⋆				⋆	⋆
5	L3 source IP	✓	✓		⋆	⋆	⋆	✓	✓	⋆	⋆
6	L3 destination IP	✓	✓	✓	⋆	⋆	⋆	✓	✓	⋆	⋆
7	L3 header length				⋆	⋆				⋆	⋆
8	L3 DSCP/TOS	✓			⋆	⋆	✓	✓	✓	⋆	⋆
9	L3 ECN				⋆	⋆				⋆	⋆
10	L3 total length				⋆	⋆				⋆	⋆
11	L3 identification				⋆	⋆				⋆	⋆
12	L3 don't fragment				⋆	⋆				⋆	⋆
13	L3 more fragments				⋆	⋆				⋆	⋆
14	L3 fragment offset				⋆	⋆				⋆	⋆
15	L3 TTL	✓			⋆	⋆				⋆	⋆
16	L3 protocol				⋆	⋆				⋆	⋆
17	L3 header checksum				⋆	⋆				⋆	⋆
18	L4 source port		✓		⋆	⋆	⋆[2]	✓	✓	⋆	⋆
19	L4 destination port	✓	✓	✓	⋆	⋆	⋆[2]	✓	✓	⋆	⋆
20	TCP sequence num				⋆	⋆				⋆	⋆
21	TCP ack number				⋆	⋆				⋆	⋆
22	TCP data offset				⋆	⋆				⋆	⋆
23	TCP reserved bits				⋆	⋆				⋆	⋆
24	TCP flags				⋆	⋆				⋆	⋆
25	TCP window size	✓	✓		⋆	⋆			✓	⋆	⋆
26	TCP checksum				⋆	⋆				⋆	⋆
27	TCP urgent pointer				⋆	⋆				⋆	⋆
28	TCP options	✓[3]	✓[3]		⋆	⋆			✓[3]	⋆	⋆
29	UDP length				⋆	⋆				⋆	⋆
30	UDP checksum				⋆	⋆				⋆	⋆

Table 3.4 Legend

✓: set to single value (no variation of the header field is supported during generation)

⋆ : single, varying, or randomized values can be set for the header field

Table 3.4 Footnotes

1 L2, L3, and L4 represents Layer 2, Layer 3 and Layer 4 of the TCP/IP network stack

2 UDP only

3 TCP_NODELAY option only

Table 3.5: Reported metrics for the top 10 generators

		Generator									
	Reported Metric [1]	iperf2	net-perf	http-erf	moon-gen	scapy	Linux pkt-gen	netcat	iperf3	TCP-replay	DPDK pktgen
1	throughput	✓	✓	✓	✓		✓		✓	✓	✓
2	latency		✓		✓		✓				✓
3	packet rate				✓		✓			✓	✓
4	total no. of packets	✓			✓		✓		✓	✓	✓
5	total no. of bytes	✓	✓		✓				✓	✓	✓
6	duration	✓	✓	✓	✓		✓		✓	✓	✓
7	jitter	✓			✓				✓		✓
8	no. of retransmissions		✓		✓					✓	
9	no. of drops	✓			✓		✓		✓	✓	✓
10	MSS	✓	✓		✓						
11	congestion win. size(s)	✓							✓		
12	CPU demand		✓	✓							
13	number of flows or connections	✓	✓	✓	✓		✓		✓	✓	✓
14	request/response transaction rates		✓	✓[2]							

Table 3.5 Footnotes

1 Metric descriptions are provided in appendix A.2

2 http only

on the list of generators in our Table 3.1. For every experiment, researchers are tasked with a preliminary assessment of generator features and an evaluation of each generator for the research objectives. We propose a methodology that uses this paper's compilations on generator category, available features and built-in reporting of the metrics. The goal is to serve the research tasks with the capabilities of an applicable generator while reporting a consistent set of metrics leading into repeatable results that are comparable with other similar studies.

1. **Definition of Workload Requirements:** Before selecting a generator, an experimenter first needs to identify specific requirements of the traffic workload that are of vital importance for the experimental goals of the research. Specifically a researcher must identify:

   - the specific features of the desired traffic workload picked from tables 3.3 and 3.4
   - the metrics that need to be reported picked from the Table 3.5

   This first step is in a sense the most important one in the methodology. The chosen requirements will serve as the driving input for each of the subsequent steps below.

2. **Availability:** A researcher may start with the list of 96 traffic generators given in the Table 3.1 to determine which generators are available. We define the availability of a generator by its active maintenance status and note also that the generator has to be within the skill-set, resources and capabilities of the researcher from the perspectives of the platform requirements and ease-of-use.

3. **Replace with Validation/Workload:** The initial list is then filtered based on the category of traffic workloads. A taxonomy for the workload characteristics has been provided in section 3.2.

4. **Features:** One of the key pivoting points for the researcher is what features are supported by a generator of choice (tables 3.3 and 3.4). The desired workload properties for the specific research goals could result in a trade-off when generators that lack some of the key features are preferred per availability concerns. This step highlights what has been desired, what decision has been made, and what trade-off has taken place so that the outcomes of the research data is assessed accordingly.

5. **Reported Metrics:** The reported metrics in Table 3.5 lists the metrics automatically made available by each generator. If a desired metric is not available in a specific generator, other generators that provide the metric could be considered. In essence, this step raises awareness about the measurement outcome objectives for the research tasks.

We expect that these steps are overlapping with the current practice. As researchers make choices on available generators at their disposal some preference has been established for the most popular ones such as the observed preference for `iperf2` in our survey. We provide these steps as a guide to raise awareness on the process of the selection of workload generation techniques in the community.

3.4.2 An Example: Load-balancing Research

To demonstrate the methodology, we walk through the steps in the previous section for a research project on the evaluation of the performance of a new fictitious transport layer load-balancing network function that leverages TCP header information in forwarding transport layer packets.

Step 1 - Workload Requirements:

1. Features:

   (a) multiple parallel TCP connections or flows with diverse transaction characteristics and relatively fixed total throughput

   (b) ability to generate packets with varying packet sizes

   (c) ability to vary header fields: layer 3 source IP addresses and layer 4 source port numbers

2. Metrics: throughput per flow, packet rate per flow and jitter

Step 2 - Availability: From Table 3.1, generators are picked based on accessibility and active maintenance status. Based on section 3.2, this leaves us with 31 generators among which we have all those in the top 10 list - `iperf2`, `netperf`, `httperf`, `moongen`, `scapy`, `linux pktgen`, `netcat`, `TCPreplay`, `iperf3`, `DPDK pktgen` and 21 others from Table 3.1.

Step 3 - Workload: Many classes of traffic generators have support for multiple TCP connections with variations in source IP addresses and port numbers. For example, some constant/-maximum throughput traffic generators (section 2.1.1) allow for multiple simultaneous TCP and UDP connections. Many model-based generators (section 2.1.5 and 2.1.6) and script driven traffic generators (section 2.1.4) also support the type of traffic desired.

Step 4 - Features: Tables 3.3 and 3.4 are utilized to narrow down the generators of interest for the research task. Based on the feature list in step-1, shortlisted generators must have check marks on rows 3, 10, and 22 of Table 3.3 and a star in rows 5 and 18 of Table 3.4. Thus, the resulting list includes `scapy`, `moongen` and `dpdk pktgen`. We do not include `linux pktgen` in the shortlist because randomized layer 4 source port numbers is not possible as shown in footnote 1 of Table 3.4, even though all the other requirements are met by this generator.

Step 5 - Metrics: The metrics requirements in step-1 is searched in the Table 3.5. By taking a look at rows 1, 3, and 7 of that table, we see that we are left with `moongen` and `dpdk pktgen`. A choice could be made between any of the two for the targeted research goal, since they both meet workload objectives.

In summary, our survey has found that most research papers have used constant / maximum throughput traffic generators over the last 13 years and that there are very few instances where trace driven model-based realistic traffic generators are used. In the next chapter we go on to explore the details of our new frame work for network traffic modeling and generation.

Chapter 4

Methodology: Framework for Realistic Traffic Modeling and Generation

This chapter provides the details of our new framework for network traffic modeling and realistic application traffic generation. We share specific algorithms and methods used in each component of the framework, showing how it enables creating and evaluating a diverse set of traffic modeling algorithms for any input application network traffic. We also describe one of the new modeling algorithms (`uhgeneric-v4` modeling algorithm) that we have developed using the framework and its corresponding traffic generation method. In addition to the framework details, we introduce a comprehensive evaluation system for realistic traffic generators, with the specific metrics designed to assess the performance of modeling and traffic generation methods created using the framework. An illustration of the framework is given in Fig. 4.1.

The framework consists of 4 main components:

- A dataset extractor
- A modeling system
- A traffic generator
- An evaluation system

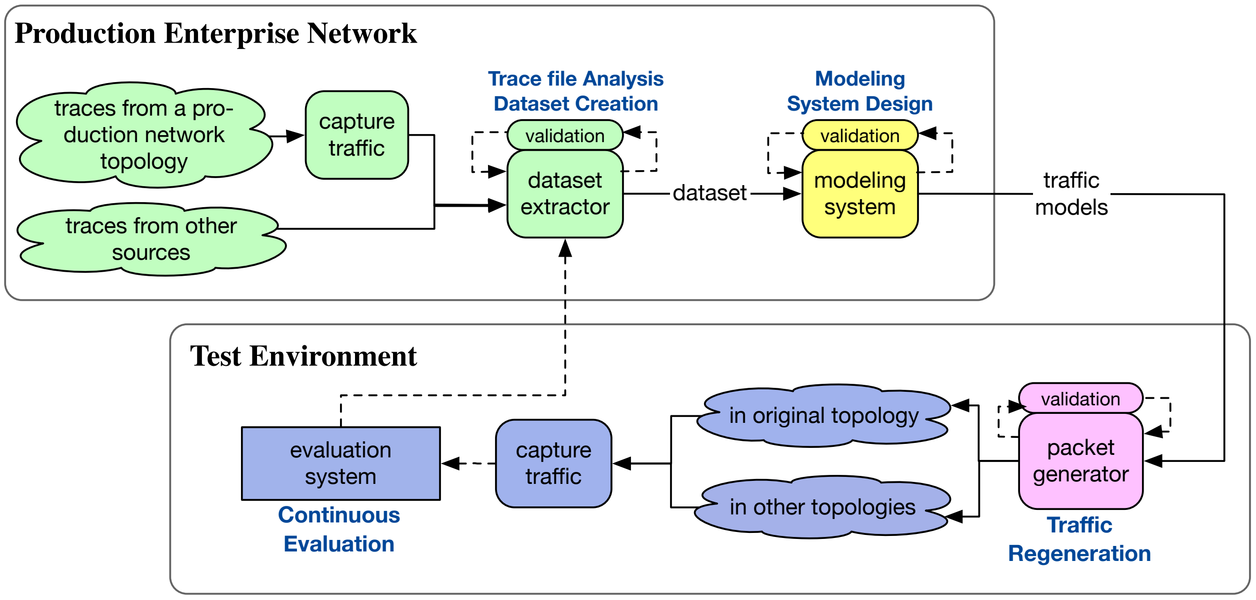


Fig. 4.1: Framework for realistic traffic modeling and generation showing the four components: a dataset extraction system, a modeling system, a packet generation system, and an evaluation system.

The dataset extractor analyzes packets in an input trace file to create a dataset. The modeling system uses various algorithms to create traffic model files from the dataset. In the traffic generator, the model files are used on nodes within a test network to generate realistic traffic. Finally, the evaluation system measures the modeling and traffic generation method's performance by comparing the generated traffic with real traffic. Each component in the framework is designed to be independent, with well-defined input and output interfaces. They have all been implemented as open-source software utilities and are available online [131].

4.1 Dataset Extractor

Synthetically generated traffic is considered realistic for an application if it has metrics and statistics that closely match traffic from a real production network with devices running that application. Thus traffic that is realistic for a specific application or network may not be realistic for another application. For example, real traffic from a web browsing application will differ significantly from video streaming application traffic. Similarly, there will be significant differences between traffic from a home network and traffic from an enterprise or data-center network. Hence all work done in realistic packet generation must start with actual network traffic traces obtained from the target application or production network.

Therefore, in the framework, the process begins with analyzing real network traffic from various applications in a dataset extraction stage that performs careful packet processing and further computation to obtain a comprehensive dataset. In our evaluation experiments (chapter 5), we obtain real network traffic by packet captures on networks with many categories of application traffic, including web browsing traffic, remote desktop protocol (RDP) traffic, secure shell (SSH traffic), video streaming traffic, and a multi-tier application traffic.

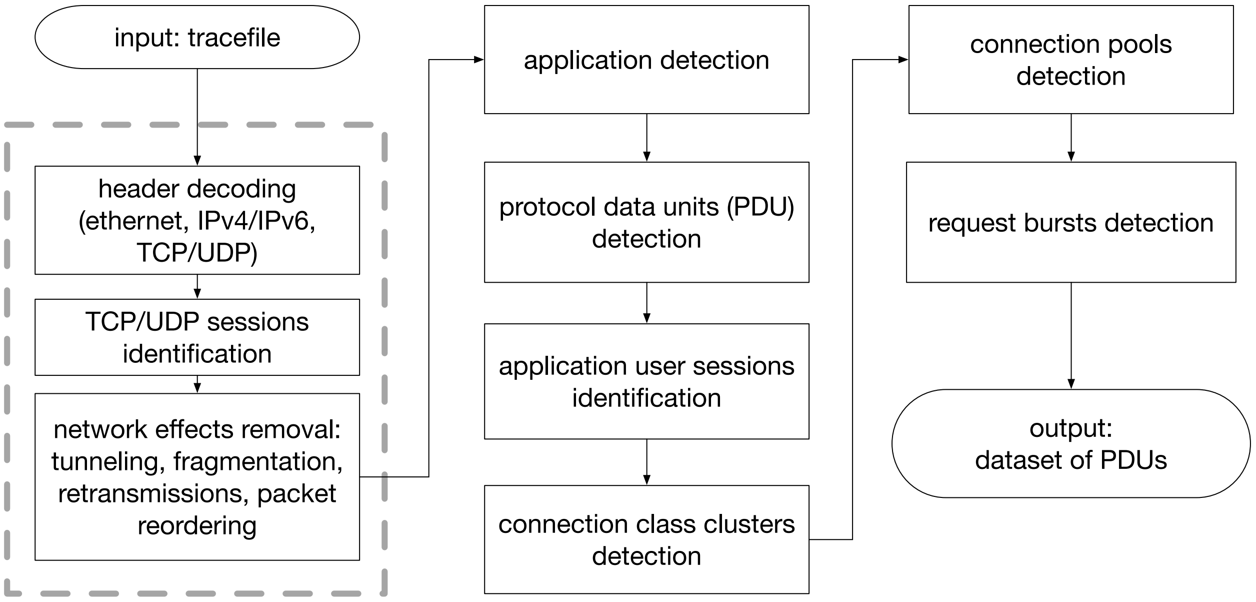

input: tracefile
header decoding (ethernet, IPv4/IPv6, TCP/UDP)
TCP/UDP sessions identification
network effects removal: tunneling, fragmentation, retransmissions, packet reordering
application detection
protocol data units (PDU) detection
application user sessions identification
connection class clusters detection
connection pools detection
request bursts detection
output: dataset of PDUs

Fig. 4.2: Dataset extraction process

The dataset extraction process first decodes packet headers in the input network traffic trace. It determines the transport layer sessions associated with each packet. It then detects how packets may have been affected by networking effects like fragmentation, retransmissions, and packet reordering, making adjustments to the dataset when required. The process continues with additional steps for application detection, protocol data unit (PDU) detection, user-session identification, connection cluster detection, connection pools detection, and request bursts detection. The output of all this data set extraction process is a set of `csv` files that describe the packets, PDUs, and connections seen in the input trace file. Fig. 4.2 illustrates each of the steps involved in the dataset extraction process [131]. We have also provided details of each step involved in the process in the subsections below.

4.1.1 Header Decoding

This first step in the dataset extraction process involves analyzing the input trace to decode basic packet headers and data across all the network stack layers. We extract and decode all

Media Access Control (MAC) headers at the data-link layer, including source and destination MAC addresses and VLAN headers. We also determine the data-link layer payload data sizes. Similarly, we extract all internet protocol version 4 (IPv4) and internet protocol version 6 (IPv6) headers at the network layer, and we extract TCP or UDP header values at the transport layer. Our implementation leverages the DPKT python library [13] to decode packet header contents while following the recommendations in the Ethernet, IPv4, IPv6, TCP and UDP RFCs [54, 104, 105, 103].

4.1.2 TCP and UDP Sessions Identification

The second step involves the determination of the transport layer sessions for each packet found in the traces. For UDP, we identify unique sessions based on a tuple consisting of the source IP address, destination IP address, source port number, and destination port number, assigning packets with the same tuple to the same UDP session. To identify TCP sessions, we use an identifier consisting of a unique number and the tuple consisting of the source IP address, destination IP address, source port number, and destination port number. We check the packets sequentially and label each new TCP SYN packet (without an ACK flag) as the beginning of a new TCP session if it does not match any previously discovered session. For each packet found with matching the tuple identifier of an existing TCP session, the tool labels the packets as belonging to the corresponding TCP session, following the TCP state machine's guidelines [105]. On detecting TCP FIN packets from each side of a session, the connection is closed, and any new packet matching the same tuple identifier is classified as a new TCP session.

4.1.3 Removal of Network Protocol Responses

It is essential to start with clean packets devoid of network-level responses to network impairments when modeling realistic traffic for an application. Hence, in this step, we make adjustments for network effects like packet reordering, fragmentation, and packet losses to produce clean packets data suitable for further analysis and modeling. We detect and remove tunnel encapsulations at the protocol stack's data-link and network layers before proceeding to decode the next upper-layer protocol. We perform fragmentation analysis at the network layer to label packets belonging to the same original packets fragment for both IPv4 and IPv6. We also analyze each TCP connection based on TCP sequence and acknowledgment numbers to identify and label all duplicated packets and reorder all out-of-order packets. The outcome of the dataset extraction process is a list that contains all Ethernet, IPv4, TCP, and UDP header values extracted for each packet, along with the first 10 bytes of payload content.

4.1.4 Application Detection

Any network traffic trace typically contains traffic from many applications. Different applications send and receive data according to differing patterns. Hence, network traffic that is realistic for a given application may not be realistic for another kind of application. Therefore separate models must be created for different applications. Consequently, it is necessary to label individual packets by their respective applications before application-level traffic modeling occurs. Network packets applications detection is an entire field of research on its own. Many methods have been proposed in research for application detection in networks. These methods are in 3 main categories which we discuss in the subsections below:

1. **Port Number Based Methods:** Port number based methods are the earliest and easiest

methods for application classification. They typically determine the application of packets in a transport layer session based on the server port numbers. If the server port number is a on the list of Internet Assigned Numbers Authority (IANA) reserved port numbers [55], the packet can be easily identified as belonging to an application corresponding to the port number on the IANA list [82]. For example, packets with port number 53 can be detected to be DNS packets. This method is fast and easy to implement. However, this method may give incorrect classifications when the port numbers in a packet header are not on the IANA list, or when applications use arbitrary port numbers. Hence additional heuristics are often used in such cases to make quick guesses from packet headers contents.

2. **Deep Packet Inspection (DPI) Based Methods:** Deep packet inspection (DPI) based methods for application identification were created to avoid the limitations of the port number based approaches. DPI based methods operate by matching various packet header values and payload contents, with predetermined signatures for known applications [19]. These methods are often fast enough to operate at line rate, and typically give better accuracy for applications with known signatures. DPI based methods are usually unable to classify packets for new applications for which signatures have not been identified previously [82]. The most commonly used open-source application detection methods in this category are `ntop nDPi` [30] and `libprotoident` [6].

3. **Machine Learning Based Methods:** More recently, various machine learning algorithms have been proposed for traffic application identification. Some of the algorithms proposed include support vector machines (SVM) [148], naive Bayes [40], k-means clustering [76] and many other supervised and unsupervised methods [28]. Many of these machine learning-based

methods can identify new applications without a requirement for prior signatures. In many cases, after creating a classification model, they can classify packets at line rates. However, their accuracy levels are often not as high as DPI based methods for applications with known signatures [82].

In our dataset extraction process, we use the port number based application classification. However, our dataset extractor is designed to be modular and has hooks for the future addition of DPI and machine learning-based methods.

4.1.5 Protocol Data Units (PDUs Detection)

When an endpoint of a network application is sending data to its counterpart, the data sent may be a file or a string of bytes. In most cases, applications send data by pushing the bytes through a network socket controlled by the operating system. However, the Ethernet protocol's maximum transmit unit (MTU), the TCP Maximum segment size (MSS), and other factors may limit how many bytes can be transferred in 1 packet through an interface per time. Hence when an application is to send a large chunk of data through a network, the operating system network stack on most devices breaks up the large chunk of data into smaller sizes so that the packets can fit into the allowed MTU and MSS of the interfaces used. The entire chunk of data or file that the application pushes at a time is what we refer to as the application-level protocol data unit (PDU). A single PDU may be broken down into many packets while being sent out of an interface. Hence, when modeling an application's network behavior, it is necessary to identify and group packets that could have been derived from the same PDU so that models of data exchange behavior at the application level can be created.

In the dataset extraction process of our framework, we use a heuristic to identify PDUs. The heuristic is based on our observations of packets in TCP or UDP sessions. We found that a PDU typically consists of an uninterrupted continuous sequence of payload bearing packets from the same endpoint. The sequence of packets in the same PDU ends when a payload bearing packet from the other endpoint is observed in the stream, or when a packet with a payload size less than the maximum possible payload is seen. A PDU also ends when a packet from the same endpoint with a relatively large inter-packet time is observed. The algorithm for this PDU detection method is given in Algorithm 1 below. Our actual implementation uses a vectorized version of the algorithm to speed up the detection process. Based on our validation experiments with packets from a web browsing application, the algorithm was able to correctly identify PDUs over 99% of the time.

Algorithm 1 Heuristic for identifying application level protocol data units (PDU)

```
 1: procedure IdentifyPDUs($l4\_session\_pkts, ipt\_threshold, max\_data\_size$)
 2:     $curr\_pdu\_id \leftarrow -1$
 3:     $prev\_pkt \leftarrow NULL$
 4:     for $pkt$ in $l4\_session\_pkts$ do
 5:         if $pkt.payload\_size \neq 0$ then
 6:             if $pkt.src\_ip \neq prev\_pkt.src\_ip$ OR $pkt.src\_port \neq prev\_pkt.src\_port$ then
 7:                 $curr\_pdu\_id$ += 1
 8:             else if $pkt.time - prev\_pkt\_time > ipt\_threshold$ then
 9:                 $curr\_pdu\_id$ += 1
10:             else if $prev\_pkt\_size < max\_data\_size$ then
11:                 $curr\_pdu\_id$ += 1
12:             end if
13:             $pkt.pdu\_id \leftarrow curr\_pdu\_id$
14:             $prev\_pkt \leftarrow pkt$
15:         end if
16:     end for
17: end procedure
```

4.1.6 Application User Session Identification

In any trace file, the traffic may come from multiple 'user-sessions' of constituent applications, and the user sessions may occur either simultaneously or sequentially. Multiple user sessions of an application may exist between the same two endpoints. For example, in a remote desktop application, a user session corresponds to one login session between a user's client computer and a server, and it begins when a user first attempts to log in to the remote computer and ends when a user logs out or closes the remote desktop client application. Similarly, for Hypertext Transfer Protocol (HTTP) website browsing, a single user session may correspond to a user's interactions with a website in the interval between the time when a user first accesses a specific website and the time when that user stops interacting with the website. In some cases, these application user sessions may not involve a physical human being; the 'user' may be a software agent in such cases.

In our dataset extractor, we use a heuristic to identify user sessions in traffic for any pair of communicating endpoints. The heuristic leverages our observation that user-sessions boundaries are associated with periods of significant idle times where there are no active connections between two communicating endpoints for an application. Typically, when a user quits an application, TCP connections that have been opened by that application will be closed. New connections will be started only when a new user session starts, after a time interval. This time interval between user sessions is thus user-interaction based and must be at least in the order of a few seconds. Thus, the heuristic extracts user sessions by grouping connections from the same client machine to the same server that overlap in time or are separated in time by less than a threshold. It is possible to obtain suitable value for the user session interval threshold by running a 1-dimensional clustering of inter-session-gaps. An excerpt from the algorithm used is given in Algorithm 2 below. Based on our validation experiments with packets from a web browsing application, the algorithm correctly identified user-sessions over 99% of the time.

Algorithm 2 Heuristic for identifying application user sessions

```
1: procedure DetectUserSessions($endpt\_pair\_pkts, default\_gap\_threshold$)
2:     $gaps, l4\_sess\_pkts \leftarrow$ GetPreL4SessionGaps($endpt\_pair\_pkts$)
3:     $threshold \leftarrow default\_gap\_threshold$
4:     $u\_idx \leftarrow 0$
5:     for $i = 0$ to Length($gaps$) $-1$ do          ▷ for each connection & pre_connection_gap
6:         if $gaps[i] > threshold$ then
7:             $u\_idx$ += 1
8:         end if
9:         for $packet$ in $l4\_sess\_pkts[i]$ do
10:            $packet.u\_idx \leftarrow u\_idx$
11:        end for
12:    end for
13: end procedure
```

4.1.7 Connection Class Clusters Detection

An application may have multiple transport layer sessions (TCP or UDP conversations) of distinct types with dissimilar statistics. It is essential to detect when such multiple connection classes exist and factor them in when creating high fidelity realistic traffic models for the traffic for such applications.

For example, in a popular remote desktop application (RDP) client application, a single user-session usually consists of at least two or more TCP connections in a sequence. When there are n TCP connections in a user session, the first $(n-1)$ connections opened at the beginning of the user session are typically brief (only a few seconds long), in comparison with the nth connection, which is usually much longer and often lasts for the duration of the RDP session. A closer examination of the RDP network traffic revealed that the first connection in any RDP session is used for user authentication, while the last connection is used for data exchange. Multiple short connections

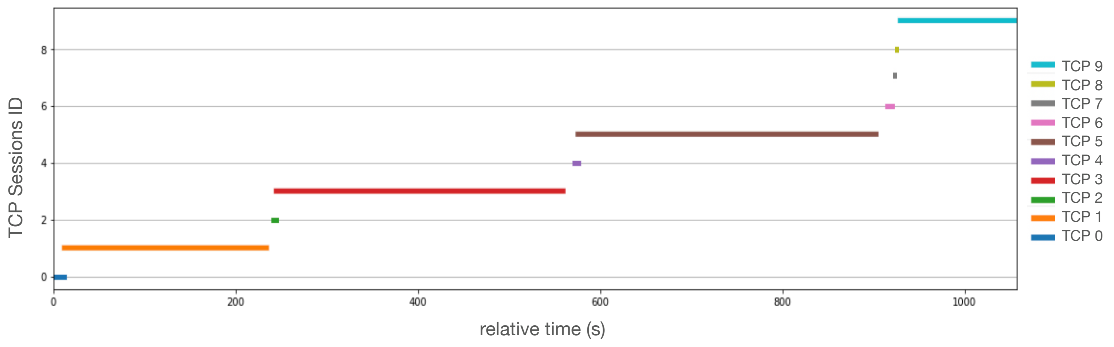


Fig. 4.3: Connection classes in RDP: 4 sequential RDP user-sessions showing 2 distinct connection classes, with connections TCP 0, 2, 4, 6, 7, 8, in connection class 1, and connections TCP 1, 3, 5, 9 in connection class 2

at the beginning of each user session were found to correspond to multiple failed login attempts. Once authentication is successful, the RDP application opens a more extended connection for data exchange, which lasts for the rest of the user session duration. The authentication connections usually have very different statistics in comparison to the data connection. A plot of the connection duration for multiple sequential RDP sessions from our captured RDP investigation is shown in Fig. 4.3.

In our dataset extractor, we use clustering algorithms to group an application's connections into clusters based on a set of features extracted from each connection. The features extracted include: the duration of each connection, the connection start time relative to the start of its user-session, the number of PDUs sent from the client, the number of PDUs sent from the server, the total data in bytes from the client, and the total data in bytes from the server. After extracting this dataset for each connection, the tool pre-processes the input dataset by standardization, it then performs dimensionality reduction using principal component analysis (PCA). After pre-processing, we use the DBSCAN algorithm to cluster the dataset, yielding possible connection classes. We use DBSCAN clustering because it does not require an a-priori specification of the number of clusters, it can find arbitrarily shaped clusters, and it is insensitive to the ordering of entries in the input dataset. Our dataset extractor component is designed to be modular and allows users to add any desired clustering algorithm, with corresponding input arguments for detecting connection classes. An excerpt from the connection class detection algorithm used is given in Algorithm 3 below. Our validation experiments with packets from RDP traffic showed that the algorithm correctly identified connections over 98% of the time.

Algorithm 3 Algorithm for connection classes detection

```
1:  procedure DetectConnClasses($app\_conns$)
2:      $dataset \leftarrow$ NewList
3:      for $conn$ in $app\_conns$ do
4:          $row \leftarrow$ NewDictionary
5:          $user\_session \leftarrow$ GetUserSession($conn$)
6:          $row[conn\_duration] \leftarrow conn.last\_pkt.time - conn.first\_pkt.time$
7:          $row[conn\_start\_time] \leftarrow conn.first\_pkt.time - user\_session.first\_pkt.time$
8:          $row[num\_pdus\_cli] \leftarrow$ Length($conn.client\_pdus$)
9:          $row[num\_pdus\_srv] \leftarrow$ Length($conn.server\_pdus$)
10:         $row[total\_data\_cli] \leftarrow$ GetTotalBytesFromClient($conn$)
11:         $row[total\_data\_srv] \leftarrow$ GetTotalBytesFromServer($conn$)
12:         Add($dataset, row$)
13:     end for
14:     $x \leftarrow$ MakeArray(dataset)
15:     $x\_scaled \leftarrow$ StandardScaler().FitTransform($x$)
16:     $x\_pca \leftarrow$ PCA().FitTransform($x\_scaled$)
17:     $conn\_classes \leftarrow$ Dbscan().fit($x\_pca$).labels
18:     for $i = 0$ to Length($app\_conns$) $-1$ do                ▷ for each connection
19:         for $pkt$ in $app\_conns[i]$ do
20:             $pkt.conn\_class \leftarrow conn\_classes[i]$
21:         end for
22:     end for
23: end procedure
```

4.1.8 Connection Pools

Some applications send and receive data by creating a pool of connections between two communicating endpoints. These applications then use connections from the pool to service any requests or responses to be sent. This behavior is common in database client applications and web browser applications. For example, in a popular web browser application that we examined, the client endpoints opened a set of connections to the server and repeatedly used each of these connections to service all outgoing requests, and connections are recycled at intervals during the user session, as shown in Fig. 4.4. It is important to capture this behavior when modeling network traffic to enable the generation of realistic traffic for applications that use connection pools.

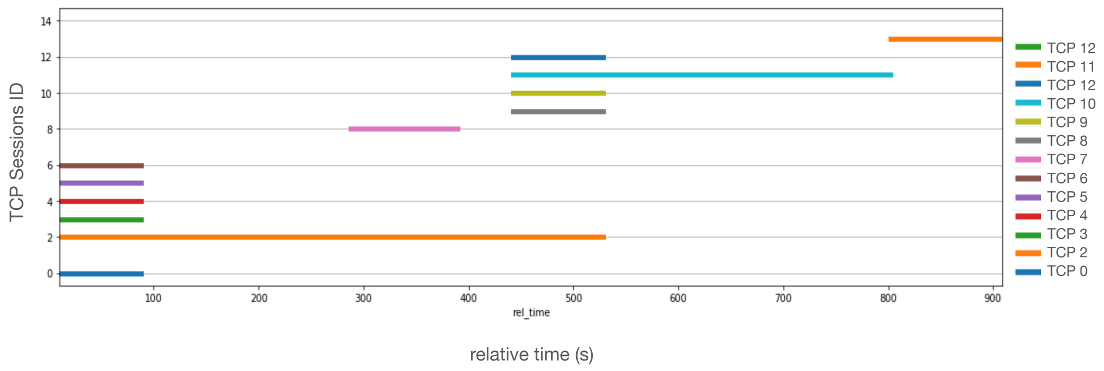


Fig. 4.4: Connection classes in web browser: A web browser user-session showing connection pools. Connections TCP 0, 3, 4, 5, 6 in a pool, and connections TCP 8, 9, 11 in a second pool

Our dataset extraction process classifies connections as belonging to the same pool if they belong to the same user session and connection class, have the same server endpoint, the same source IP, and if they start and end at approximately the same time, or within a few milliseconds of each other. We use a simplified union-finding algorithm based on the criteria above to get the connections that are in the same pool. An excerpt of the algorithm is given in Algorithm 4 below. Validation experiments with packets from web browsing traffic showed that the algorithm correctly identified connections over 97% of the time.

Algorithm 4 Algorithm for connection pools detection

```
procedure DetectConnPools($app\_conns, c\_threshold$)
    for $i = 0$ to Length($app\_conns$) $-1$ do
        $app\_conns[i].pool \leftarrow i$
    end for
    for $i = 0$ to Length($app\_conns$) $-1$ do
        for $j = i + 1$ to Length($app\_conns$) $-1$ do
            if
                $app\_conns[i].conn\_class == app\_conns[j].conn\_class$ AND
                $app\_conns[i].server\_endpoint == app\_conns[j].server\_endpoint$ AND
                $app\_conns[i].client\_ip\_addr == app\_conns[j].client\_ip\_addr$ AND
                $app\_conns[i].start\_time - app\_conns[j].start\_time <= c\_threshold$ AND
                $app\_conns[i].end\_time - app\_conns[j].end\_time <= c\_threshold$ then
                if
                    $app\_conns[i].pool == app\_conns[j].pool$ then
                    for $k = 0$ to Length($app\_conns$) $-1$ do
                        if $app\_conns[k].pool == app\_conns[j].pool$ then
                            $app\_conns[k].pool \leftarrow app\_conns[i].pool$
                        end if
                    end for
                end if
            end if
        end for
    end for
end procedure
```

4.1.9 Request Bursts Detection

Most applications generate network traffic in a bursty way. That is, they send data in a repeating sequence of sudden bursts of PDUs with idle intervals where no data is sent. For example, in web browsing traffic, bursts of requests and responses usually correspond to times when the user agent performed an action, usually by clicking on a link. A chart showing requests and responses, corresponding to user clicks from web browsing traffic is given in Fig. 4.5.

In our dataset extraction process, client PDUs in the same connection pool that are sent at approximately the same time, without any replies in between, are considered to be in the same request burst. The algorithm used for identifying request bursts is given in Algorithm 5. Our validations tests of the algorithm with web browsing traffic showed 99% accuracy in identifying request bursts.

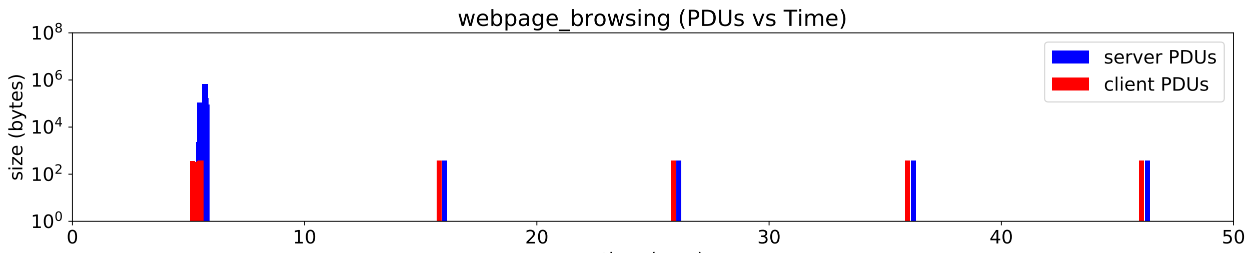


Fig. 4.5: Request bursts in web browsing traffic. Request burst and idle times correspond to intervals between user clicks

Algorithm 5 Algorithm for request bursts detection

```
 1: procedure DetectRequestBursts($connpools, rb\_threshold$)
 2:     $currburst\_id \leftarrow -1$
 3:     for $cpool$ in $connpools$ do
 4:         $prev\_time \leftarrow -inf$
 5:         for $cpool$ in $connpool.pduslist$ do
 6:             if $pdu.time - prev\_pdu\_time > rb\_threshold$ then
 7:                 $currburst\_id$ += 1
 8:             end if
 9:             $pdu.request\_burst \leftarrow currburst\_id$
10:         end for
11:     end for
12: end procedure
```

At the end of all of the dataset extraction steps described in the subsections 4.1.1 to 4.1.9 above, the result is a set of `csv` dataset files, one per each of the packets, PDUs, and transport layer (TCP and UDP) sessions found in the input trace file. These files are used as input into the traffic modeling system component of the framework.

4.2 Modeling System

In any realistic traffic generation framework, the generated traffic quality can only be as good as the model. A good model must make it possible to generate traffic that is similar to the real application traffic and must not store any payload data. It must be flexible enough to model any application effectively, and it should also allow traffic generation on networks with diverse topological variations.

Our new framework's modeling system component accepts the dataset created in the dataset extractor component to produce an output traffic model file. The modeling and generation framework is designed to be easily extensible, to allow other users to add implementations of their custom modeling methods or algorithms, and their corresponding packet generation methods. Using the framework, we created many traffic modeling methods. One of the high fidelity algorithms we created is the `uhgeneric-v4` modeling algorithm. We provide the details of this modeling algorithm in the subsection below.

4.2.1 The `uhgeneric-v4` Modeling Algorithm

We implemented a traffic modeling method using our traffic modeling system. This modeling method, the `uhgeneric-v4` modeling algorithm, performs well at regenerating realistic traffic for any generic application. The development of the modeling algorithm is based on our observations on how several applications send and receive data over connected networks. The modeling algorithm combines an expert packet processing system with machine learning clustering algorithms and stochastic models to create a model that effectively generates realistic network traffic for any application.

Real network traffic often consists of traffic from many applications. Real applications send out application-level protocol data units (PDUs) according to patterns dictated by the application's internal processes. They usually create network traffic with multiple connections that exhibit unique data exchange behavior. Hence, a good generic modeling system must replicate traffic characteristics for any application found in an input dataset. The traffic metrics statistics and distribution for the synthetically generated traffic based on the models must be as similar as possible to the real application network traffic at all levels.

Therefore, before designing the `uhgeneric-v4` modeling method, we examined network traffic from many applications to understand how they send and receive data, that is, to determine their Application Data Exchange Patterns. We explored traffic for Web browsing data, video and audio streaming traffic, log push traffic, Secure Shell (SSH) login traffic, remote desktop traffic, and many others. For each of the applications above, we extracted client and server PDUs and plotted the request and response sizes a function of time on a semi-log scale, showing how each of the applications exchanges data. Some of the unique application data exchange patterns are plotted in Fig. 4.6.

The plots clearly illustrate how data exchange patterns can be different across different applications. As an example, we noted in our web page browsing traffic that there was a corresponding reply PDU for every request PDU. On the other hand, for the video streaming application, we saw that a single request PDU was followed by a continuous stream of response PDUs at random intervals of time, while for the log push service, we observed a series of client PDUs at intervals without any reply from the server. We have designed our modeling system to capture any of these behaviors and model the traffic for each of these types of applications appropriately. In addition to the request-response exchange patterns, we also observed that traffic from various applications could be decomposed into user-sessions, connection clusters, connection pools, and request bursts,

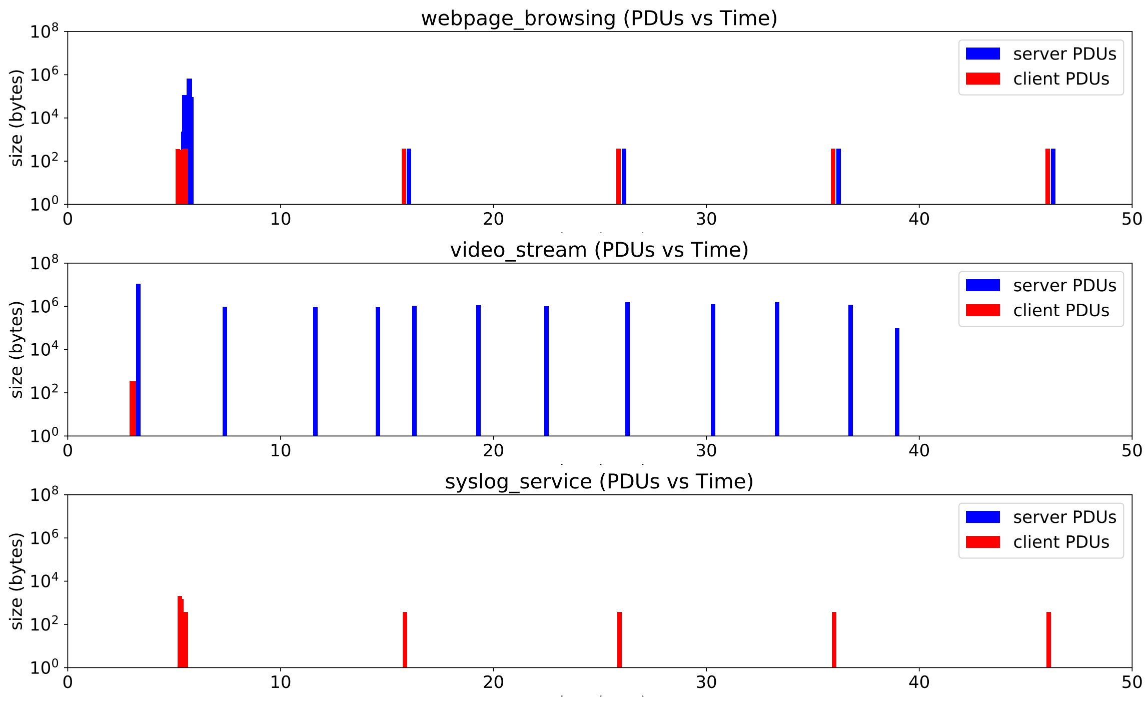


Fig. 4.6: Application data exchange patterns showing the request and response exchange nature in web browsing traffic, video streaming traffic, and syslog service traffic

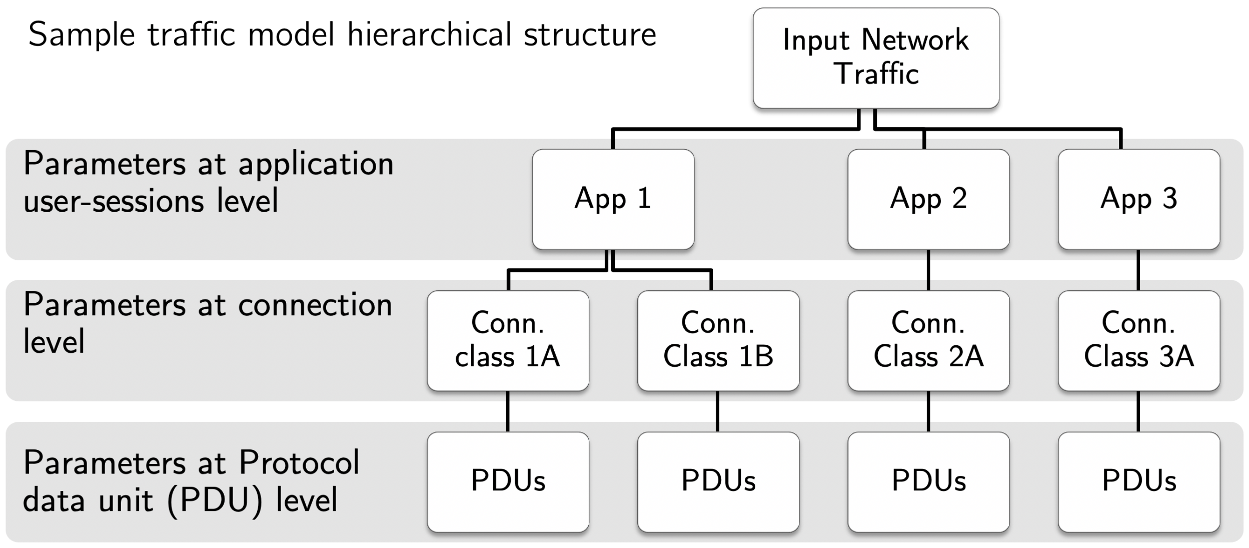

Sample traffic model hierarchical structure
Input Network Traffic
Parameters at application user-sessions level
App 1
App 2
App 3
Parameters at connection level
Conn. class 1A
Conn. Class 1B
Conn. Class 2A
Conn. Class 3A
Parameters at Protocol data unit (PDU) level
PDUs
PDUs
PDUs
PDUs

Fig. 4.7: Model parameters for 'uhgeneric-v4' traffic modeling method, showing the hierarchical structure of the modeling method.

as already discussed in sections 4.1.6 to 4.1.9.

The `uhgeneric-v4` modeling algorithm creates independent models for each application seen in the input traffic dataset. Each application's model is defined by a set of useful parameters for representing how data is sent and received over the network for any arbitrary application. The system determines average values, stochastic distributions, and empirical distributions that describe various parameters for the user-sessions and connection classes for each application being modeled from the input dataset. The hierarchical structure of the models created is given in Fig. 4.7.

The `uhgeneric-v4` modeling method consists of parameters at the application user-session level, connection level, and at individual PDU level. For each model parameter, we obtain an empirical model and the best stochastic distribution, then include them in the model file. We describe the parameters at each level of the hierarchical model below.

Parameters at application user session level include:

- **Connection Pool Inter Arrival** The time intervals between the starting times of succeeding connection pools.

- **Connection Pool Interval:** The interval between the end of a connection pool, and the beginning of the next connection pool in a user-session.
- **Has Overlapping Connections:** This parameter is a boolean flag that indicates if any overlapping connections were found for the application being modeled.
- **Inter Connection Class Sequences Distributions:** a probability distribution for each possible sequence of connection classes for user-sessions of an application.

Parameters at connection classes level include:

- **Port number:** The transport layer port number used by the server endpoint for sending and receiving data.
- **Layer4 Protocol:** The transport layer protocol (TCP or UDP) used by the connections in the application.
- **Number of Connections in Connection Pool:** The empirical distribution for the number of connections that are present in each connection pool for a specific connection class of an application.
- **Number of Request Bursts per Connection Pool:** The empirical distribution for the number of request bursts per connection.
- **Number of Requests in Request Bursts:** The empirical distribution for the number of individual requests in each request burst.
- **Inter Request Burst Time:** The empirical distribution for the interval between the start of successive request bursts.

Parameters at PDU level include:

- **Request Sizes:** The empirical distribution for the request (client PDU) sizes.
- **Number of Responses per Requests:** The empirical distribution for the number of distinct servers PDUs sent in response to each client PDU.
- **Response Sizes:** The empirical distribution for the response (server PDU) sizes.
- **Response Time Delay:** The empirical distribution for the server processing time before sending each response.

A sample of the `uhgeneric-v4` model for RDP traffic is given in appendix B. In addition to the traffic model file, the `uhgeneric-v4` modeling algorithm also creates an additional file containing a list of user-sessions found in the input dataset and their relative start times. This list of detected user-session can be used in conjunction with the traffic model file to generate traffic using our traffic generation system, which we describe in the next section below.

4.3 Traffic Generator

In our framework's traffic generator component, the main objective is to inject PDUs into the network for each application in the model based on the model's parameters. We designed the traffic generator system to send PDUs by faithfully following the pattern dictated in the model, thus generating traffic that is realistic for the applications modeled.

When using the traffic generation system for experimentation, we first set up a network on which to generate traffic. The main requirement is that the network must have nodes with IP addresses found in the input list of application user-sessions.

In a similar vein to the modeling system, the design of the traffic generator component in the framework allows easy extensibility to enable the addition of custom traffic generation algorithms. We implemented a packet generation method for the `uhgeneric-v4` model described earlier. We give a brief description of the modeling method below.

4.3.1 The `uhgeneric-v4` Traffic Generation Method

When traffic is being generated in server mode using the `uhgeneric-v4` model, the traffic generation method uses an in-band signaling technique to send control information from the client to the server. We fill the first few bytes of each request PDU sent from the client with data about the response(s) that the server is expected to send in reply. Hence, after opening the appropriate TCP and UDP listeners, the server endpoints receive incoming requests for each UDP listener and TCP connection. The server processes examine each request payload and determine the number of response PDUs to send, each response PDU's size, and the time delay before sending each response. The interaction between the client-side and the server-side processes of this traffic generation method is illustrated in Fig. 4.8.

When traffic is to be generated in client mode using the `uhgeneric-v4` algorithm, the system starts a new process for each user-session found to match the local interface IP address(es) in the user-session input file. For each application user-session process started, the process launches new connection pools at intervals dictated by the application model's user-session level parameters. In each connection pool, the system uses parameters at the connection pool level of the `uhgeneric-v4` model to generate packets. For each connection pool, the client process determines the number of TCP or UDP sessions in the pool, it determines the number of requests bursts to send, and starts a loop to send each request burst. For each request burst, the process determines the number of

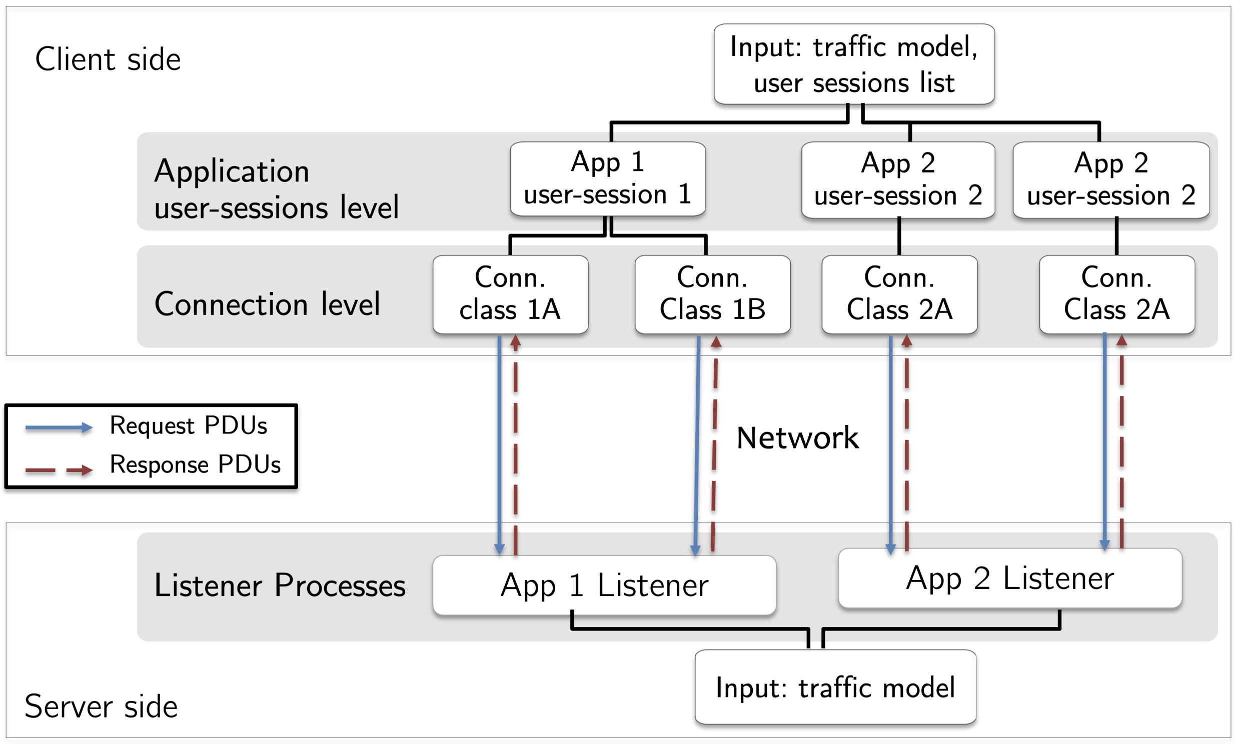


Fig. 4.8: Traffic generation using the 'uhgeneric-v4' traffic modeling method, showing the hierarchical structure, and the interaction between the client-side and the server-side of the traffic generation process

request PDUs to be sent and proceeds to send them using the initiated TCP or UDP sessions as illustrated in Fig. 4.8. For each request burst, the system calculates an inter request burst time from the model and waits for that inter request burst time before sending the next request burst.

At the individual request level, the system uses the `uhgeneric-v4` model's PDU level parameters to determine how to send each request and its replies. The system calculates a request size based on the distribution specified in the application model for each request. It also determines the number of responses to expect for that request, and then the size and response time delay expected for each response. It bundles all the response details in a stream of bytes and appends this with a random string of bytes to make up for the total calculated request PDU size. It then sends this request through the connection selected. Once the request is received at the server end of the connection, the server extracts the response details from the request and sends the expected responses back to the receiver. This completes the send-and-receive sequence for that request.

In our experiments, results, and discussion (chapter 5, we describe the results of our experiments to evaluate the traffic modeling and generation algorithms used. However, we first describe our evaluation system in the next section below.

4.4 Evaluation System

The final component of the framework is a comprehensive evaluation system. Synthetic network traffic workload is realistic for an application if it is similar to network traffic captured from the real application, in terms of metrics distributions and traffic patterns. Therefore, as part of our framework, we have designed an evaluation system that compares traffic generated based on the models with real traffic from modeled applications, to measure the effectiveness of traffic modeling and generation algorithms. We have implemented this evaluation system as a tool box that enables quantitative analysis of statistical similarity of input and generated traffic patterns [133]. It analyzes input network traffic trace(s) to provide detailed reports and graphs for many network metrics and can compare various metrics' distributions for two or more traffic traces. The evaluation system makes it possible quantify the level of confidence in the effectiveness of a realistic traffic model.

In the evaluation process, we first create a dataset as described in section 4.1 from the input traces. Next, we filter the dataset using the Ethernet addresses, IP addresses, port numbers, and application names, where necessary. Based on the filtered datasets, we carry out further analysis calculate several metrics for all applications in each input dataset to produce a report, along with tables and charts that reveal how the datasets compare based on metrics calculated. The reports, tables and charts contain baseline statistical characteristics for each metric calculated. The design makes it possible to compare multiple modeling algorithms by supplying generated traffic based on each modeling algorithm as input to the evaluation system, which produces a report with side-by-side comparisons for every metric for each modeling method to be compared. We discuss an example of this in section 5.3.2.

4.4.1 Research and Development of Metrics for System Evaluation

Many metrics are relevant to the evaluation of network traffic models. These metrics span across the physical layer, layer2, layer3, layer4, and the network protocol stack's application layer. A full list of the metrics we calculate in our evaluation process is presented in Table 4.1. We have distilled a list of metrics through a comprehensive review of several networking-related papers that utilize metrics specifically associated with network traffic workloads.

Table 4.1: List of metrics calculated by the evaluation system (`traffic_metrics`)

Single-Valued Metrics	Time-Variant Metrics
Total packets	Packet rate (packets per second)
Duration (seconds)	Inter-packet time (seconds)
	Throughput / data rate (bits per second)
	Packet size (bytes)
L2 total payload (bytes)	L2 payload throughput (bits per second)
L2 total vlan packets	L2 payload size (bytes)
L2 number of vlans	
L3 total payload (bytes)	L3 payload throughput (bits per second)
L3 fragmented packets	L3 payload size (bytes)
L4 total payload (bytes)	L4 payload throughput (bits per second)
L4 total retransmitted packets	L4 payload size (bytes)
L4 total out of order packets	L4 retransmission rate (packets per second)
L4 total acknowlegement packets	L4 out of order rate (packets per second)
L4 total push packets	L4 window size (bytes)
	L4 session inter arrival time (per second)
L4 Total connections	L4 connection inter-arrival time
	L4 connection arrival rate
App total PDUs	App PDU size (bytes)
	App inter-PDU time (seconds)
	App PDU rate (PDUS per second)
	App PDU throughput (bits per second)
	App server response time (seconds)

1. L2, L3, L4 and App represents physical layer2, layer3, layer4 and application layer in the TCP/IP network stack respectively.

Since there are many networking metrics, the most important metric for evaluating the performance of a realistic workload generator for a given experiment will depend on the specific experimental goals and expected traffic patterns typical to the application. We provide a detailed analysis of the metrics selection process in the next section (section 4.4.2).

The metric quantities in Table 4.1 are in two categories: single-valued metrics and time-variant metrics. For the single value metric, we calculate their values for each user-session of the input traffic dataset. For each of the time-varying metrics, the system calculates the metric values per second or per packet in each application user-session. The calculation process involves a sliding window time analysis at selected time resolutions for some of the time-variant metrics. The default time resolution used in our evaluation system is 0.1 secs which is possible to adjust per application traffic requirements.

4.4.2 Metrics Selection for Realistic Traffic Generator Evaluation

A traffic modeling system performs well if there is a close similarity in distributions of values between generated traffic and real application traffic for as many metrics as possible. In our evaluation of traffic models in chapter 5, one of our goals is to ensure that our traffic modeling algorithm enables the generation of realistic traffic that matches real traffic patterns at the application layer. We try to avoid using metrics whose values may have been influenced by device lower-level network protocol stack implementations.

For example, in experiments where the goal is evaluating transport layer load-balancing algorithms for an application, the critical metrics may be Packet Rates and connection inter-arrival time. Hence when evaluating a realistic traffic modeling algorithm for such a workload, the distributions of packet rates and connection inter-arrival time metrics must be as close as possible to real traffic from the application.

As another example, in an experiment that requires a realistic traffic workload for a web browsing application, it will be essential to consider the patterns of requests and responses seen. Hence the requests per second (or PDUs per second) and the server response time metrics will be critical in evaluating the realistic traffic modeling and generation method for such an experiment.

Similarly, in an experiment that requires realistic workload based on the secure shell (SSH) protocol traffic, the distribution of packet sizes and packets rate will be very important. This is because, the SSH protocol can be used as a base for many upper level applications, including remote login shell, secure file transfer and X11 remote desktop viewer applications. The pattern of packet sizes and packet rates differ for each of these applications, even though they use the same SSH protocol. Hence in experiments where realistic SSH based traffic is required, good choices of metrics for evaluating the performance of the realistic workload generator used will include distributions of packet sizes and packet rates.

We focus mainly on the application-level network metrics that are important for realistic network traffic generator evaluations. We describe each of these metrics below.

4.4.2.1 Inter-PDU Times Distribution (Seconds)

Inter-PDU time measures the time interval between successive PDUS for an application, as seen at a point on a network.

4.4.2.2 PDU Size Distributions (Bytes)

The PDU size distributions give a measure of the sizes of PDUs for an application, arriving at a point on the network.

4.4.2.3 Data Rate or Throughput Distribution (Bytes per Seconds, bps)

Data rate or throughput measures the rate of data for an application flowing across a node per unit of time. It indicates the amount of data in bytes per second sent from the source(s) flowing through a node or link on a network.

4.4.2.4 PDU Rate Distribution (PDUs per Seconds)

PDU rates are a measure of the number of PDUs per unit time, seen at a node. It indicates the amount of PDUs flowing through a node or link on a network per second.

4.4.3 First-Order Analysis

For each of the time-variant metrics in section 4.4.2 and Table 4.1, we calculate the statistics, including the mean, median, maximum, minimum, inter-quartile range (IQR), standard deviation (SD), skewness, kurtosis, and Hurst exponent. Our evaluation system presents the calculated statistics for both the generated traffic and real traffic side by side metrics in tabular format. We also create various charts, including cumulative frequency curves, box plots, and frequency histograms, for both generated and real traffic, to compare each of the metrics.

As part of our analysis, we also calculate statistical measures that directly compare each metric quantity's distributions for all input traces. For example, we evaluate the values of Kolmogorov-Smirnov two sample (KS-2-sample) test statistic [80]. Based on these, we can easily judge the similarity of a metric for two or more traces. KS-2-sample tests are often used to test the level of similarity between two empirical distributions. An illustration of the KS-2-sample statistic is given in Fig. 4.9. The test statistic value obtained from the test gives a numerical representation of the statistical distance between two distributions. Hence a lower KS-2-sample test statistic value indicates a close fit between the two distributions that are being compared.

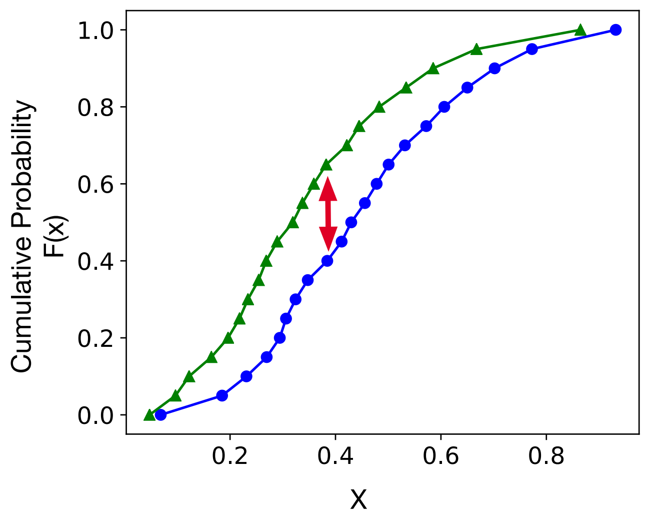


Fig. 4.9: Illustration of the Kolmogorov-Smirnov 2-sample (KS-2-sample) test statistic. The blue and green lines represent the empirical cumulative frequency curves of the quantities being compared, while the red arrow represents the KS-2-sample test statistic.

4.4.4 Second-Order Analysis

In typical application network traffic, metrics from different user-sessions of the same applications usually have some variation in the distribution of their values. This variation in network traffic distributions of the same application can sometimes be due to random user behavior. The synthetically generated realistic traffic must also exhibit this behavior of randomness in metrics for multiple user-sessions. We evaluate this behavior using the kolmogrov-sminorff two sample (KS-2-sample) tests [80] statistic values as a second-order measure to compare distributions of each metric per user-session pair in the input trace files. If there are many user-sessions in both the original and generated traffic, a good traffic model must produce generated traffic with a distribution of pairwise user-session KS-2-sample test statistic for each metric that is similar to that of the corresponding real application traffic.

For each metric, we determine the KS-2-sample test statistic based on the metric values for each pair of user-sessions in the generated traffic, and in the real application traffic. For a selected metric quantity, we create a distribution of these pairwise KS-2-sample test statistics for all user-session pairs in the real traffic and plot the cumulative frequency curves, box-plots, and frequency histograms. An illustration of these second order analysis process is given in Fig. 4.10.

For example, in our experiment on single network application traffic (section 5.1) where we evaluate our modeling and generation method based on the PDU rate metric, the input dataset is made up of traffic for 20 web browsing user-sessions. We calculate the PDU rates (as a time series data) for all 0.1 second intervals in each user-session. For this second order analysis, we then compare user-session based on how the PDU rate spreads in the input trace, by calculating the KS-2-sample values of the PDU rates for each pair-of user-sessions. We then create cumulative frequency curves, box plots and histograms for the distribution of this KS-2-sample values. After creating a traffic model from the input trace and generating traffic with the model for 20 distinct user sessions, we also extract the PDU rates for each user-session and calculate the KS-2-sample values of the PDU rates for each pair of user-sessions. We also create cumulative frequency curves, box plots and histograms from the distribution of this KS-2-sample values, and compare this side-by-side with what was obtained from the original traffic. Depending on the evaluation metric selected, a good model should produce traffic with similar distributions as that of the original traffic when the cumulative frequency curves, box plots and histograms are compared.

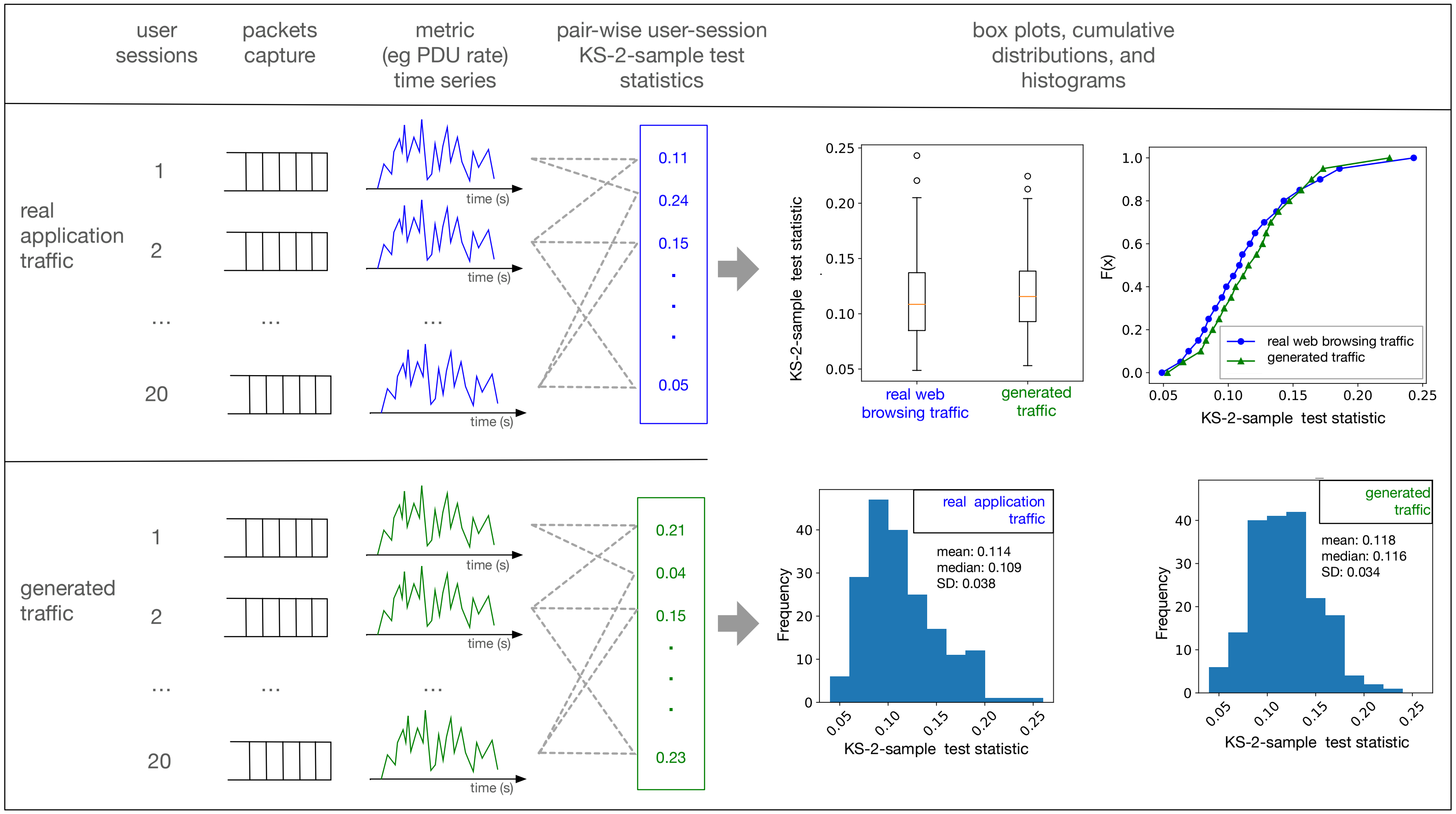


Fig. 4.10: Illustration of the second order evaluation process for evaluating performance of realistic traffic generation algorithms. First, metric time series values are calculated from each user-session. The KS-2-sample test statistics are then calculated for each pair of user sessions in the real and generated traces. The distributions of these calculated statistics are then plotted in box plots, cumulative frequency curves and histograms to compare the second order spread of the values of the selected metric across the real and generated traces.

4.5 Chapter Summary

In this chapter, we have described each component of our framework for traffic modeling and generation. We have also described our evaluation system and reported how it could be used to analyze and compare network traffic. In the next chapter, we focus on specific evaluation experiments we carried out to demonstrate the utility of the framework and evaluate our `uhgeneric-v4` modeling method.

Chapter 5

Evaluation Experiments and Discussion

We carried out several experiments on the framework, to evaluate the performance of our `uhgeneric-v4` method in modeling traffic of a diverse set of applications. In our experiments we used the global environment for network innovations (GENI) [18] virtual topology service(VTS) [16]. VTS is a distributed system that enables the creation of arbitrary isolated network topologies with any number of nodes and links of any bandwidth and delay. VTS can create networks that have actual servers, virtual machines, containers with any operating system, in addition to OpenVswitches, and routers.

This chapter discusses the results of evaluation experiments with single application traffic and multi-service application traffic. We also discuss the results of our experiment to compare multiple modeling methods for traffic generation. Our experiments' outcomes on the `uhgeneric-v4` model, with various applications, show that traffic generated by the framework and modeling method is similar to real application traffic for the metrics examined. We present our results for these experiments in the corresponding subsections below.

5.1 Modeling and Generating Single Application Traffic

We started our evaluation with experiments focused on evaluating the performance of the framework and the modeling method in generating realistic traffic for single application network traffic.

We carried out experiments to model and generate traffic for four different applications:

- Web browser client traffic
- Remote desktop protocol (RDP) traffic
- Secure shell (SSH) traffic
- HTML 5 video streaming traffic

One of our goals in creating the `uhgeneric-v4` modeling algorithm was to support the generation of realistic traffic for any application. Therefore we carried out experiments with multiple applications to investigate how the modeling algorithm performed in generating traffic for different classes of network traffic. We selected the set of applications listed above because they represent some of the broader categories of traffic popular on many networks today. We used the framework and our `uhgeneric-v4` modeling algorithm to create traffic models for each of the applications. The models created were then used to generate packets in our testbed environment. We evaluated the traffic modeling and generation algorithm's performance by comparing generated traffic with real traffic for each application.

5.1.1 Input Network Traffic Data

Real network traffic for each of the applications itemized in section 5.1 above was obtained through traffic captures on networks that had endpoint devices running multiple user-sessions of each application.

For example, to obtain an input traffic trace for modeling in the case of the web browser application, we set up a network, as shown in Fig. 5.1 in our laboratory environment. The network had a `nginx` web server loaded with web pages of a read-the-docs [106] documentation website. The client machine on the same network was used to send automated page requests for the documentation web pages at random time intervals using a `selenium` web automation driver [117], thus simulating a typical user. Another node in the network was used as a `syslog` server to store log messages pushed from the `nginx` web server. The client was made to send requests and receive replies for about twenty minutes, while the monitor node captured the resulting network packets through a span port on the network bridge. We repeated this twenty times to obtain a network traffic trace file containing traffic for twenty user-sessions for typical browser traffic for a text-based web documentation application.

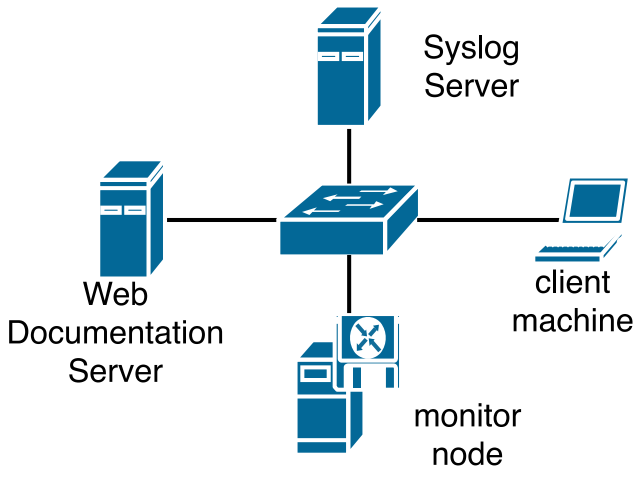


Fig. 5.1: Experiment topology for the web browser application traffic generation experiment

Similar networks were created to capture real network traffic from remote desktop, secure shell, and HTML 5 video streaming applications. In each of them, we captured traffic for as many as twenty user-sessions of the application. These trace files for each of the four applications were then used as input to the modeling and traffic generation framework.

5.1.2 Dataset Extraction, Traffic Modeling, and Generation

We went through the traffic modeling and realistic traffic generation process for each application traffic trace files captured. We applied each traffic trace file to the dataset extractor to yield a `csv` dataset file containing decoded packet headers and additional relevant fields (including detected application, user-session, and connection class) for every packet in the trace file, as described in section 4.1. We examined the resulting dataset to ensure that the decoded data matches expected values. In a few cases, we ran a utility script within the dataset extractor to modify the dataset values based on adjustments to an auto-generated metadata file, in cases where the obtained values did not match the expected information.

We passed the resulting datasets for each application into the modeling system component of the framework. For each input application traffic dataset, the outcome of the traffic modeling phase is a traffic model `json` file based on the `uhgeneric-v4` modeling algorithm that has values and stochastic distributions for parameters at the user-session level, connection class level, and individual PDU level as described in section 4.2.1.

We created replicas of the original networks where the traces were captured within our VTS testbed [16]. (For example, for the web browser application experiment, we created a replica of the network in Fig. 5.1 with the VTS testbed.) Using the traffic generator component of our framework, and our repeatable experiment orchestration framework [48], we generated traffic on each host of the test network, based on the traffic model, for twenty user-sessions of the application. The traffic generation was done in a simulated mode to ensure accurate evaluation of PDU arrival times, avoiding packet/PDU processing delays that may be caused by the operating system network stack. The result of the packet generation process is a dataset of PDUs, and corresponding parameters pushed into the network stack by each application user-session process for each application

Table 5.1: Evaluation results for modeling and generation of web documentation application traffic

Metric	Median		Mean		Maximum		Standard Deviation		KS-2-Sample
	Real Traffic	Generated Traffic	Real Traffic	Generated Traffic	Real Traffic	Generated Traffic	Real Traffic	Real Traffic	Test
Inter-PDU Times (seconds)	0.0004	0.0004	0.0148	0.0148	76.732	108.2390	0.0346	0.0346	s=0.049
PDU Sizes (bytes)	386	386	345	347	656568	656568	124	146	s=0.044
PDU Rates (PDUs per second)	20.00	20.00	38	37	169.00	191.0	38	28	s=0.088
PDU Throughput (bps)	6.19e+4	9.11e+4	1.60e+5	1.60e+5	2.90e+8	1.50e+8	1.60e+5	1.83e+5	s=0.210
Server Response Times (seconds)	0.0002	0.0002	0.0004	0.0004	0.004	0.004	0.0007	0.0007	s=0.041

considered in our experiments. This generated traffic dataset was compared with the real input application traffic dataset in the evaluation phase, which we describe below.

5.1.3 Evaluation and Discussion

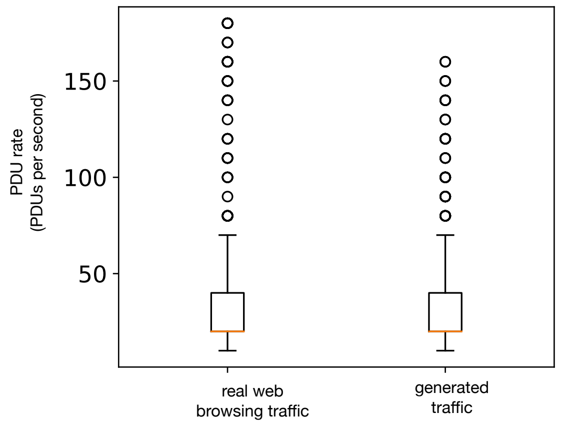
To evaluate the performance of the `uhgenric-v4` method in traffic modeling and generating realistic traffic, we compared the generated traffic based on the model with real traffic for each application (web browser, SSH, RDP, and video streaming applications) using our evaluation system. We extract 'per packet' values or 'per second' values at 0.1 second intervals for each of the metrics described in section 4.4.2 on both real and generated traffic. We use various statistical measures to make comparisons between them.

5.1.3.1 First-Order Comparison of Metric Distributions

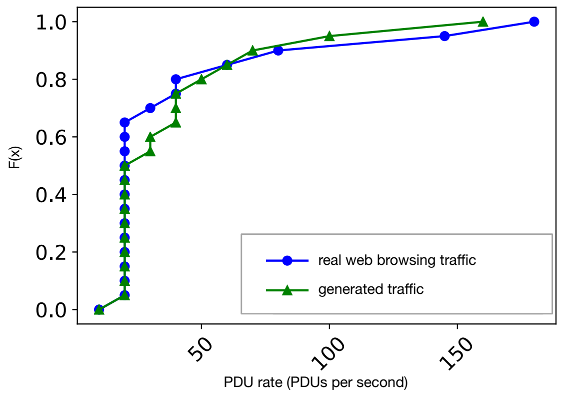
We started with a direct first-order comparison of metric values for both real and generated traffic. We compared each metric's distribution based on statistical measures (including the median, mean, maximum, minimum, and standard deviations). For example, in our experiments on modeling and generating realistic packets for a web browser application, the tabulated side-by-side statistics for each metric of the real and generated traffic is given in Table 5.1.

In the Table 5.1, the median values for each of the metrics are comparable. The median PDU size of 388 bytes in the real application trace file is exactly the same as the 386 bytes seen in the generated traces. Similarly, the median inter-PDU time of 0.004 seconds in the real trace is the same as in the generated traces. The same applies to the median values of the PDU rate, PDU throughput, and server response time metrics. The mean, maximum, and standard deviation of each metric of generated traffic in the Table 5.1, have some values that are different from those of the real traffic, but are quite close, and still realistic for real web browsing traffic. For example, 342 bytes in the mean PDU size of generated traffic is possible for a realistic web browsing workload, even though it is quite different from the value of 345 bytes seen in the real trace. The same applies to many of the other metrics' mean, maximum, and standard deviation values. The low KS-2-sample test statistic values (< 0.3) for each metric also indicates a close similarity between the empirical distributions of generated traffic and the real traffic.

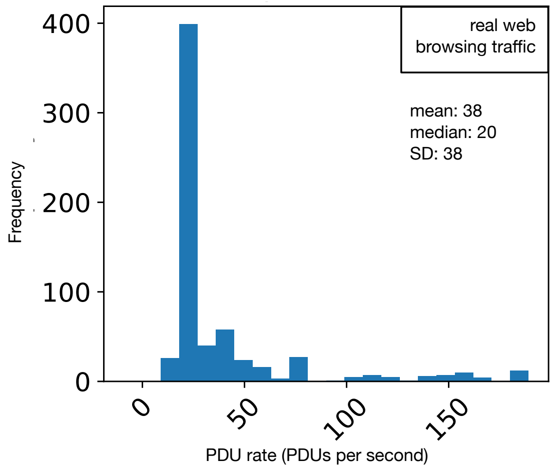
When evaluating our traffic modeling and generation algorithm's performance for web browsing traffic, it is important to consider metrics related to the patterns of requests and responses seen. In many experimental scenarios, HTTP based web application traffic is often characterized in terms of outgoing request PDUs per second sent by a client (or incoming requests per second seen at a server). In addition, the response PDU rates sent by the server, and the server response times, are also very important metrics in experiments involving HTTP workloads. Therefore, the distributions of PDUs (requests and responses) per second and the server response time metrics are very important in evaluating the realistic traffic modeling and generation method for experiments that utilize HTTP traffic. Hence, we create the corresponding box plots, cumulative frequency curves, and histograms to compare each metrics' distribution for both real web browsing traffic and generated traffic based on our model. These plots for the PDU rate metric are given in Figs. 5.2a, 5.2b, 5.2c and 5.2d.



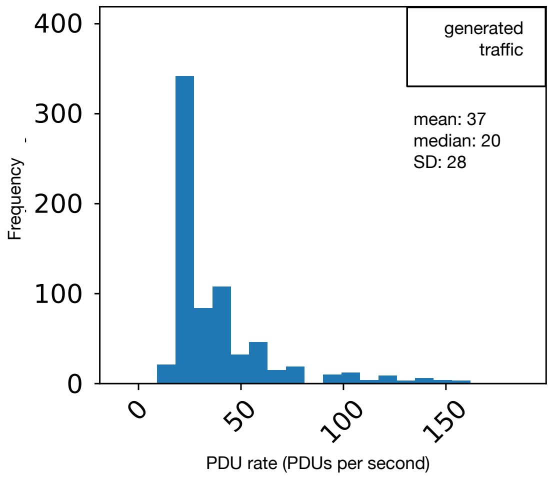
(a) Box plots comparing PDU rate distributions for real web browsing traffic and generated traffic based on the `uhgeneric-v4` modeling algorithm



(b) Cumulative frequency curves comparing PDU rate distributions for real web browsing traffic and generated traffic based on the `uhgeneric-v4` modeling algorithm



(c) Histogram of PDU rate distribution for real web browsing traffic



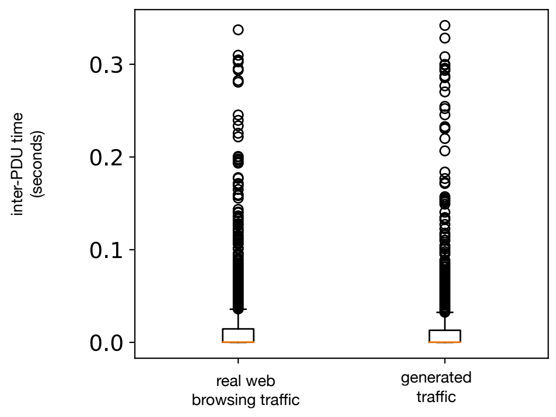
(d) Histogram of PDU rate distribution for generated web browsing traffic based on the `uhgeneric-v4` modeling algorithm

Fig. 5.2: Box plots, cumulative frequency curves and histograms comparing PDU rate distributions of real web browsing traffic with PDU rate distributions of generated web browsing traffic based on the `uhgeneric-v4` modeling algorithm

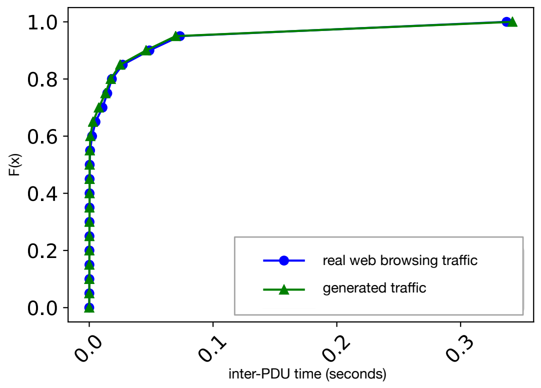
The box plots in Fig. 5.2a show that the generated traffic PDU rate has the same median and interquartile ranges as in the real web browsing traffic. The cumulative frequency curves in 5.2b also show that the PDU rates have comparable distributions; however, it also reveals that distributions do not perfectly overlap. The histograms in Figs. 5.2c and 5.2d show the very slight differences in the PDU rate mean, median, and standard deviation values between real and generated traffic. The real web browsing traffic has a mean of 38 PDUs per second, while the generated traffic has 37 PDUs per second. The median values are equal (20 PDUs per second respectively.) while the standard deviations are 38 PDUs per second and 28 PDUs per second respectively. A look at the shape of both histograms also shows how similar the distributions are.

Other metrics that are also relevant in characterizing HTTP based web browsing workloads include the inter-PDU time, PDU sizes, and throughput. The corresponding box plots, cumulative frequency curves, and histograms for the first-order comparison of all these metrics (including the PDU rates discussed above) are given in Figs. 5.3a to 5.6d.

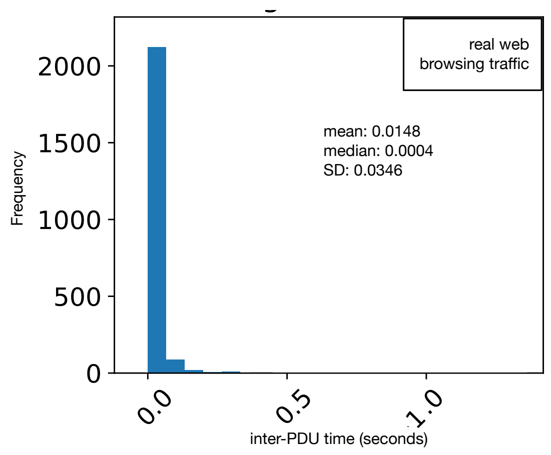
The box plots for the inter-PDU time in Fig. 5.3a show that the generated traffic inter-PDU time has the same median and interquartile ranges as in the real web browsing traffic. The box plots also exhibit matching outlier ranges. The cumulative frequency curves plotted in Fig. 5.3b also show that the distribution of the inter-PDU times of the generated traffic closely overlaps that of the real web browsing traffic. The histograms (Figs. 5.3c and 5.3d) show exactly identical values of 0.0148 seconds in the mean, 0.0004 seconds in the median, and 0.0346 seconds in the standard deviation of inter-PDU times for both real web browsing traffic and the generated traffic.



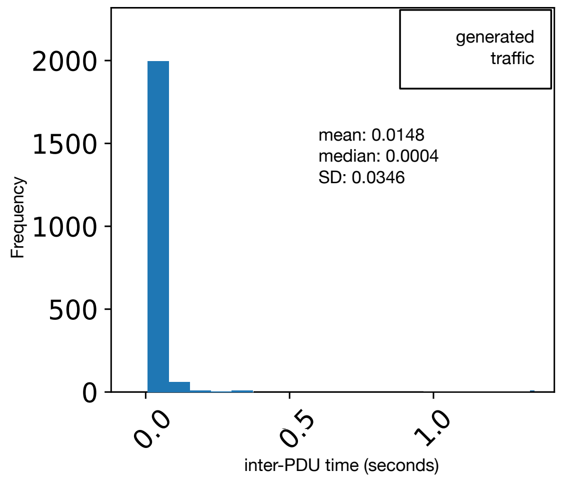
(a) Box plots comparing inter-PDU time distributions of real web browsing traffic and generated traffic based on the `uhgeneric-v4` modeling algorithm



(b) Cumulative frequency curves comparing inter-PDU time distributions of real web browsing traffic and generated traffic based on the `uhgeneric-v4` modeling algorithm



(c) Histogram of inter-PDU time distribution for real web browsing traffic

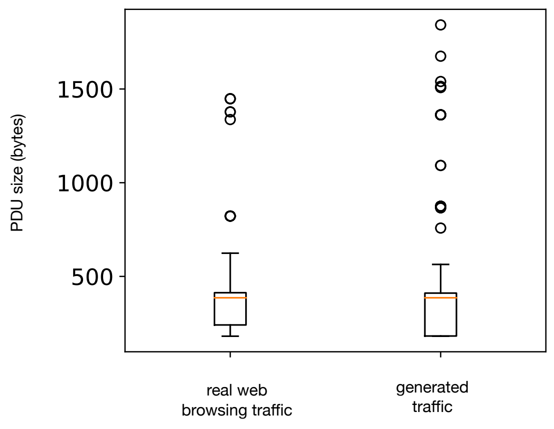


(d) Histogram of inter-PDU time distribution for generated web browsing traffic based on the `uhgeneric-v4` modeling algorithm

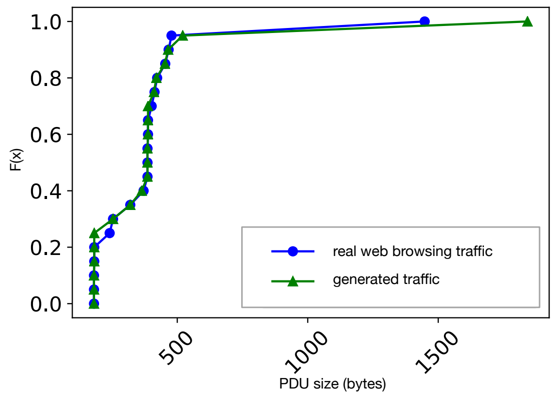
Fig. 5.3: Box plots, cumulative frequency curves and histograms comparing inter-PDU time distributions of real web browsing traffic with inter-PDU time distributions of generated web browsing traffic based on the `uhgeneric-v4` modeling algorithm

For the PDU size metric, the box plots in Fig. 5.4a shows that the generated traffic PDU size metric also has identical interquartile ranges as in the real web browsing traffic. The close overlap in the cumulative frequency curves plotted in Fig. 5.4b show that the distribution of the PDU size of the generated traffic also closely resembles that of the real web browsing traffic. In the corresponding histograms (Figs. 5.4c and 5.4d), the mean values of 345 bytes and 347 bytes in real and generated traffic are very close. The median values are equal (386 bytes). The standard deviations values of 146 bytes in the generated traffic histogram, is quite different to the value of 124 bytes in the real web browsing traffic, but still valid to pass for realistic web browsing traffic.

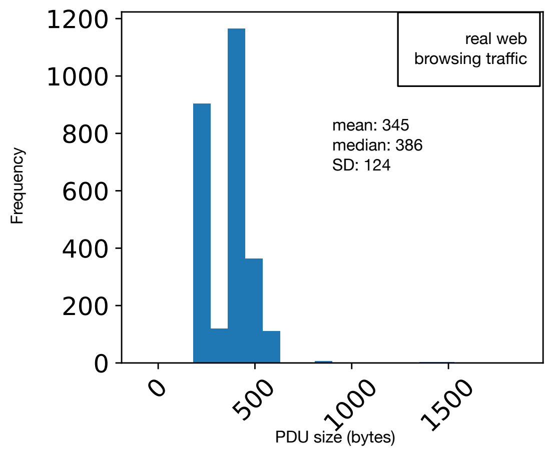
Fig. 5.5a also reveals matching interquartile ranges for PDU throughput between the real and generated traffic. The cumulative frequency curves plotted in Fig. 5.5b also show that the distribution of the PDU throughput for the generated traffic closely overlaps that of the real web browsing traffic. The histograms (Figs. 5.5c and 5.5d) show that mean, median and standard deviation values across both real web browsing traffic and the generated traffic are always in the same order. The mean of 160,299 bps in generated traffic is very close to the mean of 160,025 bps in the generated traffic. The median of 91,120 bps in generated traffic is of the same order as the median of 61,000 in the generated traffic. And the standard deviations of 160,600 and 183,071 bps are also in the same order.



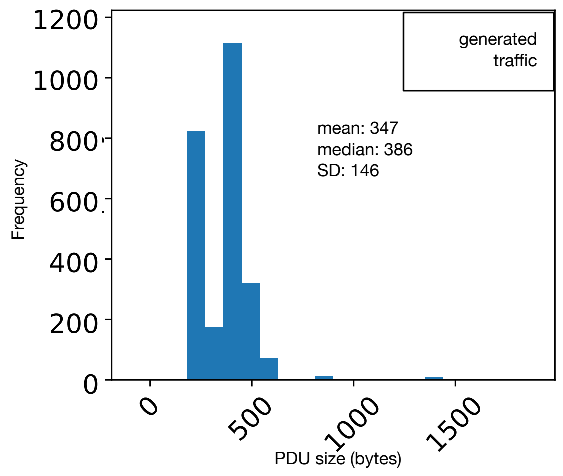
(a) Box plots comparing PDU size distributions of real web browsing traffic with PDU size distributions of generated web browsing traffic based on the `uhgeneric-v4` modeling algorithm



(b) Cumulative frequency curves comparing PDU size distributions of real web browsing traffic with PDU size distributions of generated web browsing traffic based on the `uhgeneric-v4` modeling algorithm

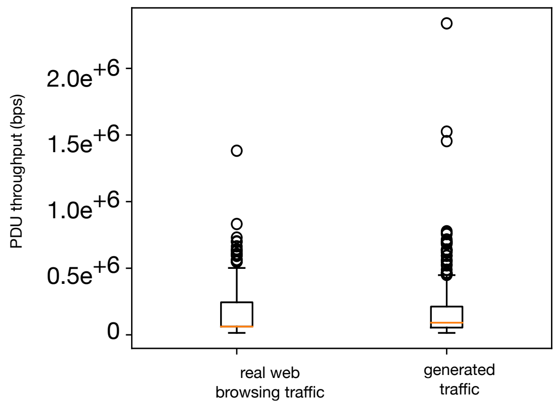


(c) Histogram of PDU size distribution for real web browsing traffic

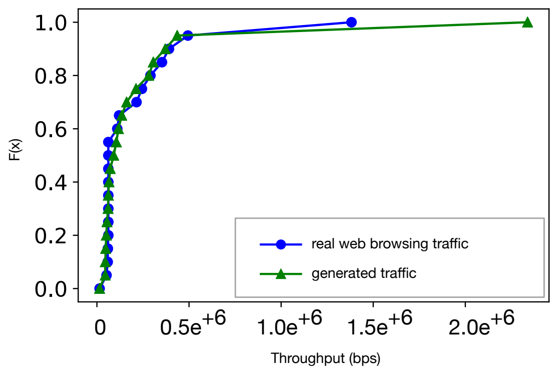


(d) Histogram of PDU size distribution for generated web browsing traffic based on the `uhgeneric-v4` modeling algorithm

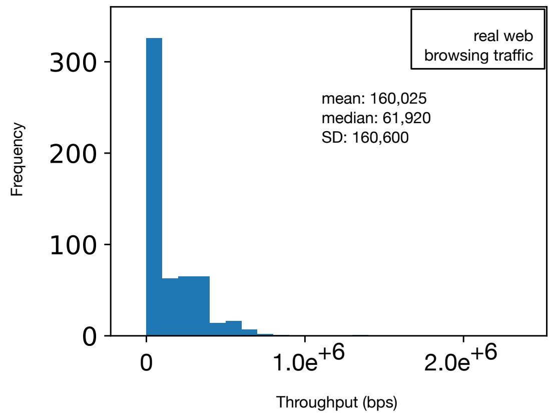
Fig. 5.4: Box plots, cumulative frequency curves and histograms comparing PDU size distributions of real web browsing traffic with PDU size distributions of generated web browsing traffic based on the `uhgeneric-v4` modeling algorithm



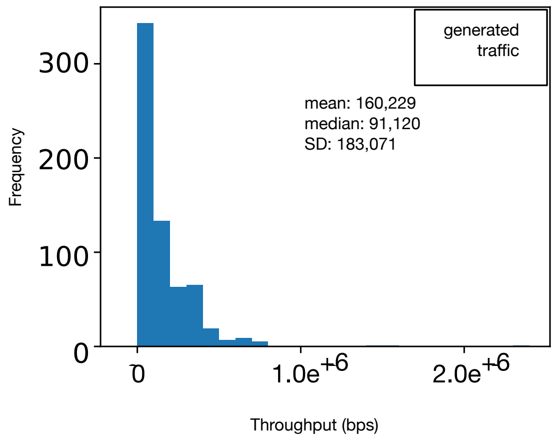
(a) Box plots comparing PDU throughput distributions of real web browsing traffic with PDU throughput distributions of generated web browsing traffic based on the `uhgeneric-v4` modeling algorithm



(b) Cumulative frequency curves comparing PDU throughput distributions of real web browsing traffic with PDU throughput distributions of generated web browsing traffic based on the `uhgeneric-v4` modeling algorithm



(c) Histogram of PDU throughput distribution for real web browsing traffic

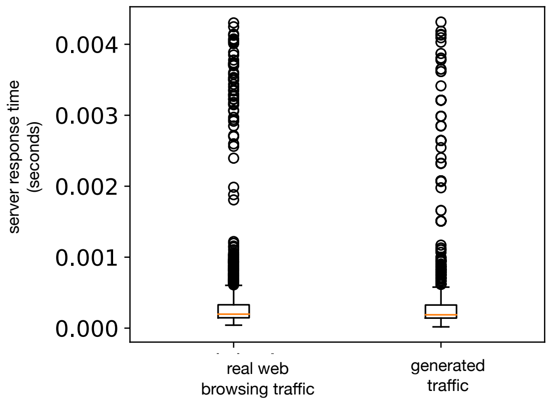


(d) Histogram of PDU throughput distribution for generated web browsing traffic based on the `uhgeneric-v4` modeling algorithm

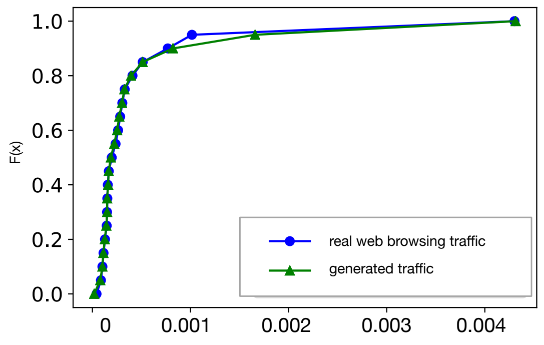
Fig. 5.5: Box plots, cumulative frequency curves and histograms comparing PDU throughput distributions of real web browsing traffic with PDU throughput distributions of generated web browsing traffic based on the `uhgeneric-v4` modeling algorithm

The box plots for the server response times in Fig. Fig. 5.6a indicate that the generated traffic server response times has the identical median and interquartile ranges as in the real web browsing traffic. The box plots also exhibit matching outlier ranges. The cumulative frequency curves plotted in Fig. 5.6b also show that the distribution of the server response times of the generated traffic closely overlaps that of the real web browsing traffic. In addition, the histograms (Figs. 5.6c and 5.6d) show exactly the same values of 0.0004 seconds in the mean, 0.0002 seconds in the median, and 0.0007 seconds in the standard deviation of server response times for both real web browsing traffic and the generated traffic.

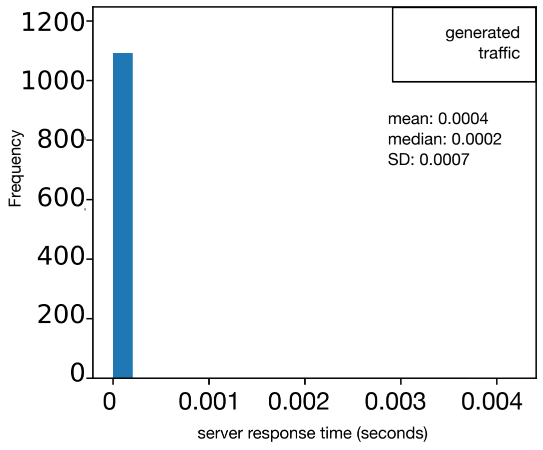
The close similarity between the distributions of these metrics (especially PDU rates, and server response times) discussed above, for generated traffic and the real web browsing traffic, indicates that the `uhgeneric-v4` model is very effective in modeling and generating realistic traffic for HTTP-based web applications. Analogous results were also obtained in our evaluation experiments with RDP, SSH, and video streaming traffic.



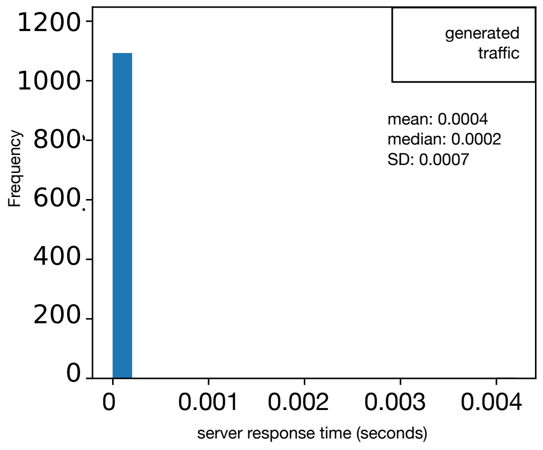
(a) Box plots comparing server response time distributions of real web browsing traffic with server response time distributions of generated web browsing traffic based on the `uhgeneric-v4` modeling algorithm



(b) Cumulative frequency curves comparing server response time distributions of real web browsing traffic with server response time distributions of generated web browsing traffic based on the `uhgeneric-v4` modeling algorithm



(c) Histogram of server response time distribution for real web browsing traffic



(d) Histogram of server response time distribution for generated web browsing traffic based on the `uhgeneric-v4` modeling algorithm

Fig. 5.6: Box plots, cumulative frequency curves and histograms comparing server response time distributions of real web browsing traffic with server response time distributions of generated web browsing traffic based on the `uhgeneric-v4` modeling algorithm

5.1.3.2 Second-Order Comparison of Metric Distributions

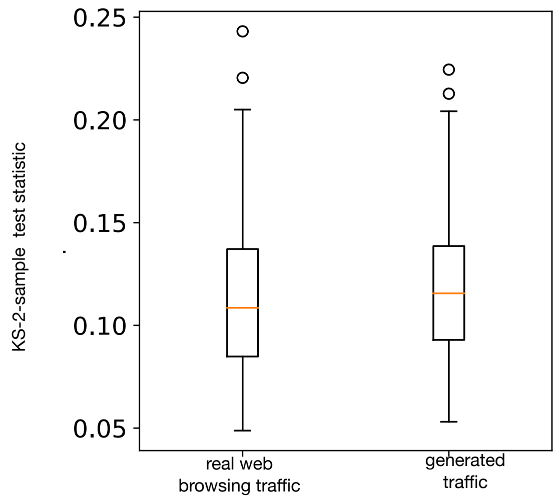
In typical application network traffic, metrics from different user-sessions of the same applications usually vary due to random user behavior. That is, distributions of each metric usually differ between user-sessions of the same application traffic. Synthetically generated traffic must also exhibit this behavior of randomness. Therefore, we performed second-order comparisons to examine how each metric's distributions vary within user-sessions of the real traffic and generated traffic.

These second order comparisons are based on the distributions of calculated Kolmogorov-Smirnov 2 sample (KS-2-sample) test statistics for user session pairs in both real traffic, and generated traffic. KS-2-sample tests are typically used to compare two empirical distributions, and the resulting test statistic value gives a numerical representation of the statistical distance between two empirical distributions. Hence a lower KS-2-sample test statistic value indicates a close fit between the two distributions that are being compared. Therefore in the case where we have many user-sessions in both the original and generated traffic, when evaluating a good traffic model, the distribution of pairwise user-session KS-2-sample test statistic for each metric in the original trace must be comparable to the distribution of pairwise user-sessions KS-2-sample test statistics for the same metric in the generated traffic.

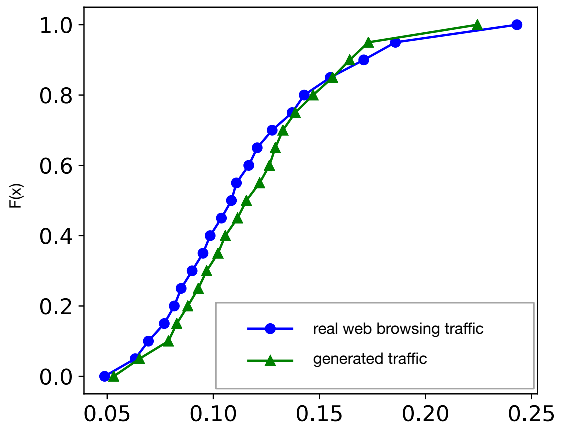
In this second order evaluation process, we first calculate the 'per packet' values or the 'per second' values at 0.1 second intervals for each user-session, and for each metric in section 4.4.2, using the real traffic dataset. For each metric and each pair of user-sessions, we go on to determine the KS-2-sample test statistic using the metric values from that pair of user-sessions. For a selected metric quantity, we create a distribution of these pairwise KS-2-sample test statistics for all user-session pairs in the real traffic and plot the cumulative frequency curves, box-plots, and frequency histograms. We then repeat the same process for the generated traffic, plotting the cumulative frequency curves, box-plots, and frequency histograms on shared axes. The plots give a measure of the second-order spread or variation of the metric values within the user-sessions of the real traffic and within the user-sessions of the generated traces. An illustration of these second order analysis process has been given in Fig. 4.10 of chapter 4. For our experiment on modeling and generating realistic traffic for the web browser application, the plotted results are provided in Figs. 5.7a to 5.11d.

The box plots in Fig. 5.7a indicate that the pairwise user-sessions KS-2-sample test statistics for the inter-PDU time in the generated traffic has similar median and interquartile ranges as found in the real web browsing traffic. The cumulative frequency curves plotted in Fig. 5.7b also show that the distribution of the inter-PDU times KS-2-sample test statistics of the generated traffic closely overlaps that of the real web browsing traffic. The histograms (Figs. 5.7c and 5.7d) show almost identical values of 0.114 and 0.118 in the mean values. And the standard deviation values of 0.038 and 0.034 are also very close.

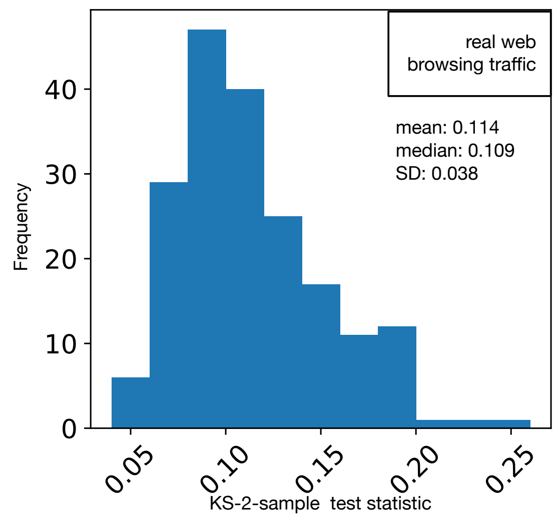
For the PDU size metric, the box plots in Fig. 5.8a show that the pairwise user-sessions KS-2-sample test statistics for the generated traffic PDU size metric has interquartile ranges that is fairly close to that of the real web browsing traffic. The close overlap in the cumulative frequency curves plotted in Fig. 5.8b show that the distributions of the KS-2-sample test statistics for PDU size of the generated traffic also closely overlaps that of the real web browsing traffic. In the corresponding histograms (Figs. 5.8c and 5.8d), the mean, median, and standard deviation values of 0.120, 0.116, and 0.034 in the generated traffic are not the same as the values of 0.091, 0.078 and 0.051 seen for the real traffic. However the values in the generated traces are close enough to pass for generated realistic web browsing traffic.



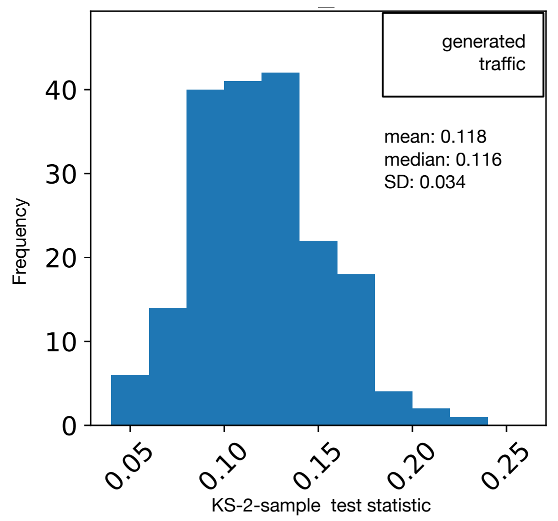
(a) Box plots comparing distributions of pairwise user-sessions KS-2-sample test statistics for the Inter-PDU time metric of real and generated web browsing traffic based on the `uhgeneric-v4` model



(b) Cumulative frequency curves comparing distributions of pairwise user-sessions KS-2-sample test statistics for the Inter-PDU time metric of real and generated web browsing traffic based on the `uhgeneric-v4` model

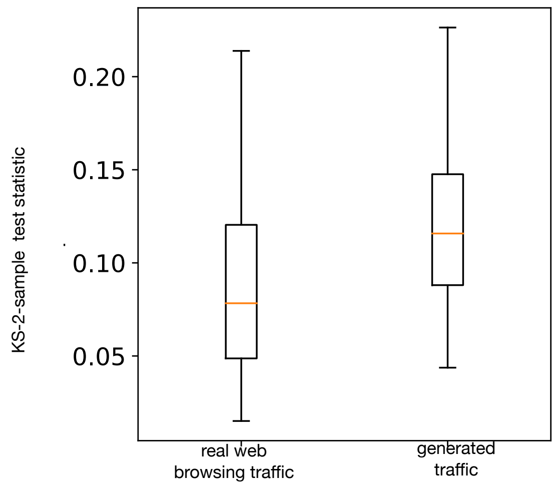


(c) Histogram of pairwise user-sessions KS-2-sample test statistics for the Inter-PDU time metric of real web browsing traffic

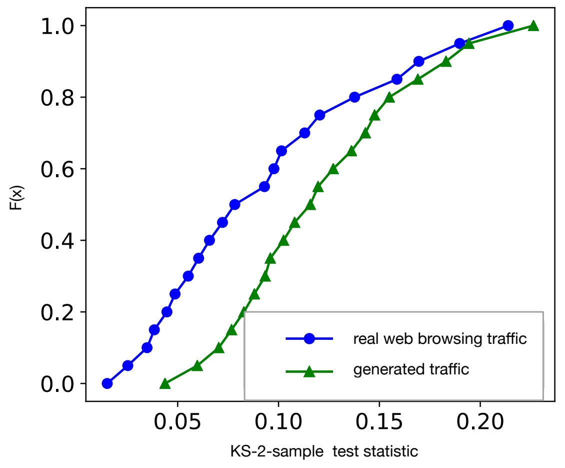


(d) Histogram of pairwise user-sessions KS-2-sample test statistics for the Inter-PDU time metric of generated web browsing traffic based on the `uhgeneric-v4` model

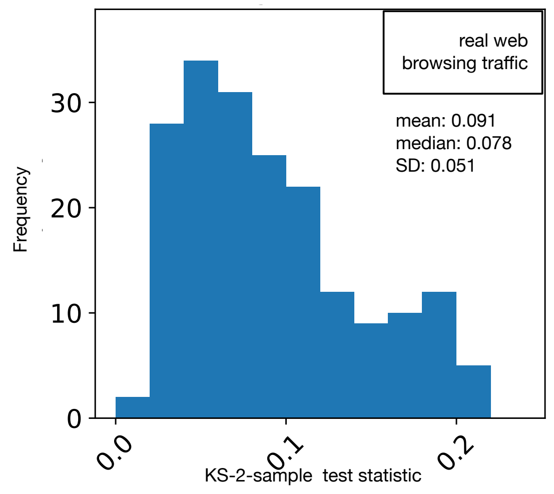
Fig. 5.7: Box plots, cumulative frequency curves and histograms for second order comparison of inter-PDU time between real web browsing traffic and `uhgeneric-v4` model generated web browsing traffic based on pairwise user-sessions KS-2-sample test statistics



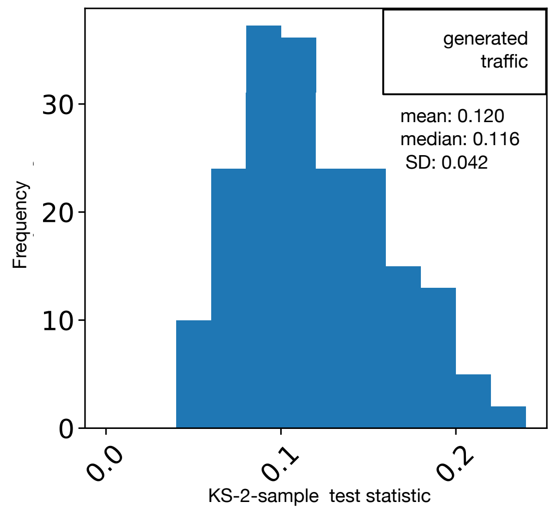
(a) Box plots comparing distributions of pairwise user-sessions KS-2-sample test statistics for the PDU size metric of real and generated web browsing traffic based on the `uhgeneric-v4` model



(b) Cumulative frequency curves comparing distributions of pairwise user-sessions KS-2-sample test statistics for the PDU size metric of real and generated web browsing traffic based on the `uhgeneric-v4` model



(c) Histogram of pairwise user-sessions KS-2-sample test statistics for the PDU size metric of real web browsing traffic



(d) Histogram of pairwise user-sessions KS-2-sample test statistics for the PDU size time metric of generated web browsing traffic based on the `uhgeneric-v4` model

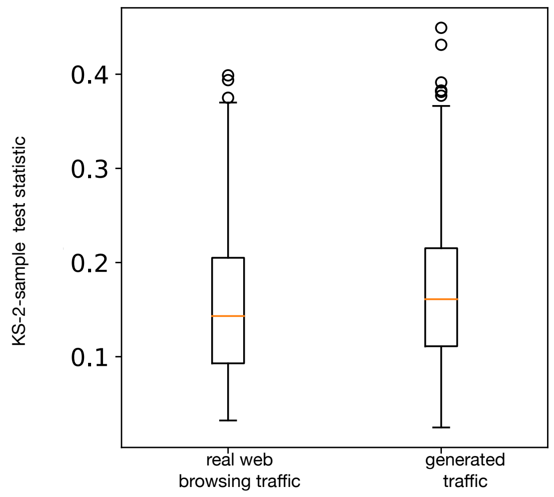
Fig. 5.8: Box plots, cumulative frequency curves and histograms for second order comparison of PDU size between real web browsing traffic and `uhgeneric-v4` model generated web browsing traffic based on pairwise user-sessions KS-2-sample test statistics

The box plots in Fig. 5.9a show that the generated traffic PDU rate's pairwise user-sessions KS-2-sample test statistics also has the identical median and interquartile ranges as in the real web browsing traffic. The cumulative frequency curves plotted in Fig. 5.9b also show that the distributions of the test statistics for PDU rates are comparable. In the corresponding histograms (Figs. 5.9c and 5.9d), the mean, median, and standard deviation values of 0.174, 0.161 and 0.080 in the generated traffic are also quite close to the values of 0.159, 0.143, and 0.086 seen for the real traffic.

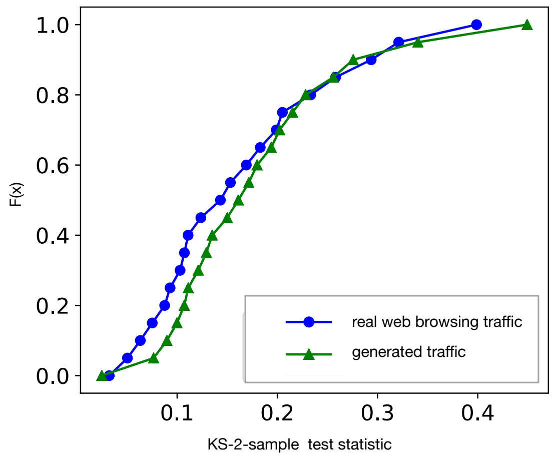
Fig. 5.10a also reveals quite different interquartile ranges for pairwise user-sessions KS-2-sample test statistics of the PDU throughput metric between the real and generated traffic. The cumulative frequency curves plotted (Fig. 5.10b) also have differing shapes. The histograms (Figs. 5.10c and 5.10d) show that mean, median and standard deviation values across both real web browsing traffic and the generated traffic also quite different. Our analysis reveals that this is due to the large order of values (in the 10^6 range) observed for throughput. When distributions with large values are being compared the KS-2-sample tests often yield results that show significant differences.

The box plots for the server response times pairwise user-sessions KS-2-sample test statistics in Fig. 5.11a show that the generated traffic has median and interquartile ranges that are marginally close to those of the real web browsing traffic. The cumulative frequency curves plotted in Fig. 5.11b also show the same marginal similarity between the distributions of KS-2-sample statistics for the server response time metric of the generated traffic and the real web browsing traffic. In the corresponding histograms (Figs. 5.11c and 5.11d), the mean, median, and standard deviation values of 0.169, 0.165 and 0.054 in the generated traffic are respectively in the same order asthe values of 0.205, 0.189, and 0.075 seen for the real traffic.

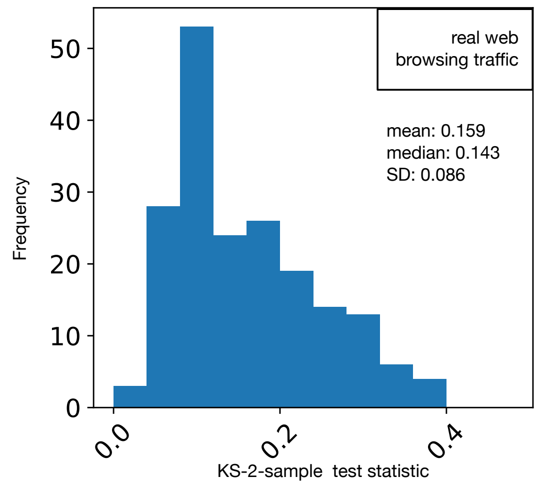
The close match between the distributions of pairwise user-sessions KS-2-sample test statistics of



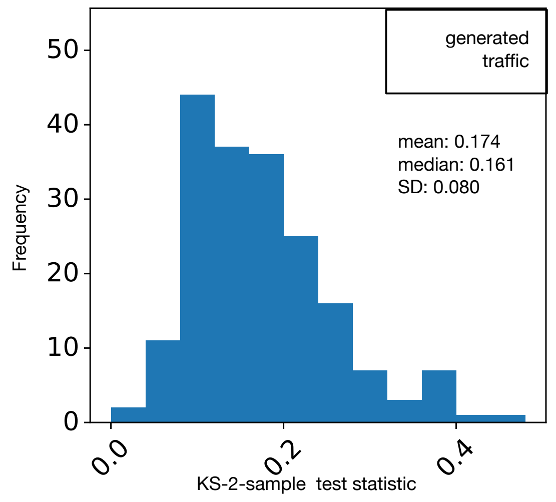
(a) Box plots comparing distributions of pairwise user-sessions KS-2-sample test statistics for the PDU rate metric of real and generated web browsing traffic based on the `uhgeneric-v4` model



(b) Box plots comparing distributions of pairwise user-sessions KS-2-sample test statistics for the PDU rate metric of real and generated web browsing traffic based on the `uhgeneric-v4` model

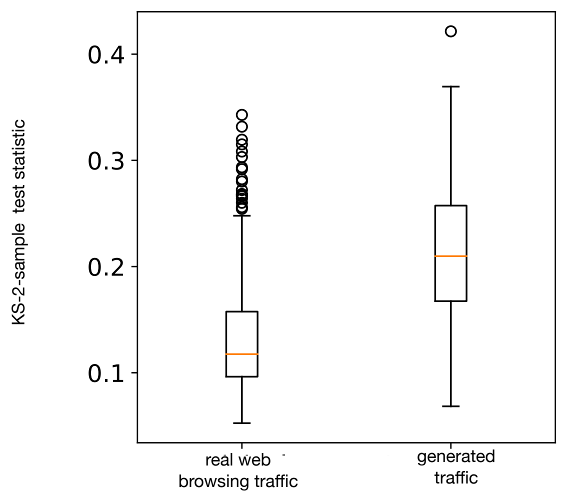


(c) Histogram of pairwise user-sessions KS-2-sample test statistics for the PDU rate metric of real web browsing traffic

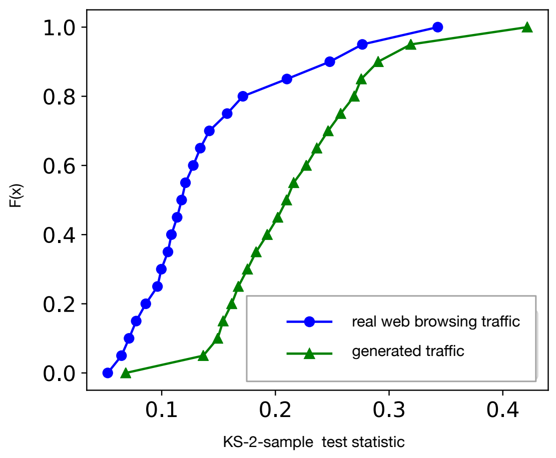


(d) Histogram of pairwise user-sessions KS-2-sample test statistics for the PDU rate metric of generated web browsing traffic based on the `uhgeneric-v4` model

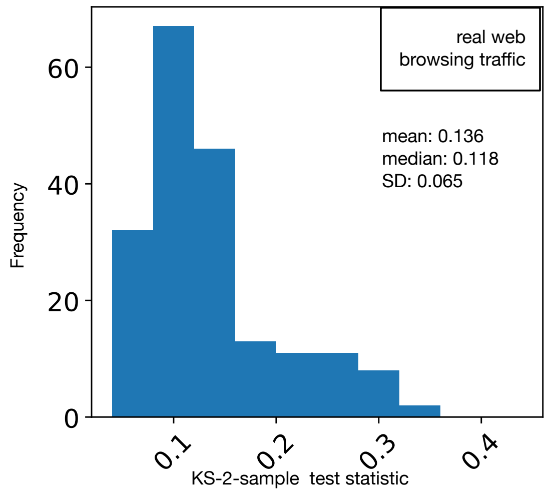
Fig. 5.9: Box plots, cumulative frequency curves and histograms for second order comparison of PDU rate between real web browsing traffic and `uhgeneric-v4` model generated web browsing traffic based on pairwise user-sessions KS-2-sample test statistics



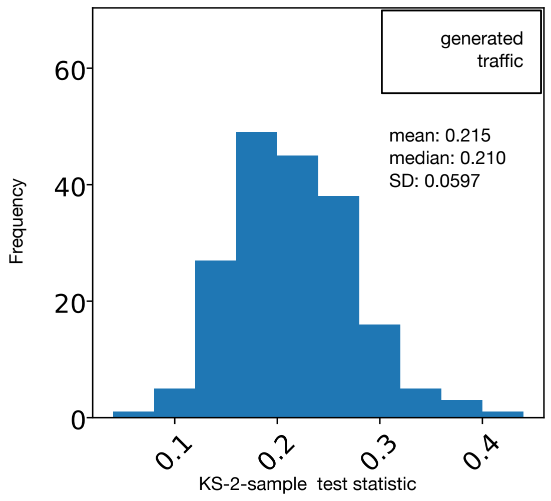
(a) Box plots comparing distributions of pairwise user-sessions KS-2-sample test statistics for the PDU throughput metric of real and generated web browsing traffic based on the `uhgeneric-v4` model



(b) Cumulative frequency curves comparing distributions of pairwise user-sessions KS-2-sample test statistics for the PDU throughput metric of real and generated web browsing traffic based on the `uhgeneric-v4` model

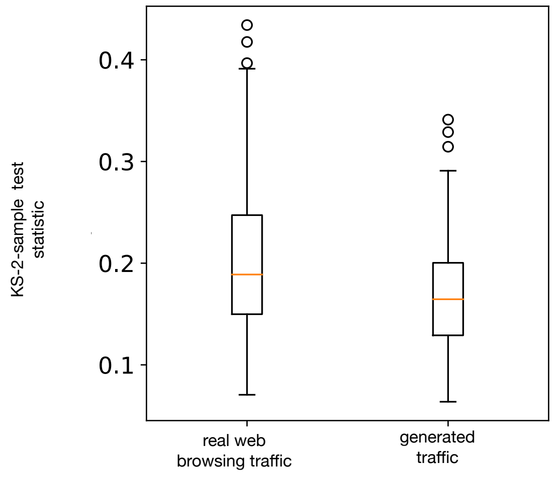


(c) Histogram of pairwise user-sessions KS-2-sample test statistics for the PDU throughput metric of real web browsing traffic

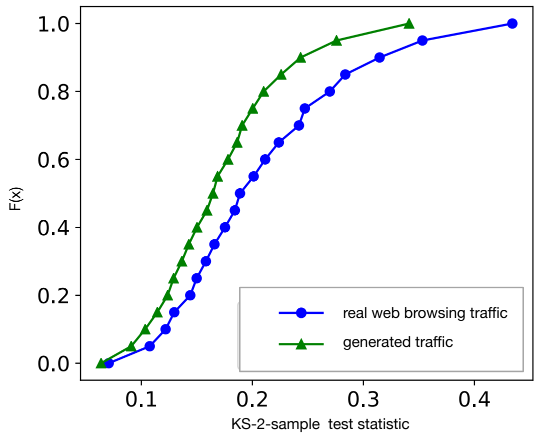


(d) Histogram of pairwise user-sessions KS-2-sample test statistics for the PDU throughput metric of generated web browsing traffic based on the `uhgeneric-v4` model

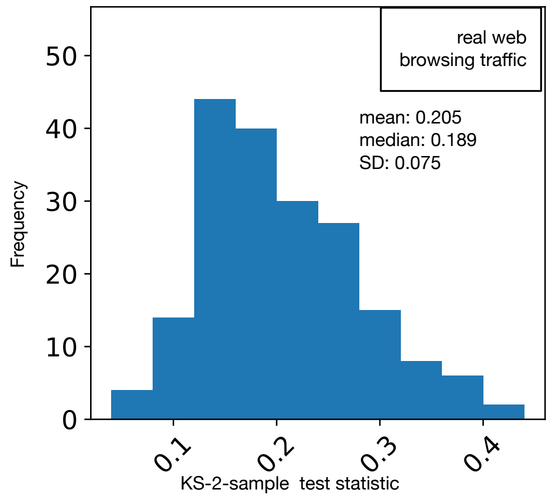
Fig. 5.10: Box plots, cumulative frequency curves and histograms for second order comparison of PDU throughput between real web browsing traffic and `uhgeneric-v4` model generated web browsing traffic based on pairwise user-sessions KS-2-sample test statistics



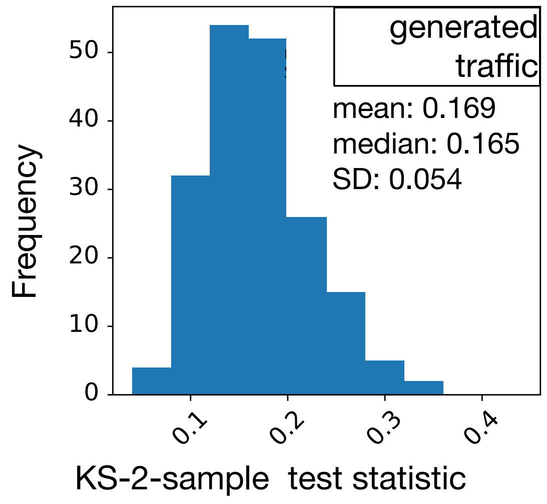
(a) Box plots comparing distributions of pairwise user-sessions KS-2-sample test statistics for the server response time metric of real and generated web browsing traffic based on the `uhgeneric-v4` model



(b) cumulative frequency curves comparing distributions of pairwise user-sessions KS-2-sample test statistics for the server response time metric of real and generated web browsing traffic based on the `uhgeneric-v4` model



(c) Histogram of pairwise user-sessions KS-2-sample test statistics for the server response time metric of real web browsing traffic



(d) Histogram of pairwise user-sessions KS-2-sample test statistics for the server response time metric of generated web browsing traffic based on the `uhgeneric-v4` model

Fig. 5.11: Box plots, cumulative frequency curves and histograms for second order comparison of server response time between real web browsing traffic and `uhgeneric-v4` model generated web browsing traffic based on pairwise user-sessions KS-2-sample test statistics

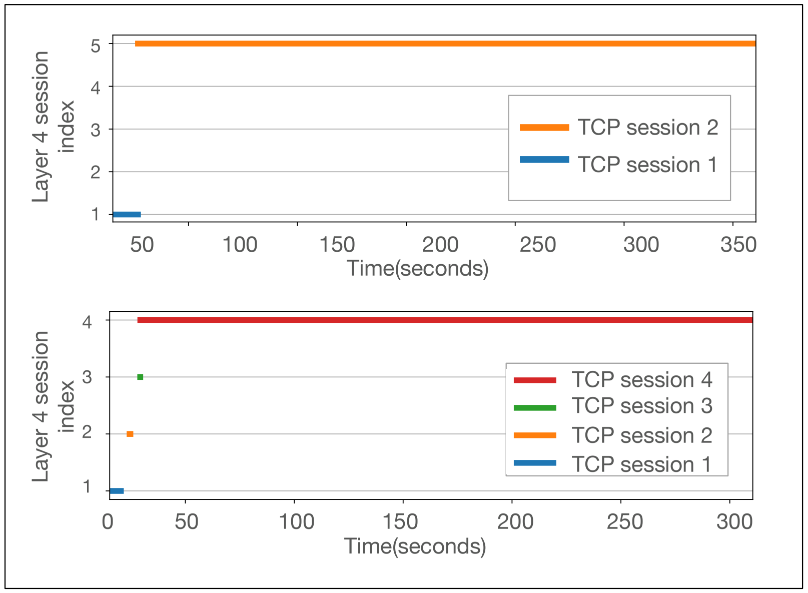
these metrics (especially PDU rates) discussed above, for generated traffic and the real web browsing traffic, indicates that the `uhgeneric-v4` model is very effective in modeling and generating realistic traffic for HTTP-based web applications. Our experiments with RDP, SSH, and video streaming traffic also obtained analogous results between real application

5.1.3.3 TCP and UDP Sessions in Typical Application User-Sessions

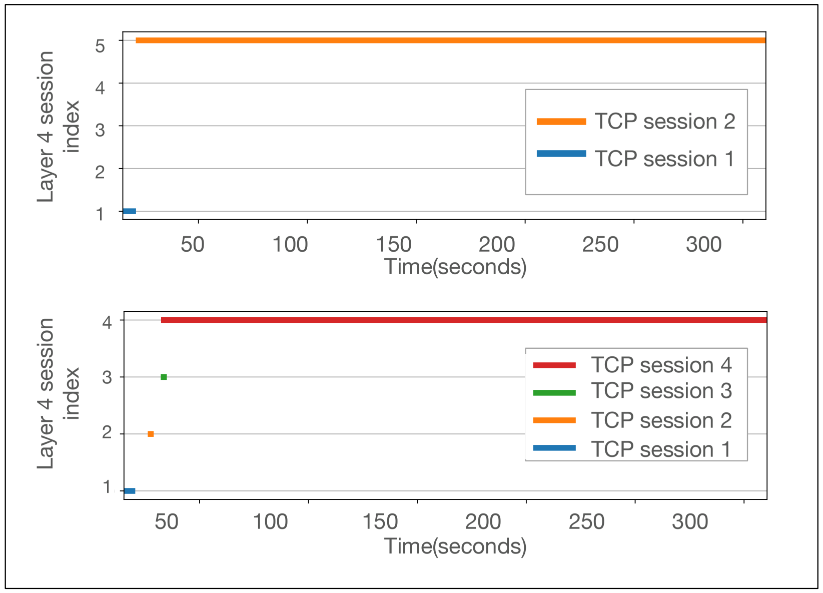
An effective realistic traffic modeling and generation algorithm should produce traffic that is similar to real application traffic in terms of the number and diversity of connections in a user-session. To evaluate this, we plotted the start times and duration for each transport layer connection used within each user-session. Fig. 5.12 presents a plot of 2 typical user-sessions each for both the real RDP traffic and the generated traffic. The plots show matching profiles across both the real and the generated traffic, indicating that the generated traffic contains transport layer connections with start times and durations that are realistic for the RDP client application based on comparison with the real traffic.

In Fig. 5.12 (a), the first TCP sessions for both real and generated traffic are short, representing connections used for authentication at the start of typical RDP user-sessions. While the second connections are longer representing data-exchange connections in the RDP user-session. Furthermore, in Fig. 5.12 (b), the first three TCP sessions for both real and generated traffic are short, while the fourth connections in both are much longer. We also obtained user-sessions with identical profiles between real and generated traffic in our web browser, SSH, and video streaming traffic generation experiments.

The evaluation outcomes discussed above gives us a good level confidence that the `uhgeneric-v4` modeling and generation algorithm performs well when used to generate realistic traffic for arbitrary individual network applications.



(a) Typical TCP sessions in real RDP traffic



(b) Typical TCP sessions in traffic generated by the uhgeneric-v4 RDP model

Fig. 5.12: L4 sessions duration in typical user-sessions of real and generated traffic for remote desktop application traffic

5.2 Modeling and Generating Multi-tier Application Traffic

In the modern internet, network traffic for an application typically consists of a mix of network traffic from many related network services and protocols. Many user applications utilize multiple network services running on different nodes in a network. This is common in many multi-tier and multi-service applications. This section focuses on examining how well the modeling and generation framework performs for one of such multi-tier applications.

5.2.1 Input Network Traffic Data

In the application, users interact with a Hypertext Transfer Protocol Secure (HTTPS) server, serving jupyter notebooks. The server, in this case, had multiple downstream connections with a group of Lightweight Directory Access Protocol (LDAP) servers, SSH servers, domain name service (DNS) servers, and servers running another proprietary service used in managing our testbed infrastructure (genilib). Each of the services are independent applications in their own right. However, in our deployment, they all produce network traffic only in response to user interaction with the jupyter notebook server.

We obtained input network traffic by capturing packets from the jupyter server, on a network with an architecture shown in Fig. 5.13. During the capture, members of our lab used various client devices to create notebooks on the jupyter server and executed commands that triggered communication between the jupyter server and the DNS, LDAP and genilb VTS servers. There were also several ssh connections from the jupyter server to virtual machines running within the VTS servers. We captured traffic on the jupyter server for 2.5 hours, while members within our lab environment carried out various tasks. This captured traffic was then used as input into our framework for traffic modeling and generation.

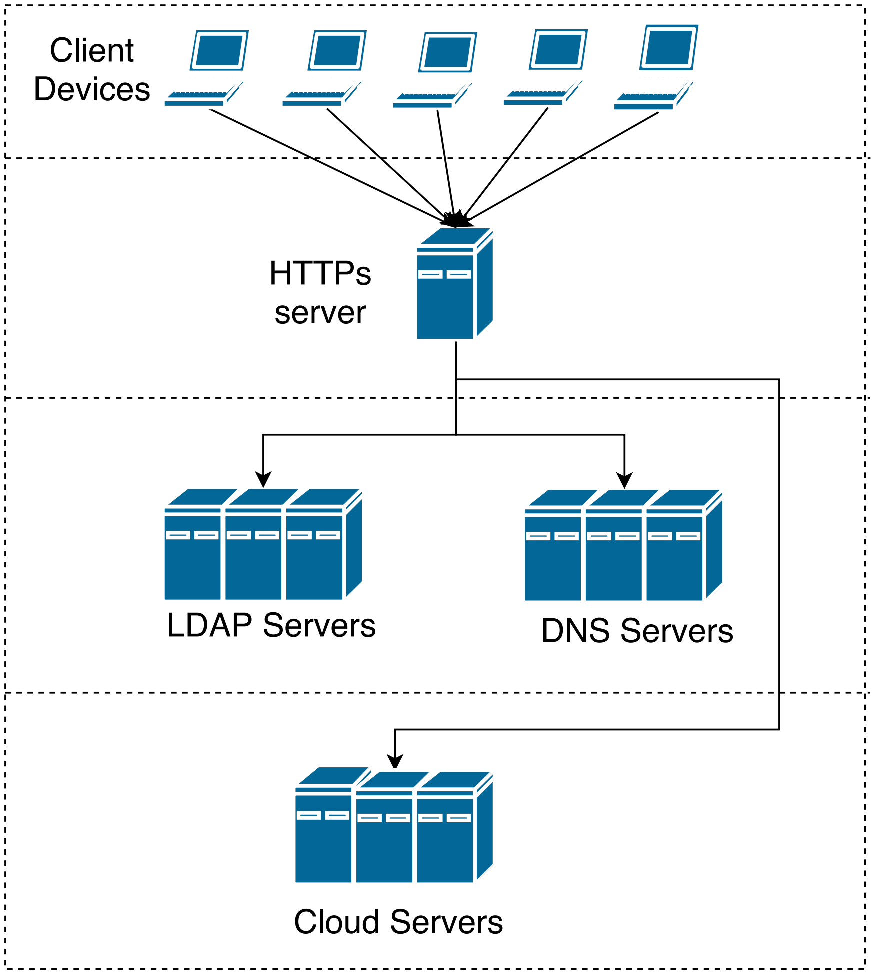


Fig. 5.13: Experiment network architecture for modeling and generation of a multi-tier network application traffic showing the various network services utilized in the multi-tier application software stack

5.2.2 Dataset Extraction, Traffic Modeling and Generation:

We applied the captured traffic trace file as input into the dataset extractor to yield a `csv` dataset file containing details of packet headers and connection details, as described in section 4.1. Initially, our dataset extraction process detected the various services in the multi-tier application, as expected. However, by making adjustments to the input metadata, the dataset extractor adjusted the dataset to recognize the capture as belonging to a multi-tier application, which had multiple connection classes belonging to each of the services included. The dataset was detected to have 26 user-sessions. This dataset was then applied to the modeling system and used to create a traffic model file based on the `uhgeneric-v4` model method. We created a test network having the same architecture as the network on which the input traffic was captured (Fig. 5.13). Using our repeatable experiment orchestration framework [48], we generated traffic on each host based on the traffic model, for twenty user-sessions of the application while we captured the generated traffic for evaluation.

5.2.3 Evaluation and Discussion

We evaluated the modeling and generation algorithm's performance in generating realistic traffic for multi-tier applications by comparing the generated traffic with the real traffic using our evaluation system (section 4.4). We calculate first and second-order statistics for each metric described in section 4.4.2. We also compare the arrival time and distributions of transport layer sessions, and we compare the percentage traffic volume composition for each network service present in the real and generated traffic.

Table 5.2: Evaluation results for modeling and generation of a multi-tier campus network application traffic

Metric	Median		Mean		Maximum		Standard Deviation		KS-2-Sample
	Real Traffic	Generated Traffic	Real Traffic	Generated Traffic	Real Traffic	Generated Traffic	Real Traffic	Generated Traffic	Test
Inter-PDU Time (seconds)	0.004	0.003	0.105	0.096	3932.161	1435.638	0.252	0.238	s=0.108
PDU Sizes (bytes)	325	413	1318	1390	383346	354956	2125	2201	s=0.045
PDU Rates (PDUs per second)	10	20	18	21	4130.0	2080.0	13	16	s=0.167
PDU Throughput (bps)	0.99e+4	1.38e4	4.25e+4	9.96e+4	1.23e+8	0.85e+8	1.02e+5	1.64e+5	s=0.154
Server response times (seconds)	0.004	0.002	0.014	0.012	3538.944	1272.321	0.029	0.026	s=0.076

5.2.3.1 First-Order Comparison of Metric Distributions

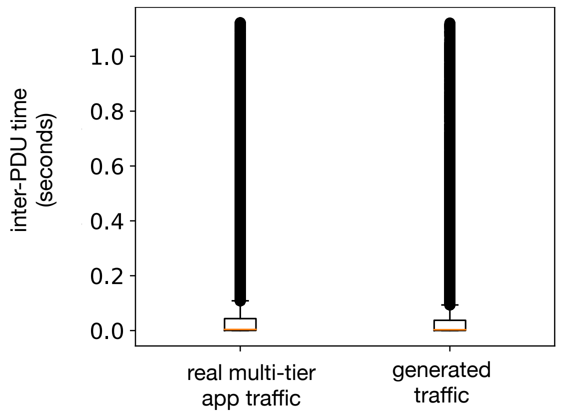
As described in section 4.4.3, we carried out first-order comparisons between real and generated traffic for each metric. We compared each metric's distribution based on statistical measures (including the median, mean, maximum, minimum, and standard deviations). For these evaluations with multi-tier application traffic, the tabulated side-by-side statistics for each metric of the real and generated traffic is given in Table 5.1.

In Table 5.2, the median values for each of the metrics are comparable. The median inter-PDU time of 0.004 seconds in the real trace is of the same order as the 0.003 seconds seen in the generated traces. Comparable values are also seen for the median values of the PDU size, PDU throughput, and server response time metrics. The mean, maximum, and standard deviation of each metric of generated traffic in the Table 5.2, have values that are different from those of real traffic but are quite comparable and still realistic for our multi-tier application traffic. For example, 0.096 seconds in the mean inter-PDU time of generated traffic is plausible for real traffic from our multi-service application, even though it is quite different from the value of 0.105 seconds seen in the real trace. The same applies to many of the other metrics' mean, maximum, and standard deviation values. The low KS-2-sample test statistic values (< 0.3) for each metric also indicate a close match between the empirical distributions of generated traffic and the real traffic.

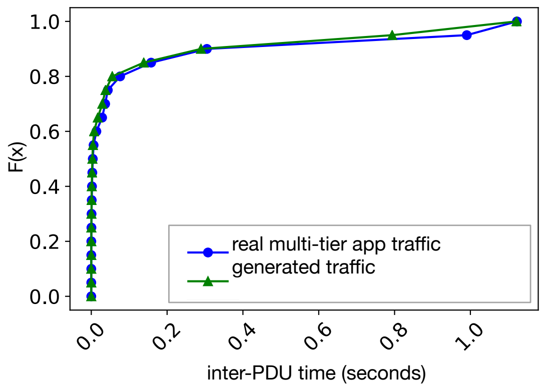
We went on to create the corresponding box plots, cumulative frequency curves, and histograms to compare each metrics' distribution for both real multi-tier application traffic and generated traffic based on our model. These plots, for the each metric, are given in Figs. 5.16a to 5.18d.

The box plots for the inter-PDU time in Fig. 5.14a show that the generated traffic inter-PDU time has the identical median and interquartile ranges as in the real multi-tier traffic. The box plots for both real and generated traffic also exhibit many similar outlier values. The cumulative frequency curves plotted in Fig. 5.14b are closely overlapping, thus indicating that the distribution of the inter-PDU times of the generated traffic is similar to that of the real multi-tier traffic. The histograms (Figs. 5.14c and 5.14d) show exactly the similar values of 0.105 seconds and 0.096 seconds in the mean, 0.004 seconds and 0.003 seconds in the median, and finally 0.025 seconds and 0.238 seconds in the standard deviation of inter-PDU times for both real m traffic and the generated traffic respectively.

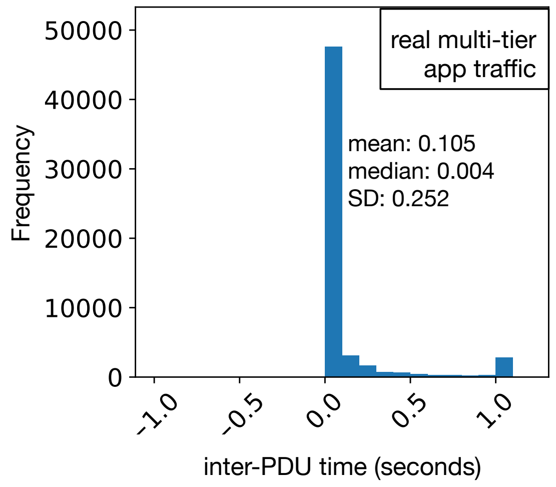
For the PDU size metric, the box plots in Fig. 5.15a show that the generated traffic also has similar median and interquartile ranges as in the real multi-tier traffic. The close overlap in the cumulative frequency curves plotted in Fig. 5.15b show that the entire distribution of the PDU size of the generated traffic also closely resembles that of the real multi-tier traffic. In the corresponding histograms (Figs. 5.15c and 5.15d), the mean values of 1318 bytes and 1390 bytes in real and generated traffic are very similar. The standard deviations values of 2201 bytes in the generated traffic histogram, is also similar to the value of 2125 bytes in the real multi-tier traffic.



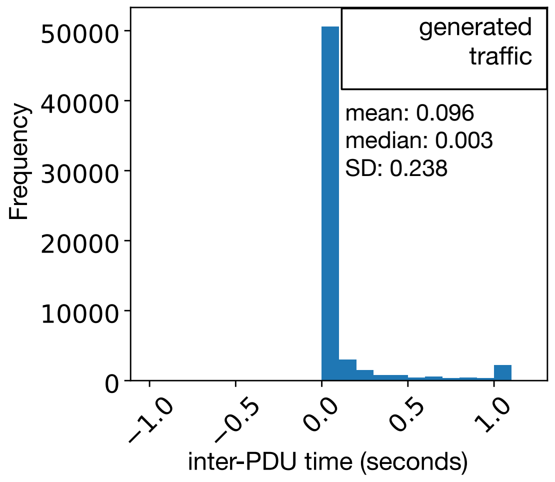
(a) Box plots comparing inter-PDU time distributions of real multi-tier application traffic and generated traffic based on the `uhgeneric-v4` modeling algorithm



(b) Cumulative frequency curves comparing inter-PDU time distributions of real multi-tier application traffic and generated traffic based on the `uhgeneric-v4` modeling algorithm

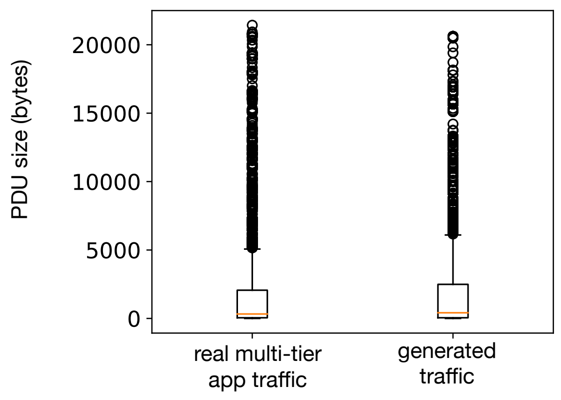


(c) Histogram of inter-PDU time distribution for real multi-tier application traffic

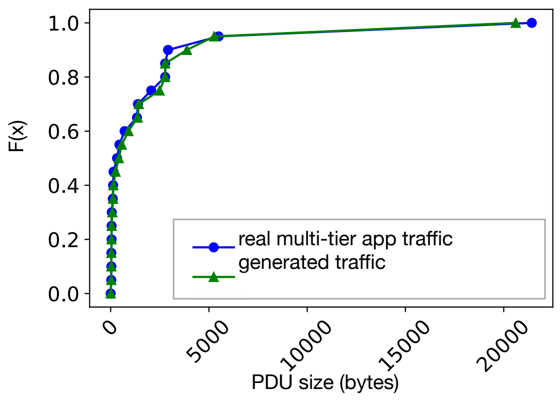


(d) Histogram of inter-PDU time distribution for generated multi-tier application traffic based on the `uhgeneric-v4` modeling algorithm

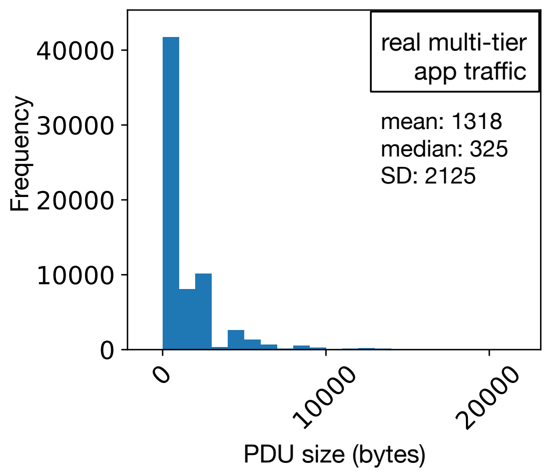
Fig. 5.14: Box plots, cumulative frequency curves and histograms comparing inter-PDU time distributions of real multi-tier application traffic with inter-PDU time distributions of generated multi-tier application traffic based on the `uhgeneric-v4` modeling algorithm



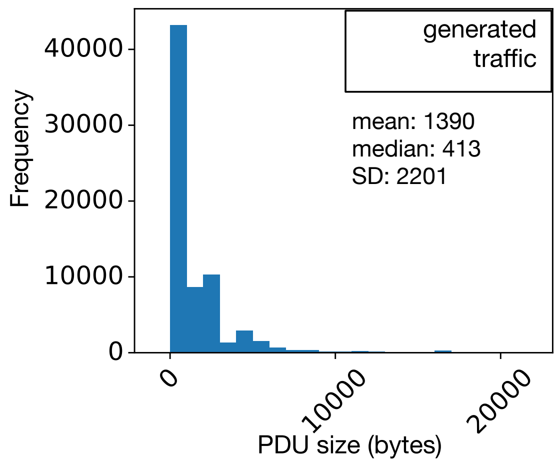
(a) Box plots comparing PDU size distributions of real multi-tier application traffic with PDU size distributions of generated multi-tier application traffic based on the `uhgeneric-v4` modeling algorithm



(b) Cumulative frequency curves comparing PDU size distributions of real multi-tier application traffic with PDU size distributions of generated multi-tier application traffic based on the `uhgeneric-v4` modeling algorithm



(c) Histogram of PDU size distribution for real multi-tier application traffic

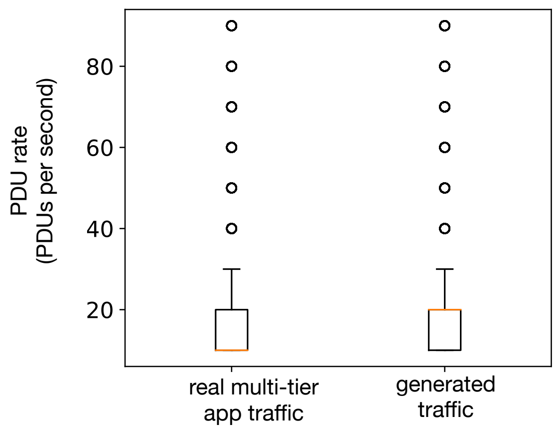


(d) Histogram of PDU size distribution for generated multi-tier application traffic based on the `uhgeneric-v4` modeling algorithm

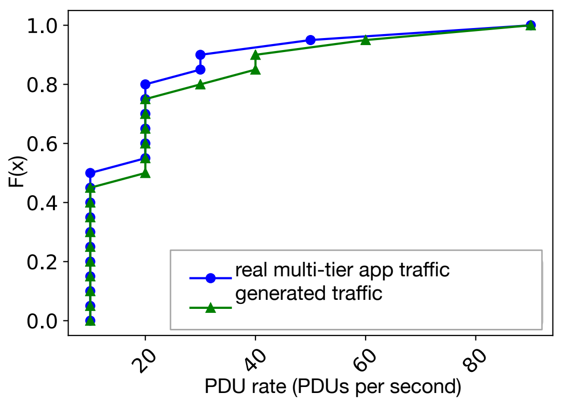
Fig. 5.15: Box plots, cumulative frequency curves and histograms comparing PDU size distributions of real multi-tier application traffic with PDU size distributions of generated multi-tier application traffic based on the `uhgeneric-v4` modeling algorithm

The box plots in Fig. 5.16a show that the generated traffic PDU rate has the same interquartile ranges as in the real multi-tier traffic. The box plots also indicate similar ranges of outlier values. The cumulative frequency curves plotted in Fig. 5.16b also show that the distributions of the PDU rates are similar, although they do not perfectly overlap. The histograms in Figs. 5.16c and 5.16d reveal very some differences in the PDU rate mean, median, and standard deviation values between real and generated traffic. The real multi-tier traffic has a mean of 18 PDUs per second, while the generated traffic has 21 PDUs per second. The median values are 10 PDUs per second and 20 PDUs per second respectively. However, a look at the shape of both histograms also show how similar the distributions are.

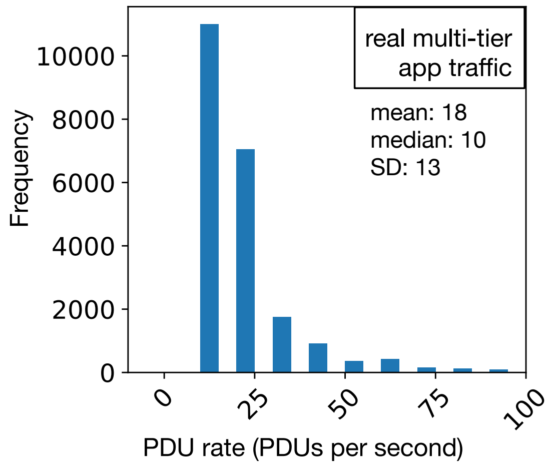
In Fig. 5.17b, the cumulative frequency curves plotted reveals that the distribution of the PDU throughput for the generated traffic marginally overlaps that of the real multi-tier traffic. The Histograms (Figs. 5.17c and 5.17d) show that mean, median and standard deviation values across both real multi-tier traffic and the generated traffic are always quite different but the generated values are close enough to pass for real multi-tier traffic. The shape of both histograms indicate very similar distributions between real and generated traffic.



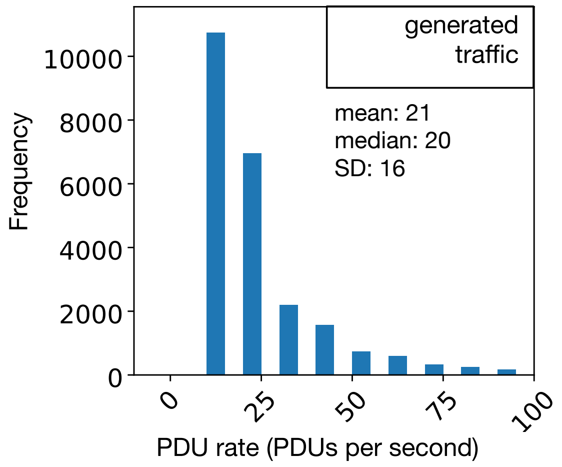
(a) Box plots comparing PDU rate distributions for real multi-tier application traffic and generated traffic based on the `uhgeneric-v4` modeling algorithm



(b) Cumulative frequency curves comparing PDU rate distributions for real multi-tier application traffic and generated traffic based on the `uhgeneric-v4` modeling algorithm

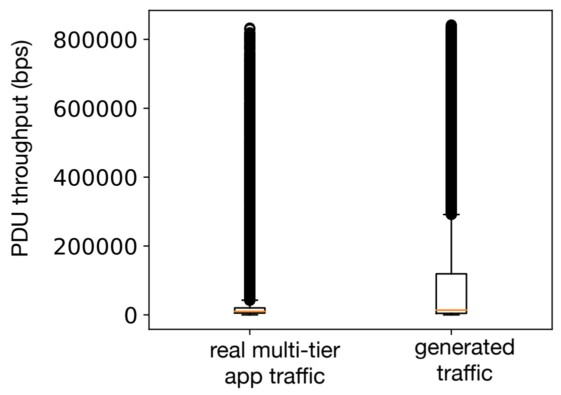


(c) Histogram of PDU rate distribution for real multi-tier application traffic

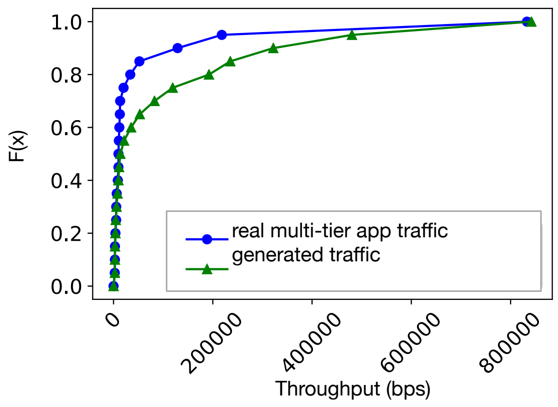


(d) Histogram of PDU rate distribution for generated multi-tier application traffic based on the `uhgeneric-v4` modeling algorithm

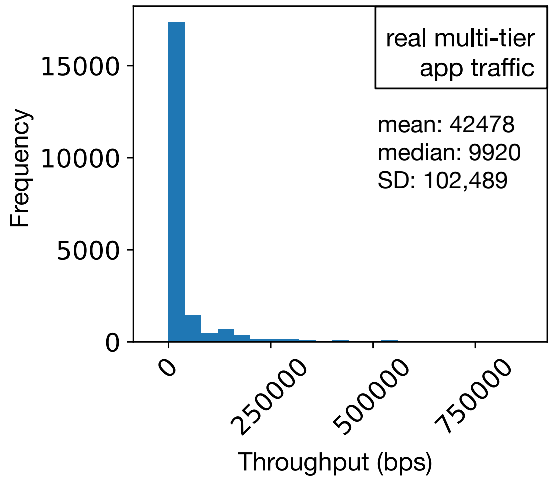
Fig. 5.16: Box plots, cumulative frequency curves and histograms comparing PDU rate distributions of real multi-tier application traffic with PDU rate distributions of generated multi-tier application traffic based on the `uhgeneric-v4` modeling algorithm



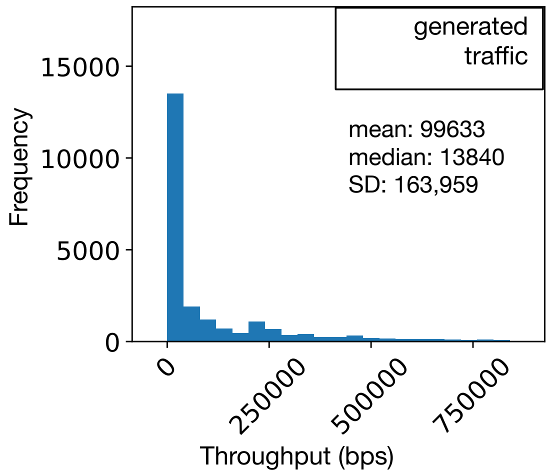
(a) Box plots comparing PDU throughput distributions of real multi-tier application traffic with PDU throughput distributions of generated multi-tier application traffic based on the `uhgeneric-v4` modeling algorithm



(b) Cumulative frequency curves comparing PDU throughput distributions of real multi-tier application traffic with PDU throughput distributions of generated multi-tier application traffic based on the `uhgeneric-v4` modeling algorithm



(c) Histogram of PDU throughput distribution for real multi-tier application traffic

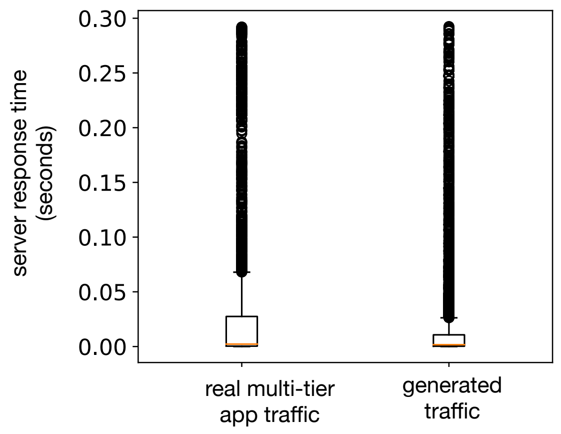


(d) Histogram of PDU throughput distribution for generated multi-tier application traffic based on the `uhgeneric-v4` modeling algorithm

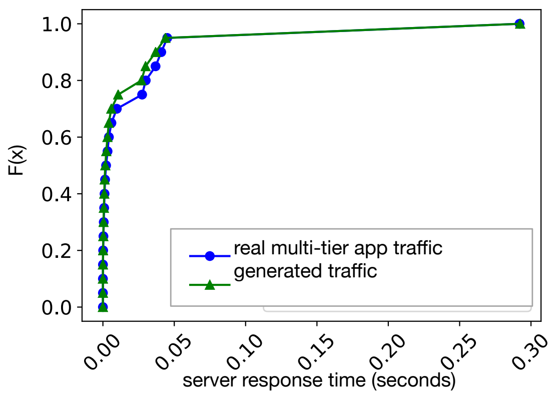
Fig. 5.17: Box plots, cumulative frequency curves and histograms comparing PDU throughput distributions of real multi-tier application traffic with PDU throughput distributions of generated multi-tier application traffic based on the `uhgeneric-v4` modeling algorithm

The cumulative frequency curves plotted in Fig. 5.18b show that the generated traffic's server response times distribution of the generated traffic closely overlaps that of the real multi-tier traffic. In addition, the histograms (Figs. 5.18c and 5.18d) show similar values of 0.014 seconds and 0.012 seconds in the mean, and exactly the same values of 0.002 seconds in the median. The values of standard deviation (0.029 and 0.026 seconds) are also very close for both real multi-tier traffic and the generated traffic.

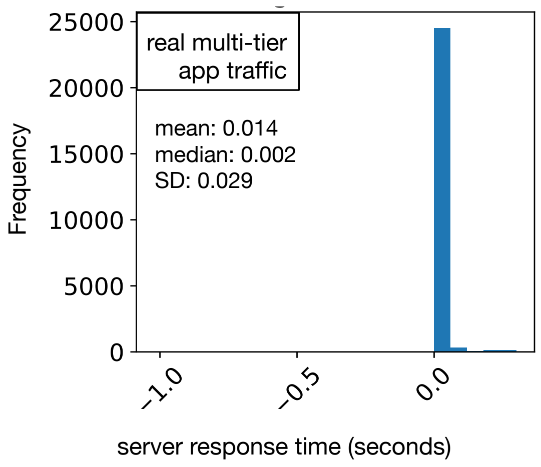
The close similarity between the distributions of these metrics (especially PDU rates, and server response times) discussed above, for generated traffic and the real multi-tier traffic, indicates that the `uhgeneric-v4` model is very effective in modeling and generating realistic traffic for multi-tier applications.



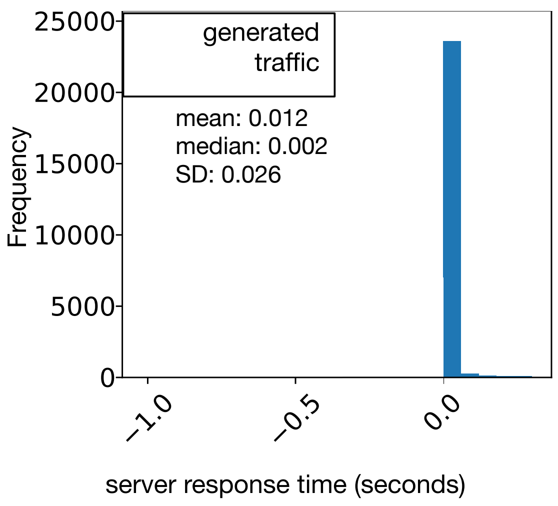
(a) Box plots comparing server response time distributions of real multi-tier application traffic with server response time distributions of generated multi-tier application traffic based on the `uhgeneric-v4` modeling algorithm



(b) Cumulative frequency curves comparing server response time distributions of real multi-tier application traffic with server response time distributions of generated multi-tier application traffic based on the `uhgeneric-v4` modeling algorithm



(c) Histogram of server response time distribution for real multi-tier application traffic



(d) Histogram of server response time distribution for generated multi-tier application traffic based on the `uhgeneric-v4` modeling algorithm

Fig. 5.18: Box plots, cumulative frequency curves and histograms comparing server response time distributions of real multi-tier application traffic with server response time distributions of generated multi-tier application traffic based on the `uhgeneric-v4` modeling algorithm

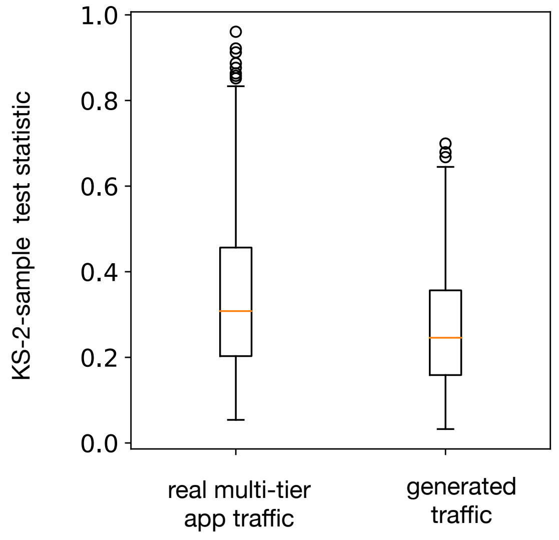
5.2.3.2 Second-Order Comparison of Metric Distributions

We carried out a second-order evaluation to compare each metric's variations for user-sessions of real and generated traffic, based on the process described in section 4.4.4. The results are presented in Figs. 5.19a to 5.23d.

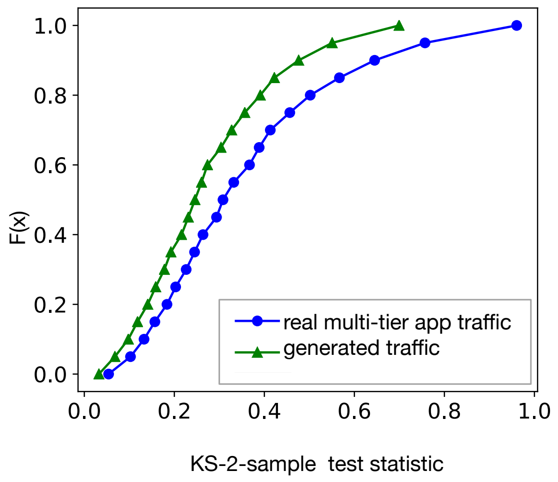
The box plots for the inter-PDU time in Fig. 5.19a show that the pairwise user-sessions KS-2-sample test statistics for the inter-PDU time in the generated traffic has median value that is close to that of real multi-tier application traffic. The cumulative frequency curves plotted Fig. 5.19b also show that the distribution of the inter-PDU times KS-2-sample test statistics of the generated traffic is marginally close to that of the real multi-tier application traffic. The mean, median, and standard deviation values of 0.268s, 0.246, and 0.144 seen in the inter-PDU time histograms of the generated traffic are different but still quite close to the values of 0.353, 0.308, and 0.198 seen in the real traffic's inter-PDU time histograms(Figs. 5.19c and 5.19d).

For the PDU size metric, the box plots in Fig. 5.20a show that the pairwise user-sessions KS-2-sample test statistics for the generated traffic PDU size metric has a similar range of values as that of the real multi-tier application traffic. In the corresponding histograms (Figs. 5.20c and 5.20d), the mean, median, and standard deviation values of 0.340, 0.319 and 0.172 in the generated traffic are not the same as the values of 0.449, 0.430 and 0.180 seen for the real traffic. However the values in the generated traces are close enough to pass for generated realistic multi-tier application traffic.

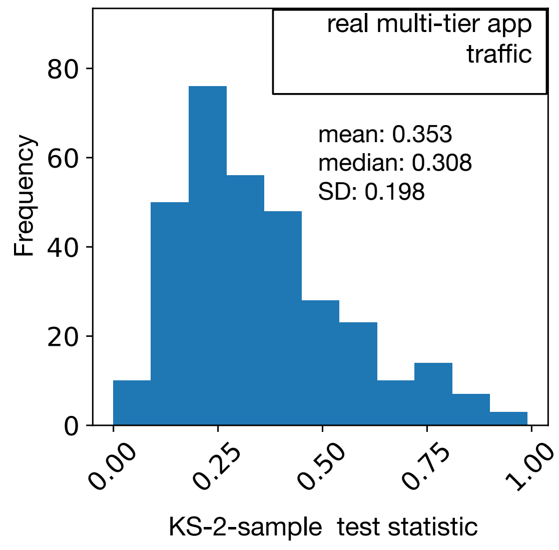
The box plots in Fig. 5.21a show that the generated traffic PDU rate's pairwise user-sessions KS-2-sample test statistics also has similar ranges as in the real multi-tier application traffic. The cumulative frequency curves plotted Fig. 5.21b also show that the distributions of the test statistics for PDU rates are only marginally similar. In the corresponding histograms (Figs. 5.21c and 5.21d), the mean, median, and standard deviation values of 0.302, 0.227 and 0.235 in the generated traffic



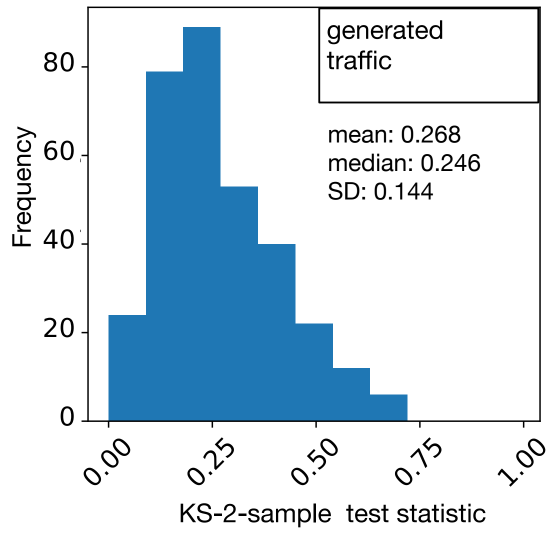
(a) Box plots comparing distributions of pairwise user-sessions KS-2-sample test statistics for the inter-PDU time metric of real and generated multi-tier application traffic based on the `uhgeneric-v4` model



(b) Cumulative frequency curves comparing distributions of pairwise user-sessions KS-2-sample test statistics for the inter-PDU time metric of real and generated multi-tier application traffic based on the `uhgeneric-v4` model

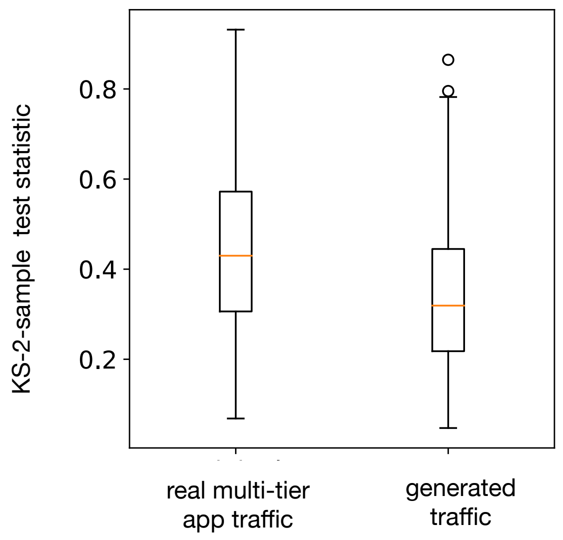


(c) Histogram of pairwise user-sessions KS-2-sample test statistics for the inter-PDU time metric of real multi-tier application traffic

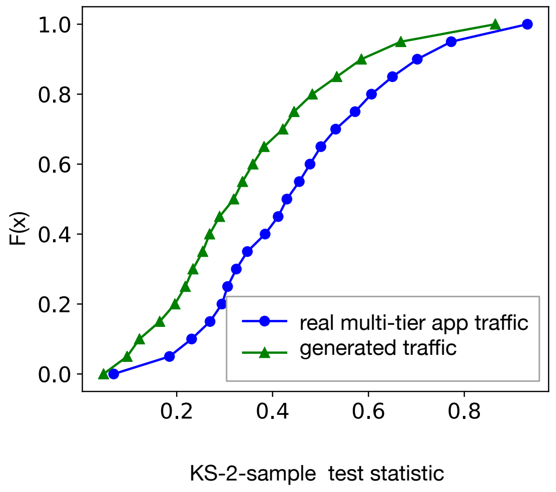


(d) Histogram of pairwise user-sessions KS-2-sample test statistics for the inter-PDU time metric of generated multi-tier application traffic based on the `uhgeneric-v4` model

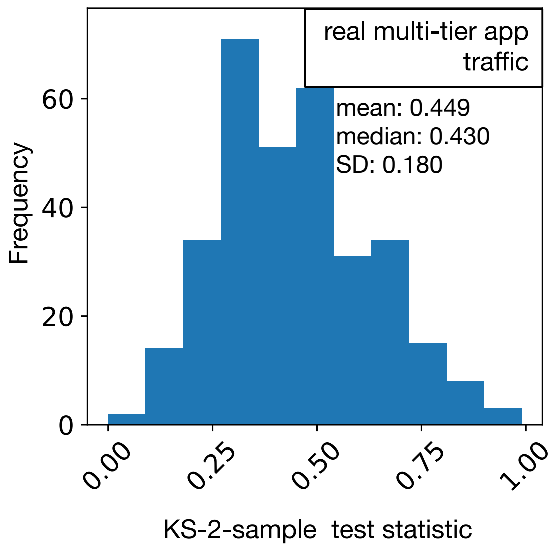
Fig. 5.19: Box plots, cumulative frequency curves and histograms for second order comparison of inter-PDU time between real multi-tier application traffic and `uhgeneric-v4` model generated multi-tier application traffic based on pairwise user-sessions KS-2-sample test statistics



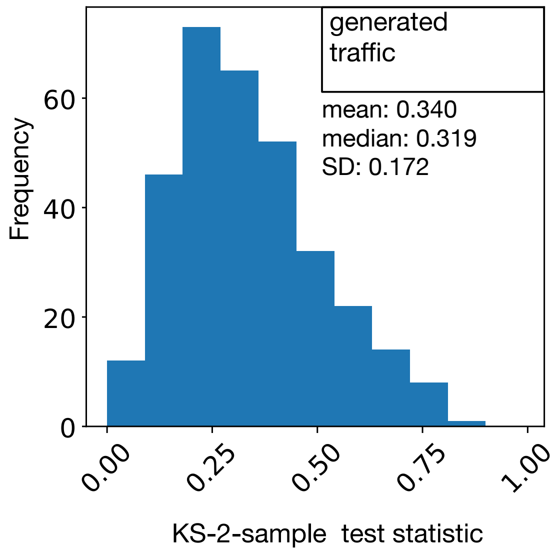
(a) Box plots comparing distributions of pairwise user-sessions KS-2-sample test statistics for the PDU size metric of real and generated multi-tier application traffic based on the `uhgeneric-v4` model



(b) Cumulative frequency curves comparing distributions of pairwise user-sessions KS-2-sample test statistics for the PDU size metric of real and generated multi-tier application traffic based on the `uhgeneric-v4` model



(c) Histogram of pairwise user-sessions KS-2-sample test statistics for the PDU size metric of real multi-tier application traffic



(d) Histogram of pairwise user-sessions KS-2-sample test statistics for the PDU size time metric of generated multi-tier application traffic based on the `uhgeneric-v4` model

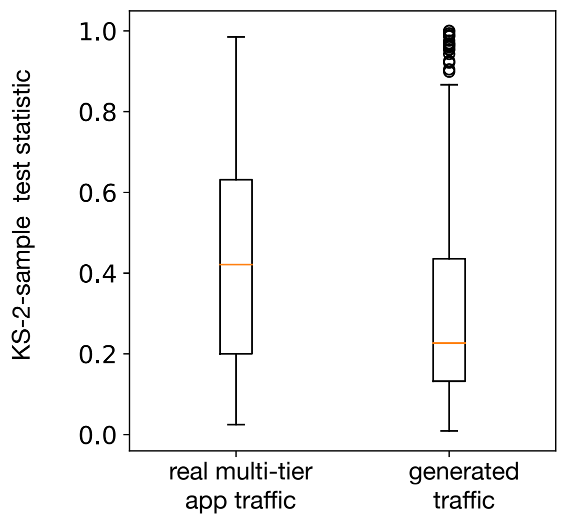
Fig. 5.20: Box plots, cumulative frequency curves and histograms for second order comparison of PDU size between real multi-tier application traffic and `uhgeneric-v4` model generated multi-tier application traffic based on pairwise user-sessions KS-2-sample test statistics

are not the same as the values of 0.413, 0.421 and 0.247 seen for the real traffic. However the values in the generated traces are close enough to pass for generated realistic multi-tier application traffic.

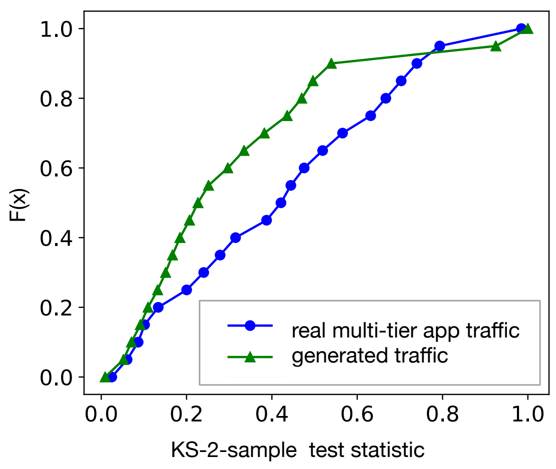
Fig. 5.22a also reveals quite different interquartile ranges for pairwise user-sessions KS-2-sample test statistics of the PDU throughput metric between the real and generated traffic. The cumulative frequency curves plotted (Fig. 5.22b) are also only marginally similar. The histograms (Figs. 5.22c and 5.22d) show that mean, median and standard deviation values across both real multi-tier application traffic and the generated traffic also quite different due to the large order of values (in the 10^6 range) observed in first order throughput values. When distributions with large values are being compared the KS-2-sample tests often yield results that show significant differences.

The box plots for the server response times pairwise user-sessions KS-2-sample test statistics in 5.23a show that the generated traffic has median and interquartile ranges that are also marginally close to those of the real multi-tier application traffic. The cumulative frequency curves plotted in Fig. 5.23b also show the same marginal similarity between the distributions of KS-2-sample statistics for the server response time metric of the generated traffic and the real multi-tier application traffic. In the corresponding histograms (Figs. 5.23c and 5.23d), the mean, median, and standard deviation values of 0.305, 0.272 and 0.152 in the generated traffic are not the same as the values of 0.477, 0.454 and 0.215 seen for the real traffic. However the values in the generated traces are close enough to pass for generated realistic multi-tier application traffic.

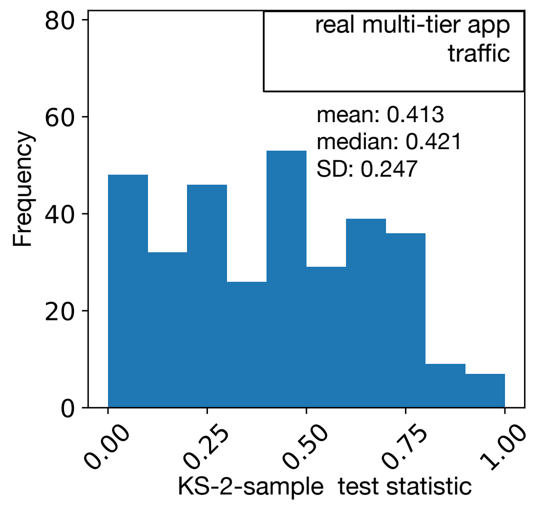
This similarity between the distributions of pairwise user-sessions KS-2-sample test statistics of these metrics (especially PDU rates) discussed above, for generated traffic and the real multi-tier application traffic, indicates that the `uhgeneric-v4` model performs adequately in modeling and generating realistic traffic for real multi-tier applications.



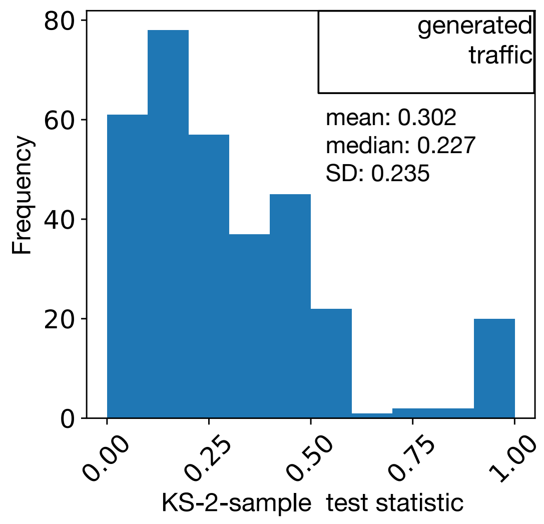
(a) Box plots comparing distributions of pairwise user-sessions KS-2-sample test statistics for the PDU rate metric of real and generated multi-tier application traffic based on the `uhgeneric-v4` model



(b) Box plots comparing distributions of pairwise user-sessions KS-2-sample test statistics for the PDU rate metric of real and generated multi-tier application traffic based on the `uhgeneric-v4` model

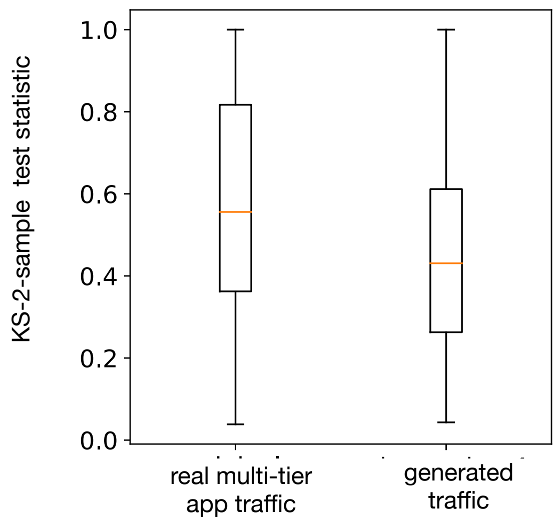


(c) Histogram of pairwise user-sessions KS-2-sample test statistics for the PDU rate metric of real multi-tier application traffic

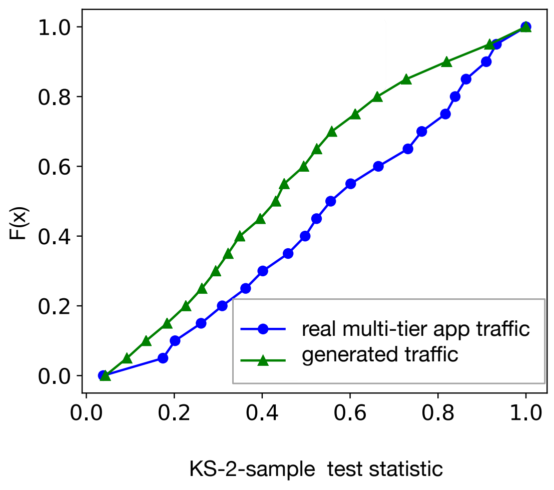


(d) Histogram of pairwise user-sessions KS-2-sample test statistics for the PDU rate metric of generated multi-tier application traffic based on the `uhgeneric-v4` model

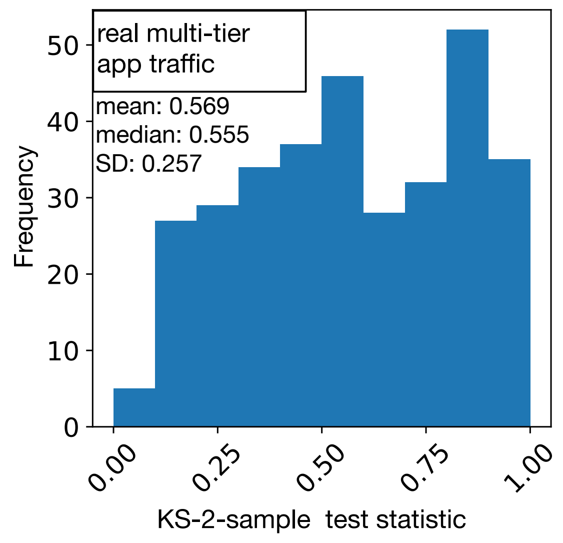
Fig. 5.21: Box plots, cumulative frequency curves and histograms for second order comparison of PDU rate between real multi-tier application traffic and `uhgeneric-v4` model generated multi-tier application traffic based on pairwise user-sessions KS-2-sample test statistics



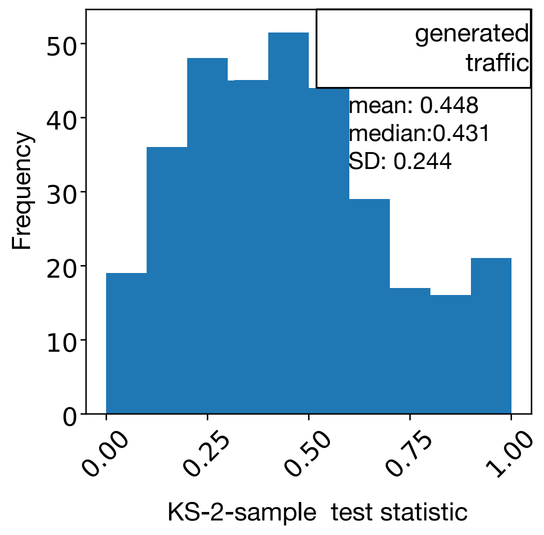
(a) Box plots comparing distributions of pairwise user-sessions KS-2-sample test statistics for the PDU throughput metric of real and generated multi-tier application traffic based on the `uhgeneric-v4` model



(b) Cumulative frequency curves comparing distributions of pairwise user-sessions KS-2-sample test statistics for the PDU throughput metric of real and generated multi-tier application traffic based on the `uhgeneric-v4` model

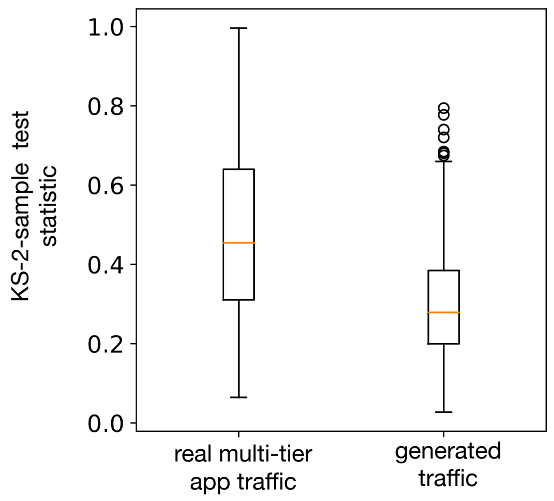


(c) Histogram of pairwise user-sessions KS-2-sample test statistics for the PDU throughput metric of real multi-tier application traffic

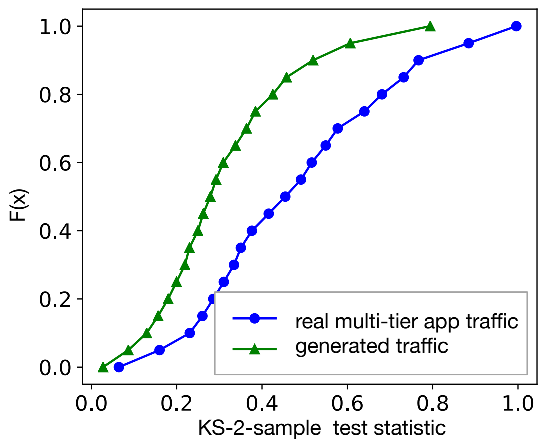


(d) Histogram of pairwise user-sessions KS-2-sample test statistics for the PDU throughput metric of generated multi-tier application traffic based on the `uhgeneric-v4` model

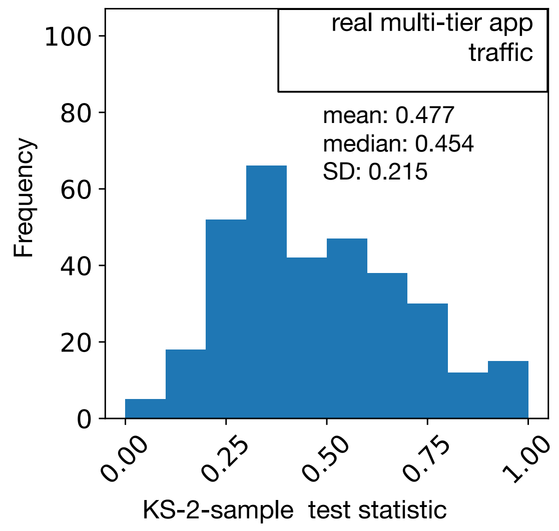
Fig. 5.22: Box plots, cumulative frequency curves and histograms for second order comparison of PDU throughput between real multi-tier application traffic and `uhgeneric-v4` model generated multi-tier application traffic based on pairwise user-sessions KS-2-sample test statistics



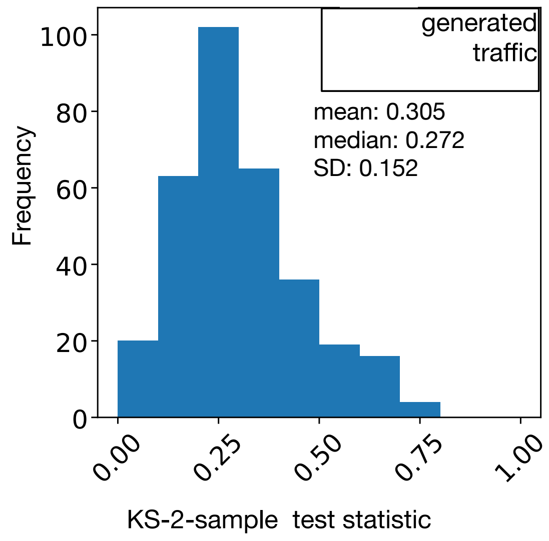
(a) Box plots comparing distributions of pairwise user-sessions KS-2-sample test statistics for the server response time metric of real and generated multi-tier application traffic based on the `uhgeneric-v4` model



(b) cumulative frequency curves comparing distributions of pairwise user-sessions KS-2-sample test statistics for the server response time metric of real and generated multi-tier application traffic based on the `uhgeneric-v4` model



(c) Histogram of pairwise user-sessions KS-2-sample test statistics for the server response time metric of real multi-tier application traffic

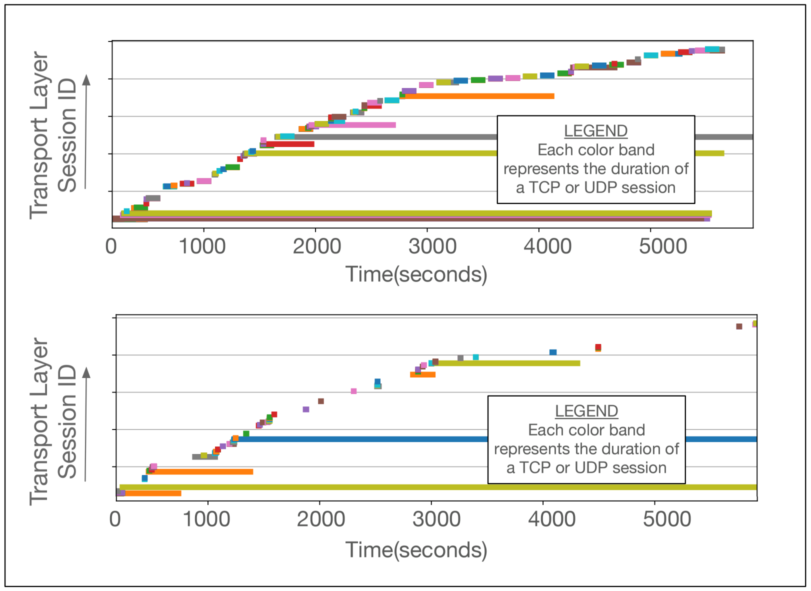


(d) Histogram of pairwise user-sessions KS-2-sample test statistics for the server response time metric of generated multi-tier application traffic based on the `uhgeneric-v4` model

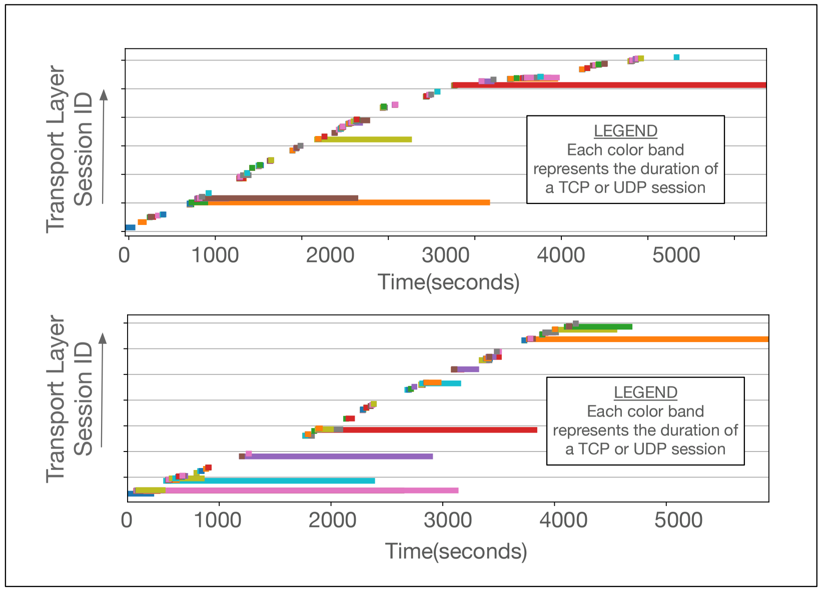
Fig. 5.23: Box plots, cumulative frequency curves and histograms for second order comparison of server response time between real multi-tier application traffic and `uhgeneric-v4` model generated multi-tier application traffic based on pairwise user-sessions KS-2-sample test statistics

5.2.3.3 TCP and UDP Sessions in Typical Application User-Sessions

To evaluate how the modeling and generation system performs in terms of the number and kind of connections used in traffic generation for each user-session, we plotted the start times and duration for each transport layer connection used within each user-session. In Fig. 5.24, we provide a plot of 2 typical user-sessions each for both the original real traffic and the generated traffic. These plots of L4 sessions in have similar profiles across both the real and the generated traffic, indicating that the generated traffic contains a number of connections and with start times and durations that are realistic for the web browser application based comparison with the real traffic.



(a) Typical TCP sessions in real multi-tier application traffic



(b) Typical TCP sessions in traffic generated by the uhgeneric-v4 model

Fig. 5.24: L4 sessions durations in typical user-sessions of real and generated traffic for multi-tier application

5.2.3.4 Multi-tier Application Composition

When generating realistic traffic for application with multiple services, it is important that the distribution of network traffic volume across services in the generated traffic is similar to the real application traffic. Hence, we evaluate our model's effectiveness by comparing the charts in Fig. 5.25 and Fig. 5.26, which show the distribution PDUs and bytes from each of the component services in the real and generated traces. For example in Fig. 5.25, HTTPS and SSH traffic dominate the total PDU counts metric in both real and generated traffic, with HTTPS and SSH having 75% and 15% respectively in the real traffic, and 71% and 17% respectively in the generated traffic. However, in 5.26, HTTPS and genilib traffic dominate the byte counts in both real and generated traffic, with HTTPS and SSH having 66% and 29% respectively in the real traffic, and 69% and 25% respectively in the generated traffic. Thus the results show good similarity between the distributions for both the PDU counts and the byte counts.

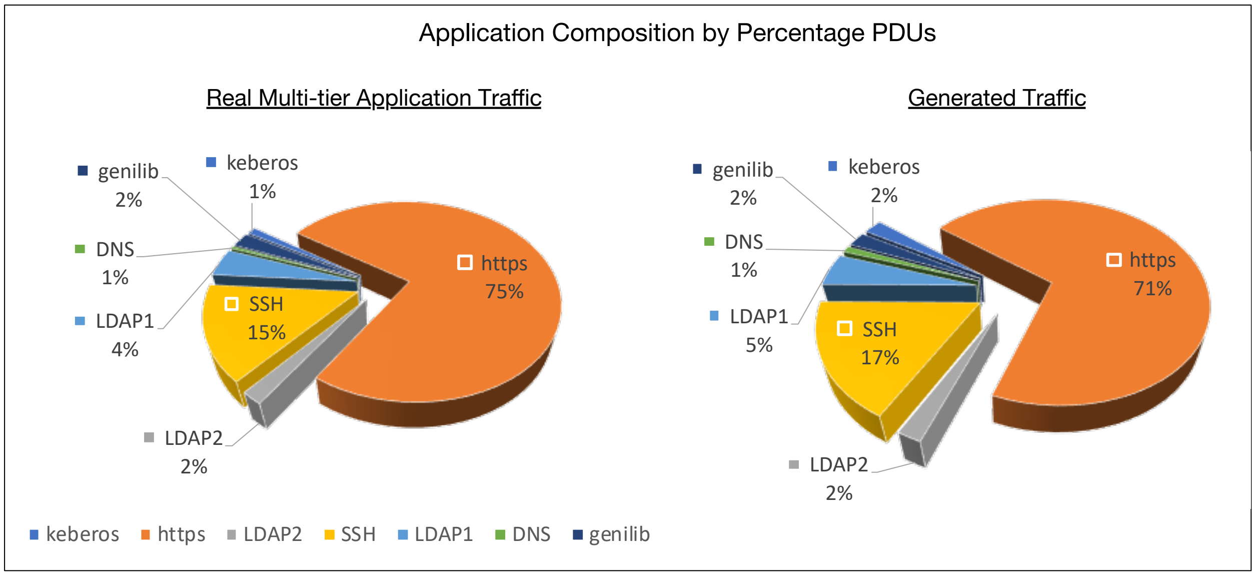


Fig. 5.25: PDU counts percent by application for modeling and generation of a multi-tier network application traffic

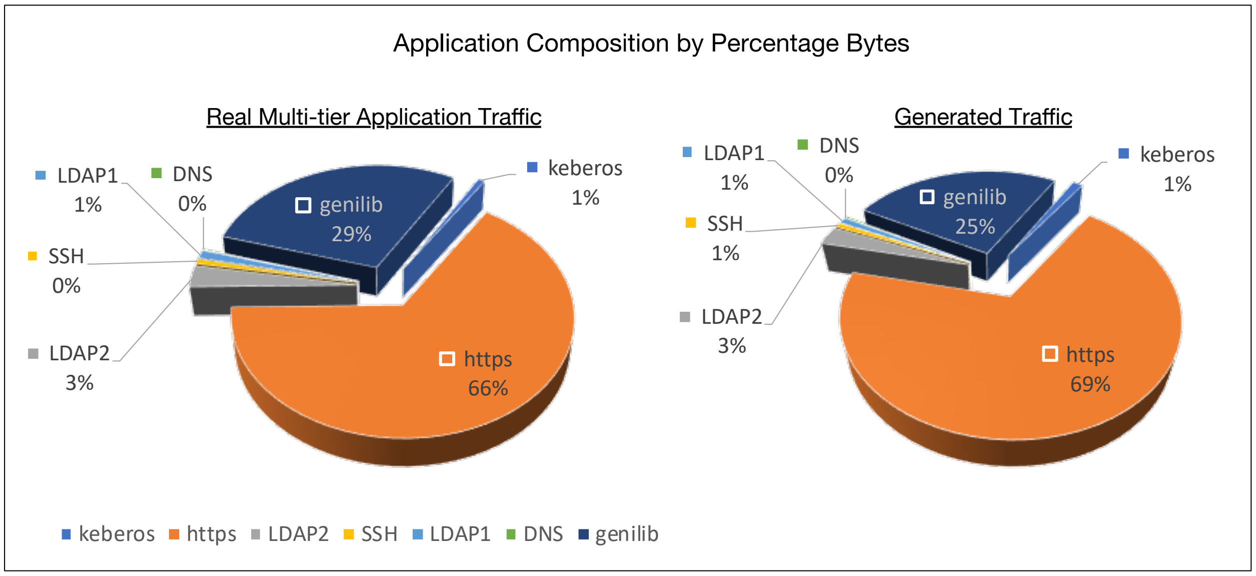


Fig. 5.26: Byte counts by application for modeling and generation of a multi-tier network application traffic

5.3 Comparing Realistic Traffic Modeling Algorithms - Effect of Modeling Higher Level Application Network Behavior

Our framework primarily makes it easy for users to implement desired traffic modeling and generation algorithms and provide the tools to evaluate the performance of the models created. Our evaluation system also makes performance comparison of multiple traffic modeling easy. This section compares four modeling and generation methods to determine which one performs better for a given realistic traffic generation objectives.

Using the modeling system, we created four traffic models file based on four different modeling algorithms we developed. We named the modeling methods `uhgeneric-v1`, `uhgeneric-v2`, `uhgeneric-v3` and `uhgeneric-v4`. The first three models are earlier iterations (versions) of the same `uhgeneric-v4` modeling method that we described in Fig. 4.7 and which we have been using in all our experiments described so far.

The concepts for high level application behavior, including connection classes, connection pools and request bursts have been discussed earlier in sections 4.1.7, 4.1.8 and 4.1.9. In the first version – i.e., the `uhgeneric-v1` model, we simply model request response exchange patterns between applications. In the `uhgeneric-v2`, we do an upgrade on the first version, by incorporating the modeling request bursts. In the third model (`uhgeneric-v3`) we added support for connection classes. Finally, in the fourth, `uhgeneric-v4`, we included support for connection pools. Hence in this section we evaluate the effect of incorporating each of these concepts into a traffic model by comparing the four the results of traffic generation based on each of the 4 models above.

5.3.1 Dataset Extraction, Traffic Modeling and Generation

In this section, we used the same input trace file described in section 5.2. The trace file was for a multi-tier application containing traffic associated with several network services, including HTTPs, DNS, LDAP, SSH, and genilib services, on a network architecture shown in Fig. 5.13. We applied the captured traffic trace file as input into the dataset extractor to yield a `csv` dataset file as already described in section 5.2.2. This `csv` dataset was then applied to the modeling system and used to create four traffic models file based on four different modeling algorithms we described in the previous subsection.

We went on to create a test network having the same architecture as the network on which the input traffic was captured (Fig. 5.13). Using our repeatable experiment orchestration framework [48], we generated traffic on each host based on each of the four traffic models while capturing the traffic for further evaluation.

5.3.2 Evaluation and Discussion

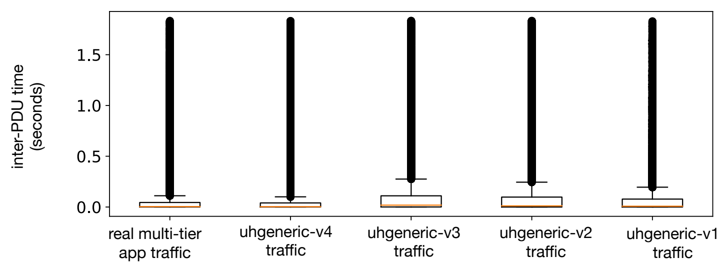
We evaluated the results of the framework by comparing the generated traffic from each of the four models with the original real traffic using our evaluation system described in section 4.4. We carried out both first order and second order analysis to compare the each of the models. The first order analysis shows that all four models do a good job at generating traffic with the producing traffic with distributions that match real multi-tier application traffic for all five metrics considered. However the second order analysis reveals that the `uhgeneric-v4` model performs better than the others in producing traffic with metric variations across user sessions that matches observations in real multi-tier application traffic. This indicates that incorporating request bursts, connection pools and connection classes into the model improves its accuracy in generating realistic traffic.

5.3.2.1 First Order Comparisons of Metric Distributions

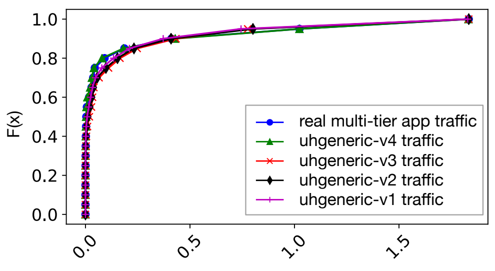
The box plots, cumulative distribution curves of each metric for both the real and generated traffic are given in Figs. 5.27a to 5.31b.

For the inter-PDU time metric, the box plots in Fig. 5.27a shows that the `uhgeneric-v4` traffic has interquartile ranges that are closer to that of the real multi-tier traffic, than the other modeling methods, even though they all have similar outlier value ranges. The large number of outlier values corresponds to the heavy tails that has been found to be associated with typical internet traffic. However, the cumulative frequency plots (Fig. 5.27b) for all four modeling methods closely overlaps with that of the real multi-tier traffic.

In the box plots within Fig. 5.28a, the median and interquartile ranges for PDU sizes for the `uhgeneric-v4` traffic is also much closer to that of the real traffic than that of all other modeling methods. The outlier value ranges in the `uhgeneric-v4` traffic are also much more similar to what

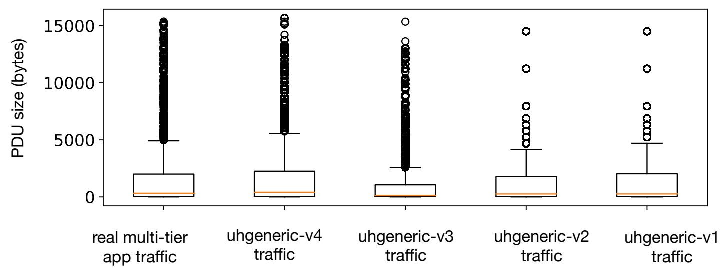


(a) Inter-PDU time Box plots

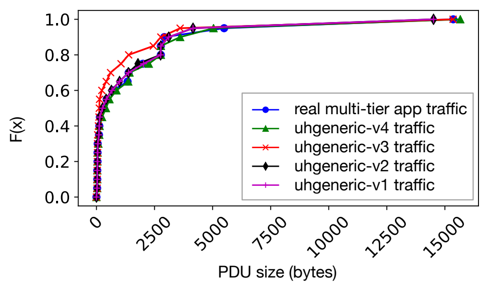


(b) Inter-PDU time CDF

Fig. 5.27: Box plots and cumulative frequency curves for First order comparison of Inter-PDU time between real multi-tier application traffic and traffic generated based on four 4 different traffic modeling algorithms



(a) PDU size Box plots



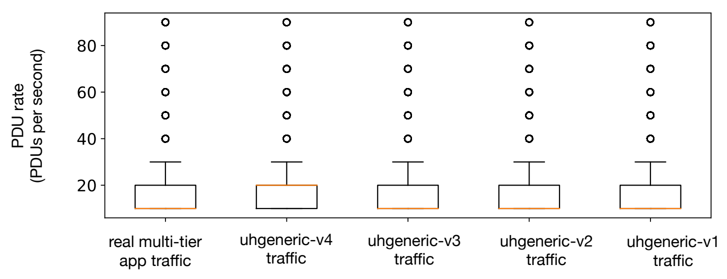
(b) PDU size CDF

Fig. 5.28: Box plots and cumulative frequency curves for First order comparison of PDU sizes between real multi-tier application traffic and traffic generated based on four 4 different traffic modeling algorithms

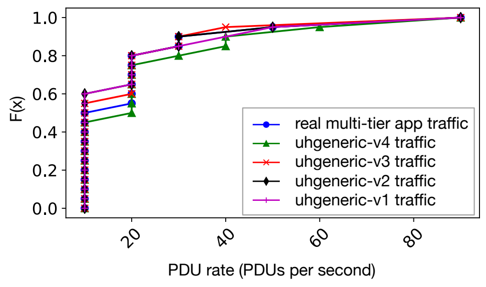
is seen in the multi-tier traffic, indicating that the `uhgeneric-v4` modeling algorithm performs better than the others in modeling our multi-tier traffic.

When we take a look at the box plots in Fig. 5.29a, it is obvious that all four modeling method display similar interquartile ranges and outlier values for the PDU rate metric. The shapes of the cumulative frequency curves (Fig. 5.29b) for each metric are also similar. This indicates that each of the modeling methods do a good job at modeling the multi-tier traffic, in terms of PDU rate distributions.

In the box plots of Fig. 5.30a, none of the traffic modeling methods do a perfect job at generating traffic with the same narrow inter quartile band of PDU throughput as seen in the real multi-tier traffic. However, the cumulative frequency curves (Fig. 5.30b) show an acceptable level of overlap

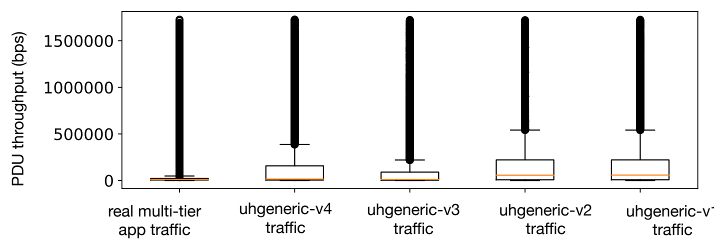


(a) PDU rate Box plots

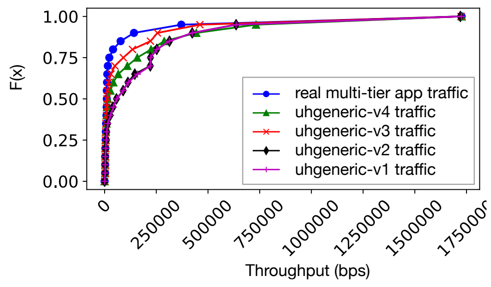


(b) PDU rate CDF

Fig. 5.29: Box plots and cumulative frequency curves for First order comparison of PDU rates between real multi-tier application traffic and traffic generated based on four 4 different traffic modeling algorithms



(a) PDU throughput Box plots



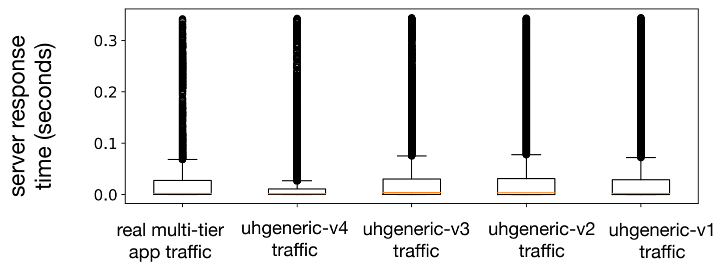
(b) PDU throughput CDF

Fig. 5.30: Box plots and cumulative frequency curves for First order comparison of PDU throughput between real multi-tier application traffic and traffic generated based on four 4 different traffic modeling algorithms

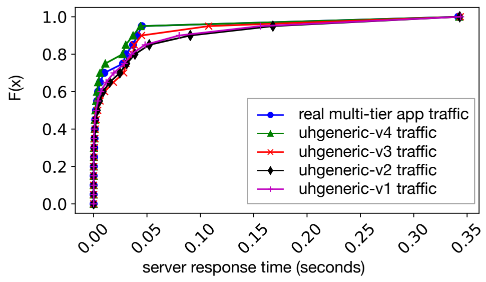
among all the traffic models compared.

For the server response times, the box plots in Fig. 5.31a show that the other models (especially the `uhgeneric-v3` model) performs better than the `uhgeneric-v4` model. The interquartile band of the `uhgeneric-v4` is very narrow, compared to that of the real multi-tier application traffic.

On the aggregate, since the `uhgeneric-v4` model performs better than the others in terms of inter-PDU times and PDU sizes, and it also performs adequately in terms of the PDU rates, we can infer that the model performs better in general, than each of its previous iterations.



(a) PDU rate Box plots



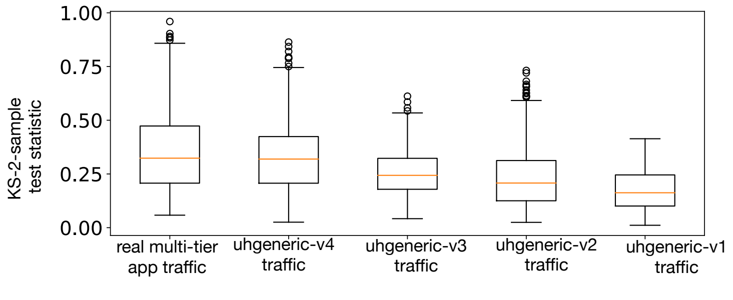
(b) PDU rate CDF

Fig. 5.31: Box plots and cumulative frequency curves for First order comparison of server response times between real multi-tier application traffic and traffic generated based on four 4 different traffic modeling algorithms

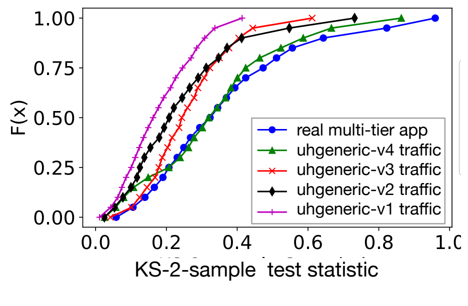
5.3.2.2 Second Order Comparisons of Metric Distributions

The results of the second-order analysis based on pairwise user-session Kolmogorov-Smirnov test statistics for each metric are presented in Figs. 5.32a to 5.36b.

For the inter-PDU time, the box plots in Fig. 5.32a shows the `uhgeneric-v4` traffic model as having identical median value and close over lap with the interquartile ranges of the real multi-tier appplication traffic, in comparison with other modeling methods. The cumulative frequency curves (Fig. 5.32b) also show that the `uhgeneric-v4` curve has a closer overlap with the curve of the real multi-tier application traffic among all the traffic models compared. Thus the inter

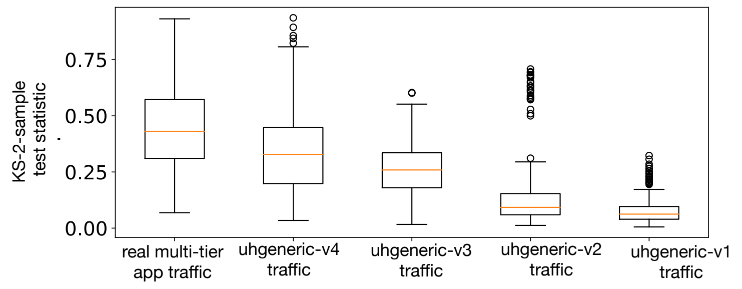


(a) Inter-PDU time Box plots of pairwise user-sessions KS-2-sample test statistics

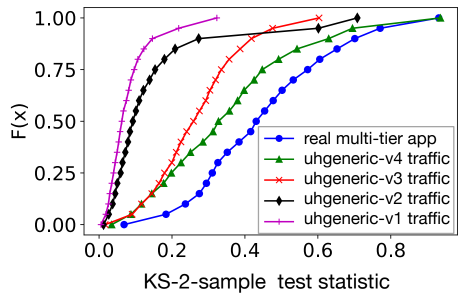


(b) Inter-PDU time cumulative frequency curves for pairwise user-sessions KS-2-sample test statistics

Fig. 5.32: Box plots and cumulative frequency curves for second order comparison of Inter-PDU time between real multi-tier application traffic and traffic generated based on four 4 different traffic modeling algorithms



(a) PDU size Box plots of pairwise user-sessions KS-2-sample test statistics



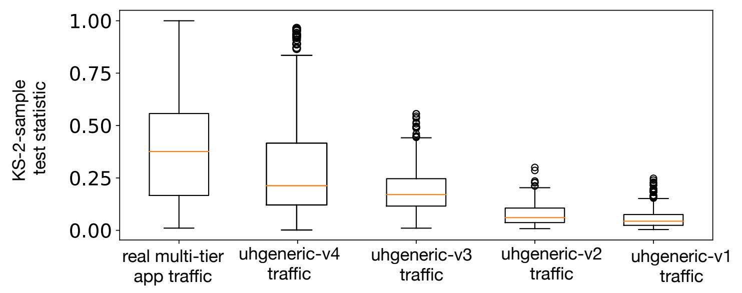
(b) PDU size cumulative frequency curves for pairwise user-sessions KS-2-sample test statistics

Fig. 5.33: Box plots and cumulative frequency curves for second order comparison of PDU sizes between real multi-tier application traffic and traffic generated based on four 4 different traffic modeling algorithms

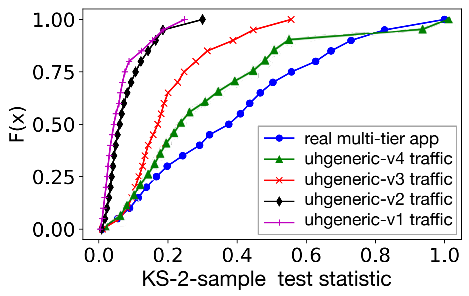
In the box plots within Fig. 5.33a, the median and interquartile ranges for PDU sizes of the the `uhgeneric-v4` traffic are also much closer to that of the real traffic than that of the other modeling methods. The `uhgeneric-v3` model is also more accurate than the earlier 2 versions, but is not as good as the `uhgeneric-v4` model. The cumulative frequency curves also corroborate this, with a marginal closeness between the curves of real multitier app traffic and curves of both `uhgeneric-v4` traffic and the `uhgeneric-v3`.

When we take a look at the box plots and cumulative frequency curves in Fig. 5.34b and Fig. 5.32b, it is again obvious that for the PDU rates metric, the `uhgeneric-v4` model performs better than the other models. The interquartile ranges in the box plots for `uhgeneric-v4` are closer to that of real traffic, than the other models.

In cumulative frequency curves within Fig. 5.35b, we observethat the curves `uhgeneric-v3` traffic and the `uhgeneric-v4` model are closer to that of the original traffic, than the other modeling methods. The same observation is made for the box plots Fig. 5.32a. Hence in terms of PDU throughput, we can confidently deduce that the `uhgeneric-v3` model and the `uhgeneric-v4` model are significantly better than the other two models in modeling this multi-tier traffic.

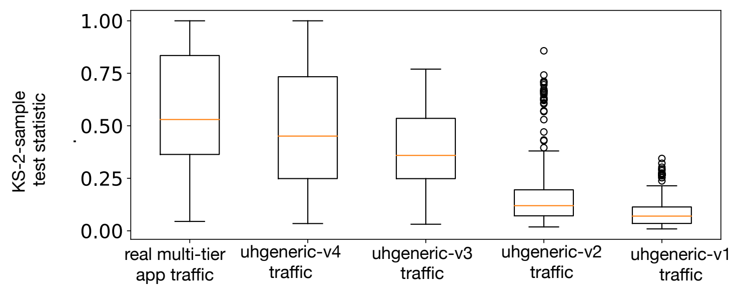


(a) PDU rate Box plots of pairwise user-sessions KS-2-sample test statistics

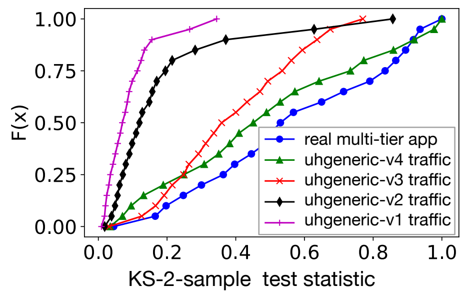


(b) PDU rate cumulative frequency curves for pairwise user-sessions KS-2-sample test statistics

Fig. 5.34: Box plots and cumulative frequency curves for second order comparison of PDU rates between real multi-tier application traffic and traffic generated based on four 4 different traffic modeling algorithms



(a) PDU throughput Box plots of pairwise user-sessions KS-2-sample test statistics

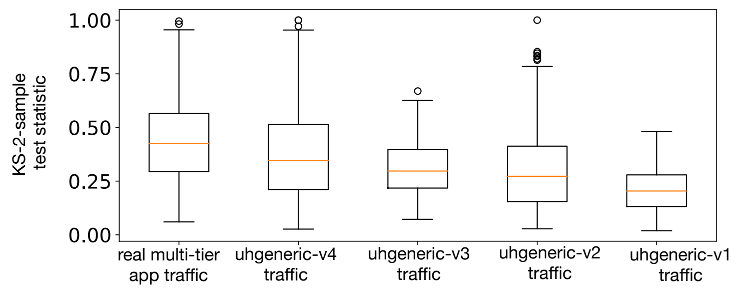


(b) PDU throughput cumulative frequency curves for pairwise user-sessions KS-2-sample test statistics

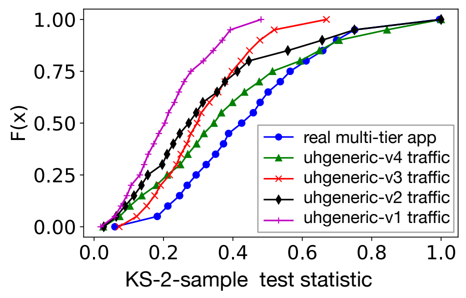
Fig. 5.35: Box plots and cumulative frequency curves for second order comparison of PDU throughput between real multi-tier application traffic and traffic generated based on four 4 different traffic modeling algorithms

Finally, for the server response times, the box plots in Fig. 5.36a show that only the `uhgeneric-v4` traffic modeling method really does a good job at generating traffic with the same inter quartile band as seen in the real multi-tier traffic. However, the cumulative frequency curves in 5.36b indicate that the other models do also generate traffic that is fairly adequte in terms of the server response times distributions.

On the aggregate, both the `uhgeneric-v3` and `uhgeneric-v4` models are more accurate than the others in terms of most metrics considered. Thus we can infer that both modeling methods are better suited for modeling our multi-tier application traffic than their `uhgeneric-v1` and



(a) Server response times Box plots of pairwise user-sessions KS-2-sample test statistics



(b) Server response times cumulative frequency curves for pairwise user-sessions KS-2-sample test statistics

Fig. 5.36: Box plots and cumulative frequency curves for second order comparison of server response times between real multi-tier application traffic and traffic generated based on four 4 different traffic modeling algorithms

`uhgeneric-v2` counterparts. However the `uhgeneric-v4` is the best of all the models.

5.4 Chapter Summary

Our experiments in this chapter have demonstrated that our new traffic modeling and generation framework effectively creates traffic models and generates realistic traffic for diverse kinds of application network traffic. We have also shown that the `uhgeneric-v4` modeling method performs well at generating traffic for simple single service applications and multi-tier applications, based on our set of evaluation metrics.

We have also demonstrated that our framework's evaluation system component effectively compares traffic generated through different traffic modeling different modeling algorithms. The comparison function can help model developers evaluate the improvements gained from new traffic models and support traffic model users to choose the best traffic models out of many options for a specific application network traffic. Our comparison of multiple models in section 5.3.2 also shows that incorporating request bursts, connection pools and connection classes into the model improves its accuracy in generating realistic traffic.

Chapter 6

Conclusion

This dissertation studies network traffic modeling and improves the state of the art in realistic traffic generation with a new framework that makes it possible to create diverse application traffic models and evaluate their performance.

The dissertation presents a survey that identifies 96 traffic generators from a large corpus of conference proceeding publications. We performed a classification of the generators based on the method of traffic generation. We determined the top 10 most popular traffic generators from our survey results by analyzing over 7000 papers published in ACM SIGCOMM and USENIX conferences over the last 13 years. We observed that the set of supported features by each traffic generator vary considerably. Based on each generator's main functionality, we categorized individual generators' features into a structured form to eventually culminate in a traffic generator selection mechanism. The survey outcome indicates that many research projects use constant or maximum throughput generators only, in evaluating new algorithms, protocols, and network functions. Just a few projects use existing realistic traffic generators. The results also reveal no common standards or conventions set for traffic workload generation in networking research.

This dissertation also gives a detailed description of our new framework for traffic modeling and generation. In addition, it presents a new high fidelity traffic modeling and generation algorithm that we developed using the framework. Our evaluation experiments demonstrate that the framework and our traffic modeling algorithm perform well, generating realistic traffic for target applications, including several single-service applications and multi-tier or multi-service applications. The dissertation shows how the framework can be used to compare multiple traffic modeling methods in terms of their effectiveness for realistic traffic generation. The dissertation also proved that the accuracy of our traffic modeling system was significantly improved by including representations for higher level application network behavior such as request bursts, connection pools and connection classes. The traffic models created using the framework can also be shared publicly by model creators with researchers, thus reducing the need for actual traffic from industry operators. Hence, alleviating the privacy concern of many industry operators have in sharing network traffic with third parties.

6.1 Future Work

There are many exciting directions in going forward with this work. We are currently working towards integrating the modeling and traffic generation system into a well-known public computer networking testbed for academia. As future work, our data extraction process can be improved with additional methods for application detection and with additional machine learning based methods for connection class clustering.

The source code, and documentations for all software components of our framework are online [131]. Hence, independent researchers can easily contribute by using the framework to create additional modeling methods and actual models for diverse applications. Therefore we are setting up a public repository where high fidelity application traffic models created with the framework can be made available to researchers, instead of actual traffic traces, reducing the need for researchers to make their models.

The dataset extractor, modeling system, and evaluation system all perform comprehensive processing on network traffic or datasets derived from real network traffic. When processing more massive datasets of input network traffic, the processes take considerable time. As a future work item, each of these components can be made faster with improvements that leverage parallel processing based solution with graphical processing units (GPUs) when available.

Bibliography

[1] Emanuele Acri. “HexInject: The Power of Raw Hex Network Access”. 2017. URL: `http://hexinject.sourceforge.net/` (visited on 04/13/2020).

[2] Oluwamayowa Adeleke. “Echo-State Networks For Network Taffic Prediction”. In: *2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*. Oct. 2019, pp. 202–206.

[3] Oluwamayowa Adeleke, Nicholas Bastin, and Deniz Gurkan. “A Survey of Methods and Outcomes in Network Traffic Generation”. In: *ACM Computing Surveys (Submitted)* vol. 54 (2020).

[4] Oluwamayowa Adeleke, Nicholas Bastin, and Deniz Gurkan. “Network Testing Using a Novel Framework for Traffic Modeling and Generation”. In: *International conference on computer communications and networks (ICCCN)*. Vol. 29. Honolulu, USA: IEEE, 2020, pp. 514–515.

[5] Agilent Technologies. “PIM-SM Multicast Performance Testing”. 2002. URL: `https://literature.cdn.keysight.com/litweb/pdf/5988-6560EN.pdf?id=1649878` (visited on 04/12/2020).

[6] Shane Alcock and Richard Nelson. “Libprotoident: Traffic Classification Using Lightweight Packet Inspection”. In: *WAND Network Research Group, Tech. Rep* (Aug. 2012).

[7] Gianni Antichi, Andrea Di Pietro, Domenico Ficara, Stefano Giordano, Gregorio Procissi, and Fabio Vitucci. “Bruno: A High Performance Traffic Generator for Network Processor”.

In: *Performance Evaluation of Computer and Telecommunication Systems, 2008. SPECTS 2008. International Symposium on*. Edinburgh, UK: IEEE, 2008, pp. 526–533.

[8] Gianni Antichi, Muhammad Shahbaz, Yilong Geng, Noa Zilberman, Adam Covington, Marc Bruyere, Nick Mckeown, Nick Feamster, Bob Felderman, Michaela Blott, Andrew Moore, and Philippe Owezarski. “OSNT: Open Source Network Tester”. en. In: *IEEE Network* 28.5 (Sept. 2014), pp. 6–12.

[9] Stefano Avallone, Marcello Esposito, Antonio Pescape, Simon Pietro Romano, and Giorgio Ventre. “Mtools: A One-Way Delay and Round-Trip-Time Meter”. In: *Recent Advances in Computers, Computing and Communications*. 2002.

[10] Stefano Avallone, Salvatore Guadagno, Donato Emma, Antonio Pescape, and Giorgio Ventre. “D-ITG Distributed Internet Traffic Generator”. In: *Quantitative Evaluation of Systems, 2004. QEST 2004. Proceedings. First International Conference on the*. IEEE, 2004, pp. 316–317.

[11] Yariv Bachar and Ophir Ovadia. “NTGen Project”. 2002. URL: `http://softlab-pro-web.technion.ac.il/projects/NTGen/html/ntgen.htm` (visited on 04/13/2020).

[12] Bastian Ballman and Stefan Krecher. “IP-Packet Generator”. 2005. URL: `http://p-a-t-h.sourceforge.net/html/index.php` (visited on 04/13/2020).

[13] Kiran Bandla. “DPKT: Fast, Simple Packet Creation and Parsing”. Dec. 2019. URL: `https://github.com/kbandla/dpkt` (visited on 12/29/2019).

[14] Paul Barford and Mark Crovella. “Generating Representative Web Workloads for Network and Server Performance Evaluation”. In: *Proceedings of the 1998 ACM SIGMETRICS joint*

international conference on Measurement and modeling of computer systems. Madison Wisconson, USA, 1998, pp. 151–160.

[15] David Barroso. “Yersinia Traffic Generator”. Apr. 2020. URL: `https://github.com/tomac/yersinia` (visited on 04/13/2020).

[16] Nicholas Bastin. “GENI - Virtual Topology Service”. 2017. URL: `https://geni-vts.readthedocs.io/en/latest/` (visited on 11/08/2018).

[17] Bruno Benchimol. “VoIP Traffic Generator”. 2005. URL: `http://voiptg.sourceforge.net/` (visited on 04/13/2020).

[18] Mark Berman, Jeffrey S Chase, Lawrence Landweber, Akihiro Nakao, Max Ott, Dipankar Raychaudhuri, Robert Ricci, and Ivan Seskar. “GENI: A Federated Testbed for Innovative Network Experiments”. In: *Elsivier Computer Networks* 61 (2014), pp. 5–23.

[19] Laurent Bernaille, Renata Teixeira, Ismael Akodkenou, Augustin Soule, and Kave Salamatian. “Traffic Cassification on The Fly”. en. In: *ACM SIGCOMM Computer Communication Review* 36.2 (Apr. 2006), p. 23.

[20] Philippe Biondi. “Scapy - Packet Crafting for Python2 and Python3”. 2011. URL: `https://scapy.net/` (visited on 11/09/2020).

[21] Nicola Bonelli, Stefano Giordano, Gregorio Procissi, and Raffaello Secchi. “Brute: A High Performance and Extensible Traffic Generator”. In: *Proceedings of SPECTS*. 2005, pp. 839–845.

[22] Candela Technologies. “Lanforge: Stateful IP Traffic Generators and Network Emulators”. 2020. URL: `http://www.candelatech.com/` (visited on 04/13/2020).

[23] Pedro Casas, Andreas Sackl, Sebastian Egger-Lampl, and Raimund Schatz. "YouTube & Facebook Quality of Experience in Mobile Broadband Networks". In: *2012 IEEE Globecom Workshops*. Anaheim, California, USA, Dec. 2012, pp. 1269–1274.

[24] Cisco Inc. "Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2017–2022". In: *Global Mobile Data Traffic Forecast* 2017 (2019), p. 2022.

[25] Cisco Inc. "TRex: Realistic Traffic Generator". 2019. URL: `https://trex-tgn.cisco.com/` (visited on 10/09/2020).

[26] Cisco Inc. "Using Test TCP (TTCP) to Test Throughput". en. 2005. URL: `https://www.cisco.com/c/en/us/support/docs/dial-access/asynchronous-connections/10340-ttcp.html` (visited on 04/13/2020).

[27] ZTI Communications. "LanTraffic V2". en-US. 2020. URL: `https://www.zti-communications.com/lantrafficv2/` (visited on 04/13/2020).

[28] Walter De Donato, Antonio Pescape, and Alberto Dainotti. "Traffic Identification Engine: an Open Platform for Traffic Classification". In: *IEEE Network* 28.2 (Mar. 2014), pp. 56–64.

[29] Rogério Leão Santos De Oliveira, Christiane Marie Schweitzer, Ailton Akira Shinoda, and Ligia Rodrigues Prete. "Using Mininet for Emulation and Prototyping Software-Defined Networks". In: *2014 IEEE colombian conference on communications and computing (COLCOM)*. Bogota, Columbia, 2014, pp. 1–6.

[30] Luca Deri, Maurizio Martinelli, Tomasz Bujlow, and Alfredo Cardigliano. "nDPI: Open-Source High-Speed Deep Packet Inspection". In: *Wireless Communications and Mobile Computing Conference (IWCMC), 2014 International*. IEEE, 2014, pp. 617–622.

[31] Valery Diomin and Yakov Tetruashvili. “Cat Karat Packet Builder”. 2010. URL: `https://sites.google.com/site/catkaratpacketbuilder/` (visited on 04/13/2020).

[32] Donfrays Software. “Inter-Networking Test Traffic Generator”. 2018. URL: `http://www.donfraysoftware.com/MITS/MITS.htm` (visited on 04/13/2018).

[33] Jon Dugan, Seth Elliott, Bruce A Mah, Jeff Poskanzer, and Kaustubh Prabhu. “iPerf3, Tool for Active Measurements of the Maximum Achievable Bandwidth on IP Networks”. 2019. (Visited on 09/10/2020).

[34] East Coast Data Comm Inc. “Stateful Traffic Generator”. 2019. URL: `https://www.ecdata.com/Products/Stateful-Traffic-Generator/` (visited on 04/13/2020).

[35] Paris Eloy. “The Network Expect Project”. 2018. URL: `https://www.netexpect.org/` (visited on 04/13/2020).

[36] Paul Emmerich, Sebastian Gallenmüller, Gianni Antichi, Andrew W Moore, and Georg Carle. “Mind the Gap-a Comparison of Software Packet Generators”. In: *2017 ACM/IEEE symposium on architectures for networking and communications systems (ANCS)*. Beijing, China: IEEE, 2017, pp. 191–203.

[37] Paul Emmerich, Sebastian Gallenmüller, Daniel Raumer, Florian Wohlfart, and Georg Carle. “Moongen: A Scriptable High-Speed Packet Generator”. In: *Proceedings of the 2015 internet measurement conference*. Tokyo, Japan: ACM, 2015, pp. 275–287.

[38] Ashok Erramilli, Matthew Roughan, Darryl Veitch, and Walter Willinger. “Self-Similar Traffic and Network Dynamics”. In: *Proceedings of the IEEE* 90.5 (2002), pp. 800–819.

[39] Excentis Inc. “ByteBlower - Making Accurate IP Testing Quick and Easy”. en. July 2013. URL: `https://www.excentis.com/products/byteblower` (visited on 04/13/2020).

[40] Dewan Md Farid, Nouria Harbi, and Mohammad Zahidur Rahman. "Combining Naive Bayes and Decision Tree for Adaptive Intrusion Detection". In: *International journal of Network Security & Its Applications* 2.2 (Apr. 2010), pp. 12–25.

[41] Wu-chang Feng, Ashvin Goel, Abdelmajid Bezzaz, Wu-chi Feng, and Jonathan Walpole. "Tcpivo: A High-Performance Packet Replay Engine". In: *Proceedings of the ACM SIGCOMM workshop on Models, methods and tools for reproducible network research.* Karlsruhe, Germany: ACM, 2003, pp. 57–64.

[42] B Fink and R Scott. "Nuttcp, v5. 3.1". 2006. URL: `https://www.nuttcp.net/` (visited on 01/23/2020).

[43] Matias Fontanini. "Libtins: C++ Packet Sniffing and Crafting Library". 2019. URL: `https://libtins.github.io/` (visited on 04/13/2020).

[44] Romain Fontugne, Pierre Borgnat, Patrice Abry, and Kensuke Fukuda. "MAWILab: Combining Diverse Anomaly Detectors for Automated Anomaly Labeling and Performance Benchmarking". In: *ACM CoNEXT '10.* Dec. 2010.

[45] Fabien Geyer, Stefan Schneele, and Georg Carle. "Reneto, a Realistic Network Traffic Generator for Omnet++/INET". In: *Proceedings of the 6th international ICST conference on simulation tools and techniques.* Cannes, France: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2013, pp. 73–81.

[46] Giovanni Giacobbi. "The GNU Netcat Project". 2014. URL: `http://netcat.sourceforge.net` (visited on 10/22/2019).

[47] GL Communications. “GL Traffic Generator: Simulation & Analysis Network Traffic Characteristics”. 2020. URL: https://www.gl.com/traffic-generators.html (visited on 04/13/2020).

[48] Deniz Gurkan and Nicholas Bastin. “UH Netlab Experiment Orchestration Service”. 2017. URL: https://bitbucket.org/UH-netlab/uhexp/src/default/ (visited on 11/08/2018).

[49] Hakawati. “Hakawati - Traffic Generators”. TISTORY, Nov. 2018. URL: http://www.hakawati.co.kr/318 (visited on 09/09/2020).

[50] Poul E. Heegaard. “GenSyn - a Generator of Synthetic Internet Traffic Used in QoS Experiments”. In: *Proceedings of 15th Nordic Teletraffic Seminar*. 2000.

[51] Harry Heffes and David Lucantoni. “A Markov Modulated Characterization of Packetized Voice and Data Traffic and Related Statistical Multiplexer Performance”. In: *IEEE Journal on Selected Areas in Communications* 4.6 (Sept. 1986), pp. 856–868.

[52] Eric Lee Helvey. “Trafgen: An Efficient Approach to Statistically Accurate Artificial Network Traffic Generation”. en. PhD thesis. Ohio University, 1998. URL: https://etd.ohiolink.edu/pg_10?0::NO:10:P10_ACCESSION_NUM:ohiou1176494135 (visited on 04/13/2020).

[53] Hobbit. “Netcat 1.10”. 1995. URL: http://nc110.sourceforge.net/ (visited on 12/02/2019).

[54] Charles Hornig. “RFC 894: Standard for the Transmission of IP Datagrams Over Ethernet Networks”. Tech. rep. Cambridge, Massachusetts, USA: Internet Engineering Task Force, 1984.

[55] IANA. “Service Name and Transport Protocol Port Number Registry”. URL: https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml (visited on 05/25/2020).

[56] Colasoft Inc. "Colasoft Packet Builder - Colasoft". 2020. URL: `https://www.colasoft.com/download/products/download_packet_builder.php` (visited on 04/13/2020).

[57] R. Jain and S. Routhier. "Packet Trains–Measurements and a New Model for Computer Network Traffic". In: *IEEE Journal on Selected Areas in Communications* 4.6 (Sept. 1986), pp. 986–995.

[58] Rick Jones et al. "Netperf: A Network Performance Benchmark". In: *Information Networks Division, Hewlett-Packard Company* (1996). URL: `https://github.com/HewlettPackard/netperf`.

[59] Roel Jonkman. "Netspec: Philosopy, Design and Implementation". PhD thesis. Lawrence, Kansas, USA: University of Kansas, 1994.

[60] K. Kant, V. Tewari, and R. Iyer. "Geist: A Generator for E-Commerce & Internet Server Traffic". en. In: *2001 IEEE International Symposium on Performance Analysis of Systems and Software. ISPASS.* Tucson, Arizona, USA: IEEE, 2001, pp. 49–56.

[61] Stein Karyl. "Spak-0.6b - Arbitrary Packet Generator/Sender". 1998. URL: `http://static.lwn.net/lwn/1998/0312/a/spak.html` (visited on 05/07/2020).

[62] Konstantinos V. Katsaros, George Xylomenos, and George C. Polyzos. "GlobeTraff: A Traffic Workload Generator for the Performance Evaluation of Future Internet Architectures". In: *2012 5th International Conference on New Technologies, Mobility and Security (NTMS).* May 2012, pp. 1–5.

[63] Keysight Technologies. "BreakingPoint VE - Virtualized Security Resilience Testing for Enterprise-Wide Networks". 2020. URL: `https://www.ixiacom.com/products/breakingpoint-ve` (visited on 04/13/2020).

[64] Keysight Technologies. “Ixchariot - Instant Performance Assessment of Complex Networks from Pre- to Post-Deployment”. 2020. URL: https://www.ixiacom.com/products/ixchariot (visited on 09/13/2020).

[65] Keysight Technologies. “Ixnetwork - L2-3 Network Infrastructure Performance Testing That Scales to Business Needs”. 2020. URL: https://www.ixiacom.com/products/ixnetwork (visited on 04/12/2020).

[66] Samad S Kolahi, Shaneel Narayan, Du DT Nguyen, and Yonathan Sunarto. “Performance Monitoring of Various Network Traffic Generators”. In: *2011 UkSim 13th international conference on computer modelling and simulation*. Cambridge, UK: IEEE, 2011, pp. 501–506.

[67] G Kramer. “Generator of Self-Similar Traffic”. 2014. URL: http://research.glenkramer.com/code/trf_gen3.shtml (visited on 04/13/2020).

[68] Charles Krasic. “Home Page of Mxtraf”. URL: http://mxtraf.sourceforge.net/ (visited on 04/13/2020).

[69] Chia-Yu Ku, Ying-Dar Lin, Yuan-Cheng Lai, Pei-Hsuan Li, and Kate Ching-Ju Lin. “Real Traffic Replay Over Wlan with Environment Emulation”. In: *2012 IEEE Wireless Communications and Networking Conference (WCNC)*. Apr. 2012, pp. 2406–2411.

[70] Juha Laine, Sampo Saariso, and Ruii Prior. “RUDE & CRUDE Traffic Generator”. 2002. URL: http://rude.sourceforge.net/ (visited on 04/13/2020).

[71] Kun-Chan Lan and John Heidemann. “Rapid Model Parameterization from Traffic Measurements”. In: *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 12.3 (2002), pp. 201–229.

[72] LBNL/ICSI berkley lab. "LBNL/LCSI Enterprise Tracing Project - Trace File Download". URL: `http://www.icir.org/enterprise-tracing/download.html`.

[73] W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson. "On the Self-Similar Nature of Ethernet Traffic". In: *IEEE/ACM Transactions on Networking* 2.1 (Feb. 1994), pp. 1–15.

[74] Leo Liang. "IPGen IP Packets Generator". en. 2016. URL: `https://sourceforge.net/projects/ipgen/` (visited on 04/13/2020).

[75] Hongqiang Harry Liu, Yibo Zhu, Jitu Padhye, Jiaxin Cao, Sri Tallapragada, Nuno P Lopes, Andrey Rybalchenko, Guohan Lu, and Lihua Yuan. "Crystalnet: Faithfully Emulating Large Production Networks". In: *Proceedings of the 26th symposium on operating systems principles*. Shanghai, China, 2017, pp. 599–613.

[76] Yingqiu Liu, Wei Li, and Yunchun Li. "Network Traffic Classification Using K-means Clustering". In: *Second International Multi-Symposiums on Computer and Computational Sciences (IMSCCS 2007)*. Aug. 2007, pp. 360–365.

[77] B.A. Mah, P. Sholander, L. Martinez, and L. Tolendino. "IPB: An Internet Protocol Benchmark Using Simulated Traffic". In: *Proceedings. Sixth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (Cat. No.98TB100247)*. July 1998, pp. 77–84.

[78] Jukka Manner. "Jugi's Traffic Generator (jtg)". 2006. URL: `http://www.netlab.tkk.fi/~jmanner/jtg.html` (visited on 04/13/2020).

[79] MARA82 Marathe and W Hawe. "Predicted Capacity of Ethernet in a University Environment". In: *Proceedings of southcon*. Orlando, Florida, USA, 1982, pp. 1–10.

[80] Frank J Massey Jr. “The Kolmogorov-Smirnov Test for Goodness of Fit”. In: *Journal of the American statistical Association* 46.253 (1951), pp. 68–78.

[81] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. “OpenFlow: Enabling Innovation in Campus Networks”. In: *ACM SIGCOMM Computer Communication Review* 38.2 (2008), pp. 69–74.

[82] Pooja Mehta and Ruchil Shah. “A Survey of Network Based Traffic Classification Methods”. In: *Database Systems Journal* 7.4 (), pp. 24–31.

[83] Sándor Molnár, Peter Megyesi, and Geza Szabo. “How to Validate Traffic Generators”. In: *Communications workshops (ICC), 2013 IEEE international conference on.* Budapest, Hungary: IEEE, 2013, pp. 1340–1344.

[84] David Mosberger and Tai Jin. “Httperf — a Tool for Measuring Web Server Performance”. In: *SIGMETRICS Perform. Eval. Rev.* 26.3 (Dec. 1998), pp. 31–37.

[85] Dan Nagle. “Packet Sender - Free Utility to for Sending and Receiving of Network Packets”. en. 2020. URL: `https://PacketSender.com/` (visited on 04/13/2020).

[86] Arvind Narayanan and Vitaly Shmatikov. “Robust De-Anonymization of Large Sparse Datasets”. In: *Security and privacy, 2008. SP 2008. IEEE symposium on.* Oakland, California, USA: IEEE, 2008, pp. 111–125.

[87] Nathan Jeff. “Nemesis Packet Injection Tool Suite”. 2013. URL: `http://nemesis.sourceforge.net/` (visited on 04/13/2020).

[88] Naval Research Laboratory. “Mgen User’s and Reference Guide Version 5.0”. URL: `https://downloads.pf.itd.nrl.navy.mil/docs/mgen/mgen.html` (visited on 04/13/2020).

[89] Naval Research Laboratory. “Multi-Generator (mgen)”. 2019. URL: `https://www.nrl.navy.mil/itd/ncs/products/mgen` (visited on 04/13/2020).

[90] Juniper Networks. “WRAP17 Traffic Generator”. Apr. 2020. URL: `https://github.com/Juniper/warp17` (visited on 04/13/2020).

[91] NMap. “Nping - Network Packet Generation Tool / Ping Utiliy”. 2019. URL: `https://nmap.org/nping/` (visited on 04/12/2020).

[92] Ilkka Norros. “A Storage Model with Self-Similar Input”. en. In: *Queueing Systems* 16.3-4 (Sept. 1994), pp. 387–396.

[93] Robert Olsson. “Pktgen: The Linux Packet Generator”. In: *Proceedings of the linux symposium*. Vol. 2. Ottawa, Canada, 2005, pp. 11–24.

[94] Omnicor. “Network Testing Tools”. 2018. URL: `http://www.omnicor.com/products/network-testing-tools` (visited on 09/13/2020).

[95] Alan Ott. “PlayCap Packet Replay”. Apr. 2020. URL: `https://github.com/signal11/PlayCap` (visited on 04/13/2020).

[96] Pacgen Team. “Pacgen Packet Generator”. en. 2013. URL: `https://sourceforge.net/projects/pacgen/` (visited on 04/13/2020).

[97] Packet Data Systems Ltd. “IP-Traffic Test and Measure”. 2019. URL: `https://www.pds-test.co.uk/products/ip_test_measure.html` (visited on 04/13/2020).

[98] Packeth Team. “packeth”. 2018. URL: `http://packeth.sourceforge.net/packeth/Home.html` (visited on 04/13/2020).

[99] V. Paxson and S. Floyd. “Wide Area Traffic: The Failure of Poisson Modeling”. In: *IEEE/ACM Transactions on Networking* 3.3 (June 1995), pp. 226–244.

[100] PB Software. “Network Traffic Generator and Monitor”. 2018. URL: `http://pbsftwr.tripod.com/id17.html` (visited on 04/13/2020).

[101] Esteban Pellegrino. “pellegre/libcrafter”. Apr. 2020. URL: `https://github.com/pellegre/libcrafter` (visited on 04/12/2020).

[102] Postel. “TG Tool”. 2017. URL: `http://www.postel.org/tg/` (visited on 04/13/2020).

[103] Jon Postel. “RFC 768: User Datagram Potocol”. Tech. rep. Marina del Rey, California, USA: Internet Engineering Task Force, 1981.

[104] Jon Postel. “RFC 791: Internet Protocol”. Tech. rep. Marina del Rey, California, USA: Internet Engineering Task Force, 1981.

[105] Jon Postel. “RFC 793: Transmission Control Protocol”. Tech. rep. Marina del Rey, California, USA: Internet Engineering Task Force, 1981.

[106] Read the docs Inc. “Home — Read the Docs”. URL: `https://readthedocs.org/` (visited on 08/20/2020).

[107] Vinay Ribeiro, Ryan King, and Niels Hoven. “Poisson Traffic Generator”. 2003. URL: `http://www.spin.rice.edu/Software/poisson_gen/` (visited on 04/13/2020).

[108] Chloé Rolland, Julien Ridoux, Bruno Baynat, and Vincent Borrel. “Using LitGen, a Realistic IP Traffic Model, to Evaluate the Impact of Burstiness on Performance”. In: *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops.* Marseille, France: Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering (ICST), 2008, p. 26.

[109] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C Snoeren. "Inside the Social Network's (datacenter) Network". In: *Proceedings of the 2015 ACM conference on special interest group on data communication*. London, United Kingdom, 2015, pp. 123–137.

[110] M Samidi. "Real-Time Voice Traffic Generator". 2004. URL: `http://static.lwn.net/lwn/1998/0312/a/spak.html` (visited on 05/07/2020).

[111] Henning Schulzrinne. "Traffic Generators". 2017. URL: `https://www.cs.columbia.edu/~hgs/internet/traffic-generator.html`.

[112] Faculty of Management Science and Informatics. "Network Information Library - Traffic Generators". 2019. URL: `https://nil.uniza.sk/traffic-generators-list/`.

[113] Douglas C Sicker, Paul Ohm, and Dirk Grunwald. "Legal Issues Surrounding Monitoring During Network Research". In: *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*. New York, New York, United States: ACM, 2007, pp. 141–148.

[114] Charles Robert Simpson and George F. Riley. "NETI@home: A Distributed Approach to Collecting End-to-End Network Performance Measurements". en. In: *Passive and Active Network Measurement*. Ed. by Chadi Barakat and Ian Pratt. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2004, pp. 168–174.

[115] Peter Siska, Marc Ph Stoecklin, Andreas Kind, and Torsten Braun. "A Flow Trace Generator Using Graph-Based Traffic Classification Techniques". In: *Proceedings of the 6th international wireless communications and mobile computing conference*. Caen, France: ACM, 2010, pp. 457–462.

[116] Skaion Corporation. "Skaion Traffic Generation System (TGS)". 2015. URL: `http://www.skaion.com/` (visited on 04/13/2020).

[117] Software Freedom Conservancy. “SeleniumHQ Browser Automation”. URL: https://www.selenium.dev/ (visited on 08/20/2020).

[118] SolarWinds. “Network Traffic Generator – WAN Killer Test”. en. 2020. URL: https://www.solarwinds.com/engineers-toolset/use-cases/traffic-generator-wan-killer (visited on 04/13/2020).

[119] Solarwinds. “WAN Killer Network Traffic Generator”. 2020. URL: https://www.solarwinds.com/engineers-toolset/use-cases/traffic-generator-wan-killer.

[120] Joel Sommers, Hyungsuk Kim, and Paul Barford. “Harpoon: A Flow-Level Traffic Generator for Router and Network Tests”. In: *SIGMETRICS Perform. Eval. Rev.* 32.1 (June 2004), pp. 392–392. URL: http://doi.acm.org/10.1145/1012888.1005733.

[121] Dug Song. “Fragroute”. 2000. URL: https://www.monkey.org/~dugsong/fragroute/ (visited on 04/13/2020).

[122] Spirent Communications. “Spirent TestCenter—Verifying Network and Cloud Evolution - Spirent”. 2020. URL: https://www.spirent.com/products/testcenter (visited on 04/13/2020).

[123] P Srivats. “Ostinato Packet Generator”. 2017. URL: http://ostinato.org/ (visited on 11/08/2018).

[124] Universita’ degli Studi di Napoli. “Other Internet Traffic Generators”. 2020. URL: http://www.grid.unina.it/software/ITG/link.php (visited on 04/13/2020).

[125] Qbone Testbed. “Gen_send, Gen_recv: A Simple Udp Traffic Generator Application”. URL: http://www.citi.umich.edu/projects/qbone/generator.html (visited on 04/13/2020).

[126] TFGen Team. “TFGen Traffic Generator”. 2000. URL: `http://www.pgcgi.com/hptools/` (visited on 04/13/2020).

[127] The Wireshark Team. “Traffic Generator Tools List - Traffic Generator Tools List”. 2019. URL: `https://wiki.wireshark.org/Tools`.

[128] A. Tirumala, Feng Qin, Jon Dugan, Jim Ferguson, and Kevin Gibbs. “Iperf: The Tcp/Udp Bandwidth Measurement Tool”. 2005. URL: `http://dast.nlanr.net/Projects` (visited on 12/15/2019).

[129] Triticom. “LANDecoder32 LAN Protocol Analyzer and Traffic Monitor”. 2006. URL: `http://www.netunlim.com/master_site/pdfs/LD32_V3.4.pdf`.

[130] Aaron Turner. “Tcpreplay”. 2011. URL: `http://tcpreplay.synfin.net/trac/` (visited on 11/13/2019).

[131] University of Houston Netlab. “Documentation for all Traffic Modeling and Generation Projects”. 2018. URL: `http://docs.uh-netlab.org/projects/trafficmodeling` (visited on 11/13/2018).

[132] University of Houston Netlab. “Documentation of Papers Analysis for the Traffic generator Survey”. 2019. URL: `https://docs.uh-netlab.org/projects/surveypaperanalysis/` (visited on 09/09/2020).

[133] University of Houston Netlab. “Documentation of traffic_metrics: Network Traces Analysis Tool”. 2019. URL: `http://docs.uh-netlab.org/projects/traffic_metrics/` (visited on 11/08/2019).

[134] Antoine Varet and Nicolas Larrieu. “Realistic Network Traffic Profile Generation: Theory and Practice”. In: *Computer and Information Science* 7.2 (2014), pp–1.

[135] Matti Vattinen. “epb - Ethernet Packet Generator”. 2019. URL: `http://m-a-z.github.io/epb/` (visited on 04/13/2020).

[136] Kashi Venkatesh Vishwanath and Amin Vahdat. “Evaluating Distributed Systems: Does Background Traffic Matter?” In: *USENIX annual technical conference*. Boston, Massachusetts, 2008, pp. 227–240.

[137] Kashi Venkatesh Vishwanath and Amin Vahdat. “Swing: Realistic and Responsive Network Traffic Generation”. In: *IEEE/ACM Transactions on Networking (TON)* 17.3 (2009), pp. 712–725.

[138] Joerg Wallerich. “NSWEB Traffic Generator”. 2008. URL: `https://www.net.t-labs.tu-berlin.de/~joerg/` (visited on 04/13/2020).

[139] Ulrich Weber. “mausezahn”. Nov. 2019. URL: `https://github.com/uweber/mausezahn` (visited on 04/13/2020).

[140] Michele C. Weigle. “Web Traffic Generation in NS2 with PackMime-HTTP”. 2011. URL: `https://www.cs.odu.edu/~mweigle/research/packmime/` (visited on 04/12/2020).

[141] Michele C. Weigle, Prashanth Adurthi, Félix Hernández-Campos, Kevin Jeffay, and F. Donelson Smith. “Tmix: A Tool for Generating Realistic Tcp Application Workloads in NS2”. In: *SIGCOMM Comput. Commun. Rev.* 36.3 (July 2006), pp. 65–76. URL: `http://doi.acm.org/10.1145/1140086.1140094`.

[142] Keith Wiles. “The DPDK Pktgen Application - Documentation”. Aug. 2019. URL: `https://pktgen-dpdk.readthedocs.io/en/latest/`.

[143] Walter Willinger. “The Discovery of Self-Similar Traffic”. In: *Performance evaluation: origins and directions*. Berlin, Germany: Springer, 2000, pp. 513–527.

[144] Walter Willinger, Murad S Taqqu, Robert Sherman, and Daniel V Wilson. “Self-Similarity Through High-Variability: Statistical Analysis of Ethernet LAN Traffic at the Source Level”. In: *IEEE/ACM Transactions on Networking (ToN)* 5.1 (1997), pp. 71–86.

[145] Michael Wilson. “A Historical View of Network Traffic Models”. In: *Unpublished survey paper* (2006). URL: `https://www.cse.wustl.edu/~jain/cse567-06/ftp/traffic_models2/` (visited on 08/08/2020).

[146] Tao Ye, Darryl Veitch, Gianluca Iannaccone, and S Bhattacharya. “Divide and Conquer: PC-based Packet Trace Replay at OC-48 Speeds”. In: *Testbeds and research infrastructures for the development of networks and communities, 2005. tridentcom 2005. first international conference on.* Trento, Italy: IEEE, 2005, pp. 262–271.

[147] Andy Yeow and Chin Heng. “Bit-Twist: Libpcap-Based Ethernet Packet Generator”. 2006. URL: `http://bittwist.sourceforge.net/` (visited on 04/12/2020).

[148] Ruixi Yuan, Zhu Li, Xiaohong Guan, and Li Xu. “An SVM-Based Machine Learning Method for Accurate Internet Traffic Classification”. In: *Information Systems Frontiers* 12.2 (Apr. 2010), pp. 149–156.

[149] Petr Zach, MARTIN Pokorny, and ARNOST Motycka. “Design of Software Network Traffic Generator”. In: *Recent Advances in Circuits, Systems, Telecommunications and Control* (2013), pp. 244–251.

[150] Sebastian Zander, David Kennedy, and Grenville Armitage. “Kute a High Performance Kernel-Based Udp Traffic Engine”. Tech. rep. 0501118A. Swinburne University of Technology. Centre for Advanced Internet Architectures (CAIA), 2005.

Appendix A

Definitions of Table Row Headers

A.1 Table 3.3

The descriptions for each feature listed in Table 3.3 are given below.

1 **Set # of packets:** Configure the total number of packets to send.

2 **Set total bytes:** Configure the total number of bytes to send.

3 **Set fixed throughput:** Set a fixed value for the throughput in bits per second (bps).

4 **Set randomized throughput:** Configure set of values, or a random distribution for the throughput at which to send packets.

5 **Set packet rate:** Configure a fixed value in packet rates per second (pps) at which packets should be sent.

6 **Set time duration:** Set a time limit for the duration of the traffic generation process.

7 **Send data files:** Configure the generator to use an arbitrary data file as data source for the payload of the packets to be sent.

8 **Replay traffic traces:** Generator supports the replay network traffic trace files.

9 **Set fixed packet size:** Configure a packet size in bytes, for all packets to be sent by the generator.

10 **Set randomized packet sizes:** Configure packet sizes to be picked from a set of values. These values can be picked from a particular random distribution.

11 **Set fixed inter-packet time:** Set a fixed value for inter-packet time intervals in seconds for the packets.

12 **Set randomized inter-packet times:** Configure inter-packet time values to be picket from a set of values or from a random distribution.

13 **Support TCP connections:** Generator supports actual TCP connections, and not just 1-sided flows.

14 **Support SCTP connections:** Generator supports actual SCTP connections, and not just 1-sided flows.

15 **Set MSS:** Configure a fixed value for maximum segment size(MSS).

16 **Set reporting intervals:** Configure time intervals at which to show a summary of the packets sent so far, while the generation process is ongoing.

17 **Set interface:** Select the network interface on which to send out packets.

18 **Specify IP address of interface:** Select the interface on which to send out packets, by specifying the IP address associated with the interface.

19 **Set CPU affinity:** Select a CPU core to use for the packet generation process on multi-core systems.

20 **Generate IP fragments:** Native support for the generation of fragmented IP packets.

21 **Bi-directional generation:** Native support for sending packets in both directions, from the source and the target, each one towards the other.

22 **Multiple parallel connections/ flows:** Native support for sending packets associated with multiple flows or connections simultaneously.

23 **Arbitrary http requests:** Configure to send any HTTP request to a target host.

A.2 Table 3.5

The descriptions for each feature listed in Table 3.5 are given below.

1 **Throughput:** The amount of data delivered by the traffic generator from the source to target per unit time, usually measured in bytes per second (bps).

2 **Latency:** The interval between the time a packet is sent from a source, and the time it is received at the destination.

3 **Packet rate:** The number of packets delivered by the traffic generator from the source to target, usually measured in packets per second (pps).

4 **Total no. of packets:** The total number of packets sent from the source to the target during the entire traffic generation process.

5 **Total no. of bytes:** The total amount of data in bytes sent from the source to the sink during the traffic generation process.

6 **Duration:** The total time elapsed during the traffic generation process usually measured in seconds.

7 **Jitter:** The variation in latency of packets usually measured in seconds.

8 **No. of retransmissions:** The total number of packets that had to be re-transmitted during the packet generation process.

9 **No. of drops:** The total number of packets that were sent from the source but not successfully received at the receiver.

10 **MSS:** The maximum segment size of TCP packets sent by the generator.

11 **Congestion win. size(s):** The congestion window size of the sending host of the traffic generator.

12 **CPU demand:** The amount of CPU utilized by the traffic generator.

13 **Number of flows or connections:** The total number of unique connections or the total number of unique flows created by the traffic generation process.

14 **Request/response transaction rates:** For the traffic generators that conform to the request-response model, this is the number of request and response pairs completed per unit time.

Appendix B

Sample Traffic Model

```
{
  "3389.0tcp": {
    "app_name": "3389.0tcp",
    "conn_models": {
      "3389.0tcp_-1": {
        "port_number": 3389,
        "l4_proto": "tcp",
        "request_sizes": {
          "len_dataset": 6408,
          "empirical": {
            "x": [0.0, 5.0, 10.0, 15.0, 20.0, 25.0, 30.0, 35.0, 40.0, ... 100],
            "y": [37.0, 37.0, 37.0, 37.0, 53.0, 53.0, 53.0, 69.0, ... 133.0],
            "mean": 85.24769,
            "max": 821.0,
            "min": 37.0,
            "RMSE_all": 128.84116,
            "ks_2_test": [0.04008, 7e-05],
            "non_outlier_prob": 0.99672,
            "outliers": [ 821.0, 1029.0, 1029.0, 1445.0, ... 21936.0, 21936.0],
            "25_50_75th_percentiles": [53.0, 69.0, 117.0]
          }
        },
        "response_sizes": {
          "len_dataset": 41661,
          "empirical": {
            "x": [0.0, 5.0, 10.0, 15.0, 20.0, 25.0, 30.0, 35.0, 40.0, ... 100.0],
            "y": [19.0, 85.0, 341.0, 645.0, 949.0, 1445.0, 1877.0, ... 15861.0],
            "mean": 4445.84242,
            "max": 15861.0,
            "min": 19.0,
            "RMSE_all": 5371.64928,
            "ks_2_test": [0.02153, 0.0],
            "non_outlier_prob": 1.0,
            "outliers": [],
            "25_50_75th_percentiles": [1445.0, 3621.0, 6421.0]
          }
        },
        "inter_req_burst_time_per_conn": {
          "mtype": "all-models",
          "len_dataset": 6042,
          "empirical": {
            "model_type": "empirical",
            "x": [0.0, 5.0, 10.0, 15.0, 20.0, 25.0, 30.0, 35.0, 40.0, ... 100.0],
            "y": [0.00033, 0.09053, 0.10359, 0.10399, 0.104, 0.10432, ... 0.68996],
            "mean": 0.2189,
            "max": 4.34376,
            "min": 0.00033,
            "RMSE_all": 0.71915,
            "ks_2_test": [0.03519, 0.00115],
          "non_outlier_prob": 0.99239,
          "outliers": [4.66039, 4.71589, 4.72737, 4.76548, 4.7922, ... 217.25088],
          "25_50_75th_percentiles": [0.10442, 0.11208, 0.19198]
        }
      }
      ....
    },
    "3389.0tcp_0": {...}
  },
  "usess_model": {
    "has_overlap_conns": false,
    "connpool_inter_arrival": {
      "mtype": "all-models",
      "len_dataset": 11,
      "empirical": {
        "model_type": "empirical",
        "x": [0.0, 5.0, 10.0, 15.0, 20.0, 25.0, 30.0, 35.0, 40.0, ... 100.0],
        "y": [0.04999, 0.10251, 0.15504, 2.0249, 3.89476, 4.14998, ... 12.74268],
        "mean": 6.58599,
        "max": 12.74268,
        "min": 0.04999,
        "RMSE_all": 5.62786,
        "ks_2_test": [0.18182, 0.98517],
        "non_outlier_prob": 1.0,
        "outliers": [],
        "25_50_75th_percentiles": [4.14998, 6.09221, 10.70509]
      }
    },
    "connpool_interval": {
      "mtype": "all-models",
      "len_dataset": 11,
      "empirical": {
        "model_type": "empirical",
        "x": [0.0, 5.0, 10.0, 15.0, 20.0, 25.0, 30.0, 35.0, 40.0, ... 100.0],
        "y": [0.0046, 0.00479, 0.00498, 0.0053, 0.00562, 0.00592, ... 8.7272],
        "mean": 2.12317,
        "max": 8.7272,
        "min": 0.0046,
        "RMSE_all": 3.70186,
        "ks_2_test": [0.18182, 0.98517],
        "non_outlier_prob": 1.0,
        "outliers": [],
        "25_50_75th_percentiles": [0.00592, 0.00871, 4.22892]
      }
    },
    ...
    "usess_connpool_seqs_distr": {
      "conn_seqs": [
        ["3389.0tcp_0", "3389.0tcp_0", "3389.0tcp_0", "3389.0tcp_-1"],
        ["3389.0tcp_0", "3389.0tcp_-1"]
      ],
      "probs": [0.2857142857142857, 0.7142857142857143]
    }
  }
 }
}
```