UNIVERSIDADE ESTADUAL DE CAMPINAS

Faculdade de Engenharia Elétrica e de Computação

Ánderson dos Santos Paschoalon

SIMITAR: Synthetic and Realistic Network Traffic Generation

SIMITAR: Geração de Tráfego de Rede Sintético e Realístico

CAMPINAS

2019

Ánderson dos Santos Paschoalon

SIMITAR: Synthetic and Realistic Network Traffic Generation

SIMITAR: Geração de Tráfego de Rede Sintético e Realístico

Dissertation presented to the Faculty of Electrical and Computer Engineering of the University of Campinas in partial fulfillment of the requirements for the degree of Master in Electrical Engineering, in the area of Computer Engineering.

Dissertação apresentada à Faculdade de Engenharia Elétrica e Computação da Universidade Estadual de Campinas como parte dos requisitos exigidos para a obtenção do título de Mestre em Engenharia Eletrica, na Àrea de Engenharia de Computação.

Supervisor: Prof. Dr. Christian Rodolfo Esteve Rothenberg

Este exemplar corresponde à versão final da dissertação defendida pelo aluno Ánderson dos Santos Paschoalon , e orientada pelo Prof. Dr. Christian Rodolfo Esteve Rothenberg

CAMPINAS

2019

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca da Área de Engenharia e Arquitetura
Elizangela Aparecida dos Santos Souza - CRB 8/8098

P262s

Paschoalon, Ánderson dos Santos, 1990-
SIMITAR : synthetic and realistic network traffic generation / Ánderson dos Santos Paschoalon. – Campinas, SP : [s.n.], 2019.

Orientador: Christian Rodolfo Esteve Rothenberg.
Dissertação (mestrado) – Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Redes de computadores. 2. Critério de informação de Akaike. 3. Transformada wavelet. 4. Internet. 5. Processo estocástico. 6. Gradiente descendente. I. Esteve Rothenberg, Christian Rodolfo, 1982-. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Informações para Biblioteca Digital

Título em outro idioma: SIMITAR : geração de tráfego de rede sintético e realístico
Palavras-chave em inglês:
Computer networks
Akaike information criterion
Wavele transformed
Internet
Stochastic process
Gradient descending
Área de concentração: Engenharia de Computação
Titulação: Mestre em Engenharia Elétrica
Banca examinadora:
Christian Rodolfo Esteve Rothenberg [Orientador]
Daniel Macêdo Batista
Lee Luan Ling
Data de defesa: 19-03-2019
Programa de Pós-Graduação: Engenharia Elétrica

Identificação e informações acadêmicas do(a) aluno(a)
- ORCID do autor: https://orcid.org/0000-0003-3712-8246
- Currículo Lattes do autor: http://lattes.cnpq.br/7698618670811185

COMISSÃO JULGADORA - DISSERTAÇÃO DE MESTRADO

Candidato: Ánderson dos Santos Paschoalon RA: 083233
Data da Defesa: 17/12/2018
Título da Tese:
"SIMITAR: Synthetic and Realistic Network Traffic Generation"
"SIMITAR: Geração de Tráfego de Rede Sintético e Realístico"

Ata de defesa, com as respectivas assinaturas dos membros da Comissão Julgadora, encontra-se no processo de vida acadêmica do aluno.

Nessa dedicatória, gostaria de agradecer a todos que me ajudarem por essa etapa, direta ou indiretamente. Aqueles que me inspiraram e me motivaram a seguir por esse caminho, aqueles que me ensinaram e me ajudaram durante o processo, e a aqueles cuja simples companhia me deram energia e me motivaram para estar aqui onde estou hoje. Agradeço a todos, seja os que estão listados abaixo, bem como aqueles cuja minha memória não me ajudou na escrita desse texto.

Eu gostaria de agradecer a Universidade Estadual de Campinas (UNICAMP) e a Faculdade de Engenharia Elétrica e Computação (FEEC) pela possibilidadede realizar meu curso de mestrado, e pela infraestrutura que me possibilitou obter este titulo. Este estudo foi financiado em parte pela Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) - Código Financeiro 001

Gostaria de agradecer ao meu professor e orientador Christian Esteve Rothemberg, sem o qual, seja pelo ensino, seja pela orientação e apoio durante o projeto, este trabalho não teria saído do papel. Gostaria de agradecer toda sua paciência e entendimento por esses últimos anos. Sua liderança será uma fonte de inspiração para mim para o restante de minha carreira, e ela está apenas começando. Sem sua idéias inovadoras, suporte e encorajamento continuo, este projeto não teria saído do papel. Especialmente pelo fato de que em pesquisa muitas vezes as coisas não saem como o esperado, e temos que reiniciar do zero o processo. Eu gostaria de expressar a minha gratisão e honra por ter um tão grande orentador, professor, lider e amigo durante estes anos. Agradeço também a todos os Intrigers, colegas de grupo e de bancada, Alex, Javier, Nathan, Cláudio, Daniel, Danny, Gyanesh, Rafael, Fabricio e todos os demais que não mencionei neste texto. Agradeço a todos os demais colegas de laboratório do LCA, em especial a Mijail, Suelen, Amadeu, Paul, ...

Agradeço a todos os companheiros e amigos que fiz em todos esses anos de Unicamp.Agradeço a todos os grandes amigos e companheiros da Opus Dei, em especial Padre Fabiano pelos conselhos, e ao meu amigo Denis, grande amigo pelo apoio.Agradeço aos meus companheiros de minha antiga casa P7, e da moradia, em especial o meu amigo (quase irmão) Lucas Zorzetti (Xildo) . Agradeço a minha namorada Rubia Agondi pelo se apoio, ajuda, amor, compreensão e paciência, e por sempre me fazer acreditar em meu trabalho.Agradeço a minha tão adorada família, a meu Pai Tirso José Paschoalon por todo sua preocupação e ensino. A minha Mãe Rosângela dos Santos Mota, por todo o seu carinho e amor. E a minha irmã Ariela Paschoalon, pela companhia e afeto. E por último e mais importante, agradeço a Deus por todos seu dons, proteção e amor.

Acknowledgements

First, I would like to thanks all who have helped me, directly or indirectly. Those who have inspired me to follow this path, those who have taught and helped me, and those whose just their company had given me motivation and energy to be here today. I thank to all I've listed down below, and all who I forgot to mention.

I would like to thank the State University of Campinas (UNICAMP) and the Faculty of Electrical Engineering and Computing (FEEC) for the possibility of completing my master's degree course and for the infrastructure that enabled me to obtain this title.

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

I would like to thank my advisor Prof. Dr. Christian Rothenberg for the trust and for letting me be part of his selected group of students. I have to say that I'm extremely grateful for all his patience and understanding all over these last years. Also, his leadership will be a source of inspiration for the rest of my career, that is just beginning. I would not be able to imagine the undertaking of this research without his innovative ideas, consistent support and continuous encouragement. Specially encouragement, since sometimes, especially on research things do not happen as we would expect, and start from the beginning is always a hard task. I would like to express my gratitude honor for having such a great instructor, teacher, leader and friend all for these past years.

Thanks to all the intrigers, desk, and group colleagues Alex, Javier, Nathan, Claudio, Daniel, Gyanesh, Raphael, Fabricio and all who I have not mentioned in this text. Also, thanks to all my LCA colleagues, especially Mijail, Suelen, Amadeu, Paul, ...

Thanks to all my colleagues and friends I made all these years I've been at Unicamp.

Thanks to all my Opus Dei friends, especially Priest Fabiano for all his advice, and my friend Denis who have given me a huge support.

Thanks to all my house companions from my old home house P7, and all my "moradia" friends, in particular, my friend (almost brother) Lucas Zorzetti (Xildo).

Thanks to my girlfriend Rubia Agondi, for all her support, help, love, understanding and patience, and for making me always believe on my work.

Thanks to my lovely family, my father Tirso José Paschoalon for all his attention and education. To my mother, Rosangela dos Santos Mota, for all her affection and love. And to my sister Ariela Paschoaln, for her company and affection.

And last, and more important, I thank God for all his gifts, protection, and love.

“ratio in homine sicut Deus in mundo”
“reason in man is rather like God in the world.”
“razão no homem é como Deus no mundo”

De regno ad regem Cypri – Saint Thomas Aquinas (Santo Tomas de Aquino)

Abstract

Real network traffic has a different impact on network devices when compared to constant traffic generated by tools like Iperf, even when both traffic profiles present the same average throughput. Busty traffic may cause buffer overflows while constant traffic does not, decreasing the measurement accuracy. The number of flows may have an impact on flow-oriented devices, such as Software Defined Networking (SDN) switches and controllers. In scenarios where SDN is expected to play an essential role in the future Internet, it becomes crucial that in-depth validation of new technologies considering these aspects. Most of the open-source realistic traffic generator tools have the modeling layer coupled to the traffic generator, making a challenge any update to newer libraries. Most of the existing traffic generators support realistic traffic generation through a broad set of options to be manually but not automatically configured. As a result, generating realistic traffic is a challenging project by itself.

In this work, we explore this subject in-depth. Our main research contributions are: (1) a review on available solutions and network traffic modeling, and (2) the design and implementation of the SIMITAR (SnIfing, ModellIng and TrAffic geneRation) traffic generator. The proposed approach provides a modeling framework separated from the traffic generator, being flow-oriented and auto-configurable. We create and use Compact Trace Descriptor files as inputs - XML files that describe traffic features for our traffic model. We are capable of replicating with accuracy flow characteristics of all tested traffic traces, including the scaling features of some as well. We give a particular focus on inter-packet times modeling, where we propose a methodology based on information criteria for automating the process modeling and selection of the best model. We also propose a cross-validation method to qualify the methodology.

Keywords: traffic generators; network traffic modelling; burstier traffic; realistic traffic; *pcap* file; packet sniffing; inter packet times; linear regression; gradient descendent; Cumulative Distribution Function (CDF); maximum likelihood; Akaike Information Criterion (AIC); Bayesian Information Criterion (BIC); packet trains; Wavelet Multiresolution Analisis; Hurst Exponent.

Resumo

Um tráfego de rede real possui um impacto diferente sobre os nós da rede se comparado ao tráfego constante gerado por ferramentas como Iperf, mesmo com uma mesma taxa de transferência. Um tráfego em rajadas pode causar estouros de buffers enquanto um tráfego constante não, e pode também diminuir a precisão das medições. O número de fluxos pode ter um impacto nos nós orientados a fluxo, como *switches* e controladores SDN. Em um cenário em que as redes definidas por software desempenharão um papel essencial na Internet futura, uma validação mais aprofundada das novas tecnologias, considerando esses aspectos, é crucial. Além disso, a maioria das ferramentas geradoras de tráfego realistas de código aberto tem a camada de modelagem acoplada ao gerador de pacotes, o que dificulta sua atualização para novas bibliotecas, tornando-as freqüentemente desatualizadas. Por fim, a maioria das ferramentas *open-source* que suportam a geração de tráfego realista, oferecem um grande conjunto de opções a serem configuradas, mas não são auto configuráveis. Dessa forma a produção de um tráfego realista customizado torna-se uma tarefa desafiadora.

Neste trabalho nos aprofundamos neste assunto. Como resultado final, para nossa pesquisa destacamos duas contribuições principais: uma investigação de revisão das soluções disponíveis e modelagem de tráfego de rede, e a proposta de nosso próprio gerador de tráfego chamado SIMITAR (acrônimo para *sniffing*, modelagem e geração de tráfego em inglês). Esta tecnologia possui estruturas separadas de modelagem e geração de tráfego, sendo orientada a fluxos e auto configurável. Ela cria e usa Descritores de Tráfego Compactos como arquivos de entrada - Arquivos XML que descrevem características para o nosso modelo de tráfego. Atualmente já conseguimos replicar com precisão métricas do nível de fluxos, e certas características de escala. Demos um enfoque especial na modelagem de tempos entre pacotes, onde propomos uma metodologia baseada em critérios de informação para automatizar a modelagem de processos e seleção do melhor modelo. Também propusemos um método de validação para medir a qualidade deste mesmo método.

Keywords: geradores de tráfego; modelagem de tráfego de rede; tráfego em rajadas; tráfego realistico; arquivo *pcap*; captura de pacotes; tempo entre pacotes; regressão linear; gradiente descendente; Função Distribuição Acumulada; máxima verossimilhança; Critério de informação de Akaike; Critério de informação Bayesiano; trem de pacotes; Análise Wavelet de multiresolução; Expoente de Hurst.

List of Figures

List of Tables

Acronyms

ACK Acknowledge. 43

AIC Akaike information criterion. 25

AICc Akaike's Information Criterion Corrected. 89

API Application programming interface. 22

ARP Address Resolution Protocol. 112

BGP Border Gateway Protocol. 56

BIC Bayesian information criterion. 25

CDF Cumulative Distribution Function. 13, 31

CDT Compact Trace Descriptor. 23

CLI Command Line Interface. 116

DHCP Dynamic Host Configuration Protocol. 56

DIC Deviance Information Criterion. 89

DNS Domain Name System. 56

DUT Device Under Test. 22

flowID Flow Identifier. 41

FNV Fowler-Noll-Vo. 41

FPGA Field Programmable Gate Array. 118

FTP File Transfer Protocol. 33

GAN Generative adversarial network. 90

GUI Graphical User Interface. 116

HTTP Hypertext Transfer Protocol. 33

HTTPS Hyper Text Transfer Protocol Secure. 56

I/O	Input/Output. 22
ICMP	Internet Control Message Protocol. 43
IoT	Internet of Things. 21
IP	Internet Protocol. 33
IPv4	Internet Protocol Version 4. 33
IT	Information Technology. 113
KNI	Kernel NIC Interface. 85
LAN	Local Area Network. 57
M2M	Machine to Machine. 21
MAC	Media Access Control. 112
MANO	Management and Orchestration. 113
MDL	Minimum Description Length. 89
MTU	Maximum transmission unit. 33
NAT	Network Address Translation. 116
NetFPGA	Network FPGA. 28
NFVI	NFV Infrastructure. 113
NIC	Network Interface Card. 28
nMDL	Normalized Minimum Description Length. 89
NOS	Network Operational System. 112
NVF	Network Function Virtualization. 21
PDF	Probability Density Function. 13, 31
PT-MMPP	Power-tail Markov-Modulated. 53
QoE	Quality of service. 34
QoS	Quality of service. 34
QQplot	Quantile-quantile plot. 36

RTT Round Trip Time. 37

SCTP Stream Control Transmission Protocol. 117

SDN Software Defined Networking. 21

SIMITAR SnIffing, ModellIng, and TrAffic geneRation. 21

SNMP Simple Network Management Protocol. 56

SQL Structured Query Language. 41

SSH Secure Shell. 56

SYN Synchronize. 43

TACACS Terminal Access Controller Access-Control System. 56

TCP Transmission Control Protocol. 33

UDP User Datagram Protocol. 33

UML Unified Modeling Language. 24

VLAN Virtual LAN. 117

VNF Virtualized Network Function. 21

WAN Wide area network. 57

WMA Wavelet multi-resolution energy analysis. 35

WSA Wavelet-based scaling analysis. 35

XML Extensible Markup Language. 38

Contents

1 Introduction

1.1 Motivation

The type of traffic used for performing evaluation matters; this is a fact. Studies show that realistic Ethernet traffic provides different and variable load characteristics on routers [Sommers e Barford 2004], even with the same average bandwidth consumption, showing that constant traffic is not sufficient for complete technology validation. This conclusion indicates that tests which employ traffic generators with constant rates are not enough for complete validation of new technologies. Bursty traffic can cause packet losses and buffer overflows, impacting on network performance and measurement accuracy [Cai *et al.* 2009]. Small packets tend to degrade application performance [Srivastava *et al.* 2014]. Furthermore, realistic traffic is essential on security research, such as for the evaluation of firewall middleboxes, studies on intrusion, and malicious workloads [Botta *et al.* 2012].

New networking scenarios such as SDN and virtualized networks (NVF and VNFs) become harder to predict in terms of performance compared to hardware-based technologies, due to the multiple layers of software and platform parameters demanding validation in a broadening range of use cases [Han *et al.* 2015]. Another critical question about the interaction between application-network has had the flow-oriented operation of SDN networks, in which each new flow arriving on an SDN switch demands further communication with the controller. Therefore the controller can be a bottleneck on the switches performance. Also, new types of traffic patterns introduced by IoT and Machine-to-Machine (M2M) communication [Soltanmohammadi *et al.* 2016] increase the complexity of the network traffic characterization, turning pre-defined models used by traffic generators obsolete.

Furthermore, realistic traffic generators are essential security research, since the generation of realistic workloads is essential for evaluation of firewall middleboxes. It includes studies of intrusion, anomaly detection, and malicious workloads. By realistic, we refer to traffic that represents well the traffic features, such as protocols, payloads, and protocols, able to emulate benign and malicious workloads.

Aiming to address these gaps, this dissertation introduces SIMITAR, an auto-configurable network traffic generator. SIMITAR stands for *SnIffing, ModellIng, and TrAffic geneRation*, which correspond to the main operation processes of the proposed framework. SIMITAR has an application independent traffic model, that can represent a wide variety of scenarios. It also decouples the traffic modeling and packet-generation layer, using a factory design pattern, enabling its application on different scenarios, and technology update, via technology abstraction. SIMITAR code and all scripts used in this dissertation are available at

GitHub [Paschoalon 2019] for validation, experiment reproducibility, and re-use purposes.

1.2 Related Work

Traffic generators are tools to transfer or inject network packets in a controlled manner, aiming not at the actual data transfer data but at the functional validation and performance benchmarking of devices under test (DUT) for varying technologies or scenarios. The open-source community offers a vast variety of traffic generators. Since most have been built for specific goals, each uses different methods for traffic generation, and offer control over different traffic features, such as throughput, packet-sizes, protocols, and so on [Botta *et al.* 2012].

Traffic generators can be classified into two main groups: replay engines [Varet 2014] and model-based tools. Replay engines, such as TCPReplay and TCPivo [Feng *et al.* 2003], work replicating in a given network interface a given packet capture file. These tools can generate realistic traffic but have their constraints. They are deterministic since will always reproduce the same traffic from the packet capture. Replay engines require storage of packet capture, what can be a problem for traffics of high bandwidth traffic. Also, they assume the user has access to packet captures appropriate for his testing purposes, which is not always true, due to a limited number of public sources. Model-based tools rely on software models to replicate one or more characteristics of the traffic.

Model-based tools have their limitations as well. Traffic generators that emulate the applications, are designed to represent only specific scenarios on computer networking, and are not enough to represent a large variety of scenarios. Many traffic generator tools only offer constant-rate and Poisson models, which does not represent well the complexity of internet traffic [Leland *et al.* 1994]. Other tools such as D-ITG offer dozens of parameters and models to be configured, but delegate to the user the task of creating, validate and script his traffic model. To the best of our knowledge, we found only two open-source auto-configurable tools: Swing and Harpoon. However, none of them has an extensible architecture, which turns supporting modern and fast I/O APIs (such as DPDK [DPDK – Data Plane Development Kit 2019]) a hard task. Table 1 present a summary of the above mentioned features for some relevant

Table 1 – Comparison of existing traffic generation tools.

Solution	**Auto-configurable**	**Realistic Traffic**	**Traffic Custumization**	**Extensibility**
Harpoon	yes	yes	yes	**no**
D-ITG	**no**	yes	yes	**no**
Swing	yes	yes	**no**	**no**
Ostinato	**no**	**no**	yes	yes
LegoTG	**no**	**no**	yes	yes
sourcesOnOff	**no**	yes	yes	**no**
Iperf	**no**	**no**	yes	**no**
SIMITAR	**yes**	**yes**	**yes**	**yes**

traffic generators: Swing [Vishwanath e Vahdat 2009], Harpoon [Sommers e Barford 2004], sourcesOnOff [Varet 2014], D-ITG [Botta *et al.* 2012], Iperf [iPerf - The network bandwidth measurement tool 2019], Ostinato [Ostinato Network Traffic Generator and Analyzer 2016] and LegoTG [Bartlett e Mirkovic 2015].

1.3 Objectives, Requirements, and Methodology

Based on the provided context, we defined a set of targets for our research:

1. **Survey**: Evaluate open-source Ethernet workload tools and address features each one has. We wanted to know the existing solutions, innovation points on the current state of affairs, and how can we some could be integrated and reused by our solution;
2. **Background studies**: Study the characterization and mathematical modeling of Ethernet traffic, what are the best models and challenges.
3. **Definition of Realistic Traffic**: Define what realistic traffic generation is, and how to measure if any synthetic traffic is realistic or not.
4. **Design**: Create a general method for modeling and parameterization of Ethernet traffic;
5. **Development**: Create a self-configurable tool that observes and uses real network traffic, and reproduce its behavior characteristics, avoiding the storage of large *pcap* files.

Towards the above-stated objectives, we had identified a set of requirements of the envisioned traffic generation tool should meet:

- **Auto-configurable**: It must be able to extract data from real traffic and store in a database, and use it to parametrize its traffic model. It must be able to obtain data from real-time traffics and from *pcap* files;
- **Technology independent**: It must have a flow-based abstract model for traffic generation, not attached to any specific technology.
- **Extensibility**: traffic modeling and generation must be decoupled. Ideally, it must be able to use as a traffic generator engine any library or traffic generator tool;
- **Simple usage**: It must be easy to use. It has to take as input a Compact Trace Descriptor, just as a traffic replay engine (such as TCPreplay) would take a *pcap* file;
- **Human readable model**: it must produce a human-readable file as output that describes our traffic using our abstract model. We call this file a Compact Trace Descriptor (CDT);

- **Traffic generation programmability**: It must have what we call traffic generation programmability. The compact trace descriptor must be simple and easy to read. That way, the user may want to create our custom traffic, in a platform agnostic way.
- **Flow-oriented**: traffic modeling and generation must be flow-oriented. Each flow must be modeled and generated separately.

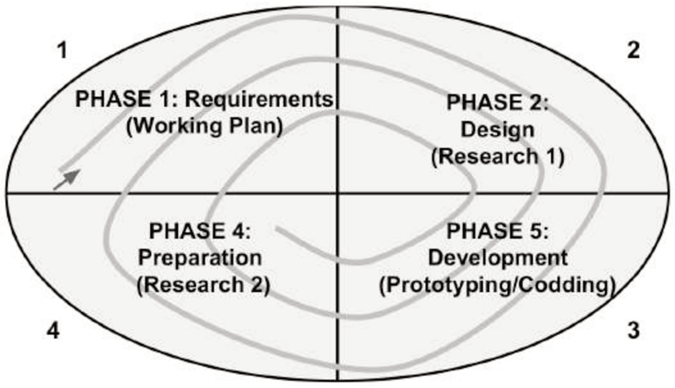

1
2
PHASE 1: Requirements (Working Plan)
PHASE 2: Design (Research 1)
PHASE 4: Preparation (Research 2)
PHASE 5: Development (Prototyping/Codding)
4
3

Figure 1 – Spiral research and development procedure

We have adopted a spiral procedure of development, as suggested Sommerville on *Software Engineering* [Sommerville 2007], but adapted to an academic research process. Figure 1 shows the model of development we had adopted. It had four main phases:

1. *Requirements (working plan)*;
2. *Design (research 1)*;
3. *Development (prototyping/codding)*;
4. *Preparation (research 2).*

On the **Requirements phase**, we create tasks, formalized on *Working Plans* documents. These tasks should cover the whole process. On the **Design (research 1) phase**, where the focus of the research are related works. We research on literature to learn about topics defined by the task, to help we design our solutions. On this step, during the initial phases of the project, we also conceived the architecture along with UML diagrams[1]. Some small changes are inevitable, but structural changes turn to be impractical on later phases. The next phase is the **Development phase**. It is the prototyping and coding phase. The last is the **Preparation phase** when we evaluate the results achieved on the Development phase, and we go back to research again, aiming to prepare the next *Working Plan*. On this phase, the focus of the research is on points of innovation.

[1] **Unified Modeling Language (UML)** is a modeling language designed to provide a standardized way of representing and design systems [Booch 2005].

1.4 Outline

In this introductory Chapter, we had presented an abstract of state of affairs, and the main goals of our research. ***Chapter 2*** presents a survey on open-source traffic generator tools, summarizing the benefits, and features supported by each one. The chapter offers a review of topics on realistic traffic generation and defines important concepts on network traffic modeling; such as self-similarity and heavy-tailed distributions. Also, the chapter presents a survey on techniques for validating traffic generator tools

Chapter 3 presents *SIMITAR* traffic generator. We describe its low-level requirements and define an architecture and their algorithms. We explain its operation and suggest some use cases. In ***Chapter 4***, we go deep on the modeling process we had developed for our traffic generator. We validate the effectiveness of Information Criteria AIC and BIC as a method selection of stochastic models for Ethernet traffic. We also discuss some other algorithms we developed such as *calcOnOff* and the application protocol guesser. In ***Chapter 5***, we define a set of metrics based on previous tests on validation of traffic generators found in the literature. Here, we focus on the packet, flow, and scaling metrics. We test our tool in an emulated SDN testbed with Mininet [Mininet – An Instant Virtual Network on your Laptop (or other PC) 2019][2], using OpenDayLight [The OpenDayLight Platform 2019] as the SDN controller.

Chapter 6 highlights future actions to improve SIMITAR on realism and performance along with other future research avenues, including improving its computational performance, expand it to new APIs of traffic generation and calibration of its constants. Finally, we end the work presentation with a conclusion (***Chapter 7***).

Appendices A, ***B***, and ***C*** provide supplementary materials for Chapter ***2***: Review of Mathematical Concepts, Computer Networks, and Traffic Generators. Appendix ***D*** provides charts in addition to those presented in Chapter 4, and Appendix ***E*** complements the presentation of the architecture discussed in Chapter 3 by presenting SIMITAR UML class diagrams.

[2] **Mininet** is a network emulator. It can run a collection of hosts, switches, routers and links over a single Linux kernel, using lightweight virtualization [Introduction to Mininet · mininet/mininet Wiki 2019].

2 Literature Review

2.1 Traffic Generators

Traffic generators are tools to transfer or inject network packets in a controlled manner, aiming not at the actual data transfer data, but validation and performance benchmarking of devices under test (DUT) [Molnár *et al.* 2013]. There is a vast variety of traffic generators described on literature [Botta *et al.* 2012] and available in the open-source community[1].

Together with many traffic generators, there are many open-source APIs for traffic generation. Some are low-level APIs, which enables precise control of each packet generated, and are used in the implementation of traffic generators[2]. Also, they are computationally more efficient compared to high-level APIs for traffic generation. We've listed some low-level APIs below:

- GNU Socket API (C) [Sockets 2019];
- Libpcap (C) [Tcpdump & Libpcap 2019];
- Libtins (C++) [libtins: packet crafting and sniffing library 2019];
- Scapy (Python) [Scapy – Packet crafting for Python2 and Python3 2019];
- DPDK (C) [DPDK – Data Plane Development Kit 2019].

We also have high-level APIs, usually provide by traffic generator, which simplifies the programming of custom traffic. For example:

- D-ITG API (C) [D-ITG, Distributed Internet Traffic Generator 2015];
- Ostinato API (Python) [Ostinato Network Traffic Generator and Analyzer 2016];
- MoonGen API (Lua) [MoonGen 2019];
- DPDK-Pktgen scripting interface (Lua) [Getting Started with Pktgen 2015].

[1] http://www.icir.org/models/trafficgenerators.html

[2] For example: D-ITG [Botta *et al.* 2012] and Iperf [iPerf - The network bandwidth measurement tool 2019] uses the GNU Socket API [Sockets 2019], Ostinato [Ostinato Network Traffic Generator and Analyzer 2016] uses libpcap [Tcpdump & Libpcap 2019], and MoonGen [Emmerich *et al.* 2015] uses DPDK [DPDK – Data Plane Development Kit 2019].

There are many taxonomies for traffic generators available in the literature. Classify traffic generators is usually "blur" process since packet generators feature many times fall into more than one class. We present two taxonomies:

- Traffic generation strategy;
- Traffic generator implementation.

2.1.1 Strategy

Traffic generators can be classified into two main groups: replay engines [Varet 2014] and model-based tools:

- **Replay engines**: These tools can read *pcap* files, and inject copies of the packet on a network interface. Eg.: TCPReplay [Tcpreplay home 2019], TCPivo [Feng *et al.* 2003], D-ITG [Botta *et al.* 2012].
- **Model-based traffic generators**: they generate synthetic traffic, controlling one or more feature of the traffic; such as header fields, packet sizes and inter-packet times.

Model-based traffic generators can be sub-classified based on the abstraction layer the model operates. We follow here the taxonomy presented by Botta et al. [Botta *et al.* 2010]. Figure 2 shows these traffic generators organized in a layer diagram.

- **Application-level traffic generators**: they try to emulate the behavior of network applications, simulating real workloads stochastically or responsively[3]. As an example, we have Surge, which mimics the communication between clients and web servers;
- **Flow-level traffic generators**: they can reproduce flow characteristics, such as flow duration, start times distributions, and temporal traffic volumes. Harpoon can extract these parameters from Cisco NetFlow data, collected from routers;
- **Packet-level traffic generators**: it is the most used traffic generators. They can control packet-features like inter-departure times, packet size, throughput and packets per second. For example, D-ITG [Botta *et al.* 2012], and TG [Traffic Generator 2011] can control inter-packet times via stochastic distributions. However, most of them only permit the configuration of constant-rate models, by setting the packet rate or the traffic bandwidth, such as Iperf [iPerf - The network bandwidth measurement tool 2019], BRUNO [Antichi *et al.* 2008], and Ostinato [Ostinato Network Traffic Generator and Analyzer 2016].

[3] **Responsiveness** refers to the ability of responding the changes in real-time. In the traffic generator's context, it refers to the ability of changing inner parameters depending on patterns of arriving packets.

- **Multi-level traffic generators**: this is a more recent class of network traffic generator. They take into account existing interaction among each layer of the network stack, to create network traffic as close as possible to reality. The most important tool is Swing [Vishwanath e Vahdat 2009] which input collected *pcap* files.

We have done an extensive survey on packet generators available on the open-source community and classified them according to the first taxonomy. Also, we summarized the main features of each one. The result of this work is the Tables 14, 15, 16, and 17, in the Appendix C. We also have a list of the tool repositories at Table 18.

2.1.2 Implementation

- **Software-only traffic generators**: Implementations of traffic generators utterly independent of its running hardware platform. This implementation comprehends most of traffic generator tools, including all previously mentioned.
- **Software and hardware-dependent traffic generators**: are traffic generators implemented in software, but dependent on the underlying hardware. The most preeminent examples of this class used DPDK [DPDK – Data Plane Development Kit 2019] as packet-generator API. DPDK works directly on the NIC interface, avoiding Operational Systems overheads. As cited on its official website, this approach permits huge precision. As examples we have MoonGen [Emmerich *et al.* 2015] and DPDK-PktGen [Getting Started with Pktgen 2015]
- **Hardware traffic generators**: these open-source traffic generators are implemented in hardware description language (VHDL/Verilog), and work on NetFPGAs. Some examples of implementations are PacketGenerator [Covington *et al.* 2009], Caliper [Ghobadi *et al.* 2012], and OSNT Packet Generator [Antichi *et al.* 2014].

2.2 Realistic Traffic and Traffic Modeling

2.2.1 Realistic Network Traffic Generation

As presented, there is a considerable amount of open-source traffic generators available, each one of them with many different sets of features available. However, on the generation of realistic workload, the set of possibilities becomes much more restricted. On the other hand, there are many works on characterization, modeling, and simulation of different types of network workload. As stated by Botta et al. [Botta *et al.* 2012], a synthetic network traffic generation over real networks should be able to:

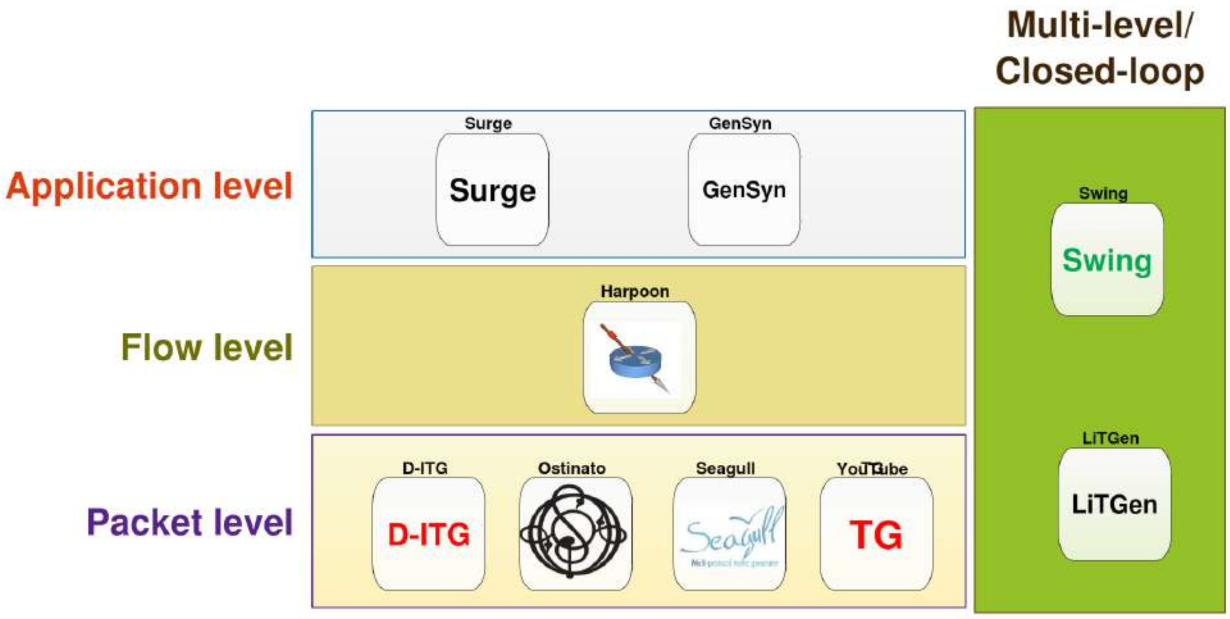


Figure 2 – Diagram representing different traffic generators, according to its abstraction layer.

1. Capture real traces complexity over different scenarios;
2. Be able to custom change some specific properties of the generated traffic ;
3. Return measure indicators of performance experienced by the workload.

As we have found out over the literature in our research, the measure of realism of a traffic generator is given by how well a traffic generator can represent features at the level its model works. For example:

- **Swing**: Vishwanath and Vahdat [Vishwanath e Vahdat 2009] validate their work against packet, flow, and application level features;
- **Harpoon**: Sommers and Barford [Sommers e Barford 2004] validate harpoon on flow-level features;
- **D-ITG**: Botta et al. [Botta *et al.* 2012] validate their work against application-level and packet-level features;
- **sourcesOnOff**: Varet and Larrieu validate sourcesOnOff on packet-level features [Varet 2014].

Therefore, we defined a realistic traffic generator as follows:

Realistic Traffic Generator

A realistic traffic generator is a tool that its model can reproduce real traces complexity and behavior, at the same level of abstraction its traffic model works: on the packet, flow, application or multi-level. In other words, the validation techniques must give similar results to the real and synthetic traces.

We are going to discuss metrics on validation of traffic generators in the next section. The rest of this section will highlight topics on network traffic modeling. We are not going to discuss application modeling, since each one may have their specific behavior. We are going to discuss points that apply to any traffic in general:

- Inter-packet times (throughput) modeling;
- Packet-sizes modeling;
- Packet-header fields;
- Flow modeling;
- Closed-loop behavior modeling.

2.2.2 Inter-packet times (throughput) modeling

Classical models for network traffic generation were the same used in telephone traffic, such as Poisson or Poisson-related. They can describe the randomness of an Ethernet link but cannot capture the presence of "burstiness" in a long-term time scale, such as traffic "spikes" on long-range "ripples". Lerand et al. [Leland *et al.* 1994], points in his seminal work, in 1994, that the nature of the Ethernet traffic is self-similar. It has a fractal-like shape since characteristics seen in a small time scale should appear on a long-scale as well, that have been referred, in the most of the time, as long-range dependence or degree of long-range dependence (LRD). One way to identify if a process is self-similar is by checking its Hurst parameter, or Hurst exponent H, as a measure of the "burstiness" and LRD. A random process is self-similar and LRD if $0.5 < H < 1$ [Rongcai e Shuo 2010] (Appendix A).

Willinger et al. pointed out that the Ethernet traffic has a high variability (or infinite variance) [Willinger *et al.* 1997]. Processes with such characteristic are said to be heavy-tailed. In practical terms, that means a sudden discontinuous change can always occur. To be heavy-tailed means that the stochastic distribution is not exponentially bounded. In other words, some value far from the mean does not have a negligible probability of occurrence. We can express self-similar and heavy-tailed processes using heavy-tailed stochastic distributions, such

Table 2 – Probability density function (PDF) and Cumulative distribution function (CDF) of some random variables, and if this stochastic distribution has or not self-similarity property. Some functions used to express these distributions are defined at the Table 3

Distribution	PDF Equation	CDF Equation	Parameters	Heavy-tailed
Poisson	$f[k] = \frac{e^{-\lambda}\lambda^k}{k!}$	$F[k] = \frac{\Gamma(\lfloor k+1 \rfloor, \lambda)}{\lfloor k \rfloor !}$	$\lambda > 0$ (mean, variance)	no
Binomial	$f[k] = \binom{n}{k} p^k (1-p)^{n-k}$	$F[k] = I_{1-p}(n-k, 1+k)$	$n > 0$ (trials) $p > 0$ (success)	no
Normal	$f(t) = \frac{1}{\sqrt{2\sigma^2\pi}} e^{\frac{(t-\mu)^2}{2\sigma^2}}$	$F(t) = \frac{1}{2}[1 + \text{erf}(\frac{t-\mu}{\sigma\sqrt{2}})]$	μ (mean) $\sigma > 0$ (std.dev)	no
Exponential	$f(t) = \begin{cases} \lambda e^{-\lambda t}; & t \geq 0 \\ 0; & t < 0 \end{cases}$	$F(t) = 1 - e^{-\lambda t}$	$\lambda > 0$ (rate)	no
Pareto	$f(t) = \begin{cases} \frac{\alpha t_m^\alpha}{t^{\alpha+1}}; & t \geq t_m \\ 0; & t < t_m \end{cases}$	$F(t) = \begin{cases} 1 - (\frac{t_m}{t})^\alpha; & t \geq t_m \\ 0; & t < t_m \end{cases}$	$\alpha > 0$ (shape) $t_m > 0$ (scale)	yes
Cauchy	$f(t) = \frac{1}{\pi\gamma}[\frac{\gamma^2}{(t-t_0)^2+\gamma^2}]$	$F(t) = \frac{1}{\pi}\arctan(\frac{t-t_0}{\gamma}) + \frac{1}{2}$	$\gamma > 0$ (scale) $t_0 > 0$ (location)	yes
Weibull	$f(t) = \begin{cases} \frac{\alpha}{\beta^\alpha} t^{\alpha-1} e^{(t/\beta)^\alpha}; & t \geq 0 \\ 0; & t < 0 \end{cases}$	$F(t) = \begin{cases} 1 - e^{-(t/\beta)^\alpha}; & t \geq 0 \\ 0; & t < 0 \end{cases}$	$\alpha > 0$ (shape) $\beta > 0$ (scale)	yes
Gamma	$f(t) = \frac{\beta^\alpha}{\Gamma(\alpha)} t^{\alpha-1} e^{-\beta t}$	$F(t) = 1 - \frac{1}{\Gamma(\alpha)}\Gamma(\alpha, \beta x)$	$\alpha > 0$ (shape) $\beta > 0$ (rate)	no
Beta	$f(t) = \frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha,\beta)}$	$F(t) = I_x(\alpha, \beta)$	$\alpha > 0$ (shape) $\beta > 0$ (shape)	no
Log-normal	$f(t) = \frac{1}{t\sigma\sqrt{2\pi}} e^{-\frac{(\ln(x)-\mu)^2}{2\sigma^2}}$	$F(t) = \frac{1}{2} + \frac{1}{2}\text{erf}[\frac{\ln(x)-\mu}{\sqrt{2}\sigma}]$	μ (location) $\sigma > 0$ (shape)	yes
Chi-squared	$f(t) = \frac{1}{2^{\frac{k}{2}}\Gamma(\frac{k}{2})} t^{\frac{k}{2}-1} e^{-\frac{t}{2}}$	$F(t) = \frac{1}{\Gamma(\frac{k}{2})}\gamma(\frac{k}{2}, \frac{x}{2})$	$k \in \mathbb{N}_{>0}$	no

as Pareto and Weibull. Table 2 shows the reference for these stochastic distributions. In the last column, we indicate if the distribution is or not heavy-tailed.

These concepts of High variability and Self-similarity are called Noah and Joseph Effects [Willinger *et al.* 1997]. Willinger et al. point that the superposition of many ON/OFF sources (or packet trains) using ON and OFF times that obey the Noah Effect (heavy-tailed probabilistic functions), also obey the Joseph effect. That means, it is a self-similar process and can be used to describe Ethernet traffic. Some works on the literature on synthetic traffic uses this principle, like sourcesOnOff [Varet 2014], or have to heavy-tailed processes, such as like D-ITG. Furthermore, some later studies advocate the use of more advanced multiscaling models (multifractal), addressed by investigations that uses envelope processes [Melo e Fonseca 2005].

Table 3 – Definitions of some functions used by PDFs and CDFs

Function	Definition
Regularized Incomplete beta function	$I_x(a,b) = \frac{B(x\|a,b)}{B(a,b)}$
Incomplete beta function	$B(x\|a,b) = \int_0^x t^{a-1}(1-t)^{(b-1)}\mathrm{d}t$
Beta function	$B(x\|a,b) = \int_0^1 t^{a-1}(1-t)^{(b-1)}\mathrm{d}t$
Error function	$\mathrm{erf}(x) = \frac{1}{\sqrt{\pi}}\int_x^{-x} e^{-t^2}\mathrm{d}t$
Lower incomplete Gamma function	$\gamma(s,x) = x^s\Gamma(s)e^{-x}\sum_{k=0}^{\infty}\frac{x^k}{\Gamma(s+k+1)}$

Table 4 – Two different studies evaluating the impact of packet size on the throughput. Both compare many available open-source tools on different testbeds. In all cases, small packet sizes penalize the throughput. Bigger packet sizes achieve a higher throughput.

Article and setup	Traffic Generators		
	Tool	Maximum bit-rate at small packet sizes	Maximum bit-rate at big packet sizes
Article: Comparative study of various Traffic Generator Tools [Srivastava *et al.* 2014] ; *Setup*: Linux (Centos 6.2, Kernel version 2.6.32), Inter(R) Xeon(R) CPU with 2.96GHz, RAM of 64GB , NIC Mellanox Technologies MT25418 [ConnectXVPI PCIe 2.0 2.5GT/s - IB DDR] 10 Bbps. *Protocol*: TCP	PackETH	150 @(64 bytes)	1745 @(1408 bytes)
	Ostinato	135 @(64 bytes)	2850 @(1408 bytes)
	D-ITG	62 @(64 bytes)	1950 @(1408 bytes), 9808 @(1460 bytes, 12 threads)
	Iperf	*	8450 @(1460 bytes, 12 threads)
Article: Performance Monitoring of Various Network Traffic Generators [Kolahi *et al.* 2011]; *Setup*: Intel (R) Pentium 4(R), CPU with 3.0GHz, RAM 1GB, NIC Intel Pro/100 Adapter (100Mbps), Hard Drivers Seagate Barracuda 7200 series with 20BG. *Protocol*:TCP	Iperf	46.0 @(128 bytes)	93.1 @(1408 bytes)
	Netperf	46.0 @(128 bytes)	89.9 @(1408 bytes)
	D-ITG	38.1 @(128 bytes)	83.1 @(1408 bytes)
	IP Traffic	61.0 @(128 bytes)	76.7 @(1408 bytes)

2.2.3 Packet-sizes modeling

The literature shows that the packet size of a trace may result in a considerable impact in a trace throughput since small packets cause significant overhead on packet processing [Rongcai e Shuo 2010] [Kolahi *et al.* 2011]. Table 4 summarizes the results from two different works about throughput impact of packet sizes. On packet size distributions' characterization, we can find many works as well. For example, Castro et al. pointed that 90% of UDP packets were smaller than 500 bytes, and most packets transmitted using TCP have 40 bytes (acknowledgment) and 1500 bytes (Maximum Transmission Unit, MTU) [Castro *et al.* 2010]. Ostrowsky et al. found that on UDP traces, the modes of two regions were 120 and 1350 bytes, with a cut-off value of 750 bytes. They also found that roughly UDP packets constituted 20% of the total number of packets on captures [Ostrowsky *et al.* 2007]. Castro et al. points on his work that captured traces on routers were all bimodal, and the majority is TCP. However, the size of each mode may change depending on the application. For example, an HTTP traffic tends to have a mode closer to the MTU compared to an FTP capture [Castro *et al.* 2010].

2.2.4 Packet-header fields

Accurate replication of network traffic should be able to control packet headers such as protocols, ports, addresses, and so on. Traffic generators provide support for these features, more frequently in a limited way. Most offer support just standard protocols, such as TCP, UDP, and IPv4. On the other hands, there are some which provide a vast variety of support and control over packet headers like PackETH [PACKETH 2015] and D-ITG [D-ITG, Distributed Internet Traffic Generator 2015]. Other tools are even able to enable someone to extend this feature and develop support to new protocols. For example, Ostinato and Seagull permit the customization and creation of protocols [Seagull – Open Source tool for IMS testing 2006].

2.2.5 Flow modeling

Some packet-level traffic generators permit the control of flow generation, mostly manually through an API or scripting. In terms of automatic flow configuration, an example is Harpoon [Sommers *et al.* 2004], which can to automatically configure its flows, using as input NetFlow Cisco traffic traces to automatically setting parameters. Harpoon deals with flow modeling in three different levels: file level, session level, and user level, not dealing with packet level at all. In the file level, Harpoon model two parameters: the files size and the time interval between consecutive file requests, called inter-file request time. The middle level is the session level, that consist of sequences files transfer between two distinct IP addresses. The session level has three components: the IP spatial distribution, the second is the inter-session start times and the third is the session duration. The last level is the user level. In Harpoon, "users" are divided on "TCP" and "UDP" users, which conduct consecutive session using these protocols. This level has two components: the user ON time, and the number of active users. By modeling the number of users, Harpoon can reproduce temporal (diurnal) traffic volumes.

2.2.6 Closed-loop (responsive) models

The closed-loop operation means that the traffic generator uses feedback to reconfigure its model. That means the traffic generator can change its behavior at run time according to the observation made in real-time, changing the traffic created. These modifications involve changes on parameters of statistical distributions of inter-departure time and packet size, for example. Swing [Vishwanath e Vahdat 2009] and application-level traffic generators like Surge [Barford e Crovella 1998] and GenSyn [Heegaard 2000] uses this strategy.

2.3 Validation of Traffic Generator Tools

After the implementation of a traffic generator, it needs to be validated. Thus, we need a set of proof of concepts to evaluate if it reached its purposes or not. Researchers have been proposed many validation techniques, according to the traffic generator intended behavior. Magyesi and Szabó [Molnár *et al.* 2013] presented a survey of these techniques, grouped by type of metric. The authors classified the techniques into four categories: packet based metrics, flow-based metrics, scaling characteristics, and QoS/QoE related metrics. Here we present a short review of each group of these validation techniques.

2.3.1 Packet Based Metrics

Packet-based metrics are the most used metrics in the validation of traffic generators [Molnár *et al.* 2013]. The most relevant packet based metrics are throughput [Botta *et al.* 2010] [Srivastava *et al.* 2014] [Kolahi *et al.* 2011] [Emmerich *et al.* 2015] (bytes and packets), packet size distribution [Castro *et al.* 2010] and inter-packet time distribution (inter-arrival and inter-departure) [Varet 2014] [Botta *et al.* 2012].

2.3.2 Flow Based Metrics

Since SDN device, which execute flow-based operations are becoming widespread, flow-based metrics are becoming increasingly relevant [Molnár *et al.* 2013] [Kreutz *et al.* 2015]. Magyesi and Szabó [Molnár *et al.* 2013] consider the essential flow metrics, the flow size distribution, and volume. The flow volume stands for the number of flows of traffic. The flow size distribution is a measure of the time-length of the flows in network traffic. The flow volume corresponds to the number of flows that a device must process simultaneously. Moreover, the flow sizes define how much time each of these instances will run.

2.3.3 Fractal and Scaling Characteristics

Hurst Exponent

Second order characteristics such as "burstiness" and long-range dependence are responsible for the complex nature of internet traffic [Molnár *et al.* 2013]. Due to its non-stationary nature, traditional methods fail to extract useful information [Molnár *et al.* 2013]. The first analysis found in the literature to extract fractal characteristics was made by Lerand et al. estimating the Hurst exponent [Leland *et al.* 1994]. They demonstrated the self-similar nature of the Ethernet traffic. As explained in the previous section, a self-similar process should have a Hurst exponent valued larger than 0.5 and smaller than 1 ($0.5 < H < 1$).

Wavelet-based Analysis

Over the years, wavelet-based analysis has become an efficient way to reveal correlations, bursts and scaling nature of the Ethernet traffic [Molnár *et al.* 2013]. Many papers have used wavelet-based analysisciteswing-paper [Huang *et al.* 2001] [Abry e Veitch 1998]. Huang et al. [Huang *et al.* 2001] and Abry and Veitch [Abry e Veitch 1998] offer an extensible explanation of wavelet-based scaling analysis (WSA) or wavelet multi-resolution energy analysis (WMA).

We will now make a brief derivation of the energy curves used in wavelet analysis (WMA). First, consider a time series $X_{0,k}$ for $k = 0, 1, ...2^n$:

$$\{X_{0,k}\} = \{X_{0,0}, X_{0,1}, ..., X_{0,2^n}\} \tag{2.1}$$

Then, we roughly approximate X_0 in another time-series X_1 with half of the original resolution, but using$\sqrt{2}$ as normalization factor:

$$X_{1,k} = \frac{1}{\sqrt{2}}(X_{0,2k} + X_{0,2k+1}) \tag{2.2}$$

Taking the differences, instead of the averages (equation 2.2), evaluate the so-called *details*.

$$d_{1,k} = \frac{1}{\sqrt{2}}(X_{0,2k} - X_{0,2k+1}) \tag{2.3}$$

Continuing this process respectively , writing coarser time series X_2 from X_1, until we reach X_n. Therefore, we will get a collection of *details*:

$$\{d_{j,k}\} = \{d_{1,0}, d_{1,1}, ..., d_{1,2^{n/2}}, ..., d_{n,0}\} \tag{2.4}$$

This collection of details $d_{j,k}$ are called Discrete Haar Wavelet Transform. Using the *details*, we can calculate the energy function E_j, under the scale j, using:

$$E_j = \frac{1}{N_j} \sum_{k=0}^{N_j - 1} |d_{j,k}|^2; \qquad j = 1, 2, ..., n \tag{2.5}$$

were N_j is the total number of coefficients at scale j. Plotting a graph of $\log(E_j)$ as a function o the scale j, we will obtain a wavelet multiresolution energy curve.

On energy wavelet multiresolution energy curves, it is possible to capture three central behaviors, according to the time-scale. On **periodic time series**, the Energy values will be small. Time-series with no error in the time-periods of the scales j, the energy values E_j will be zero. So periodicity will be sensed if the value of the energy function decrease. Perfect **white noise time series** maintains the same value of the energy function. So an approximately constant value for the energy function E_j indicates white noise behavior (which can be represented by a Poisson process [Grigoriu 2004]). On **self-similar time series**, the energy function $\log(E_j)$ grows approximately linearly with the scale j. Following these rules, we can quickly identify periodicities, and self-similar and Poisson process characteristics, just seeing if it decays, growths, or constant-shapes on wavelet energy plots.

Later studies suggested the use of multi-fractal models, instead of the self-similar models (also called monofractal) [Molnár *et al.* 2013] [Ostrowsky *et al.* 2007]. Since there is a lack of multiscaling analysis in validation of traffic generation in the literature, this type of analysis will stay for future works.

QQplots Analysis

Another way to analyze scaling characteristics is through *QQplots* (Appendix A) [WILK e GNANADESIKAN 1968] [Understanding Q-Q Plots | University of Virginia Library Research Data Services + Sciences 2019] [Q–Q plot 2019]. *QQplot* is a visual method to compare sample data with a specific stochastic distribution. To create a *QQplot*, we must order the orders the measured data values (**samples**) from the smallest to largest. Then, we have to plots the samples against the expected value given by the model we want to validate. The **sample** values appear along the **y-axis** and the **theoretical** values along the **x-axis**, as we represent on the Figure 3. Finally, we draw a line with 45 degrees of inclination, representing how the samples would behave if they had the same behavior of the theoretical model. The more linear, the more the data is likely to be expressed by this specific stochastic distribution. Also, Depending on how the curve be behaves, some features of the empirical dataset compared to the theoretical can be observed, such as **heavy-tail**, **light-tail**, **bimodal** behavior, and the **curve skew** (Figure 3). We present a complete tutorial about this subject in the Appendix A.

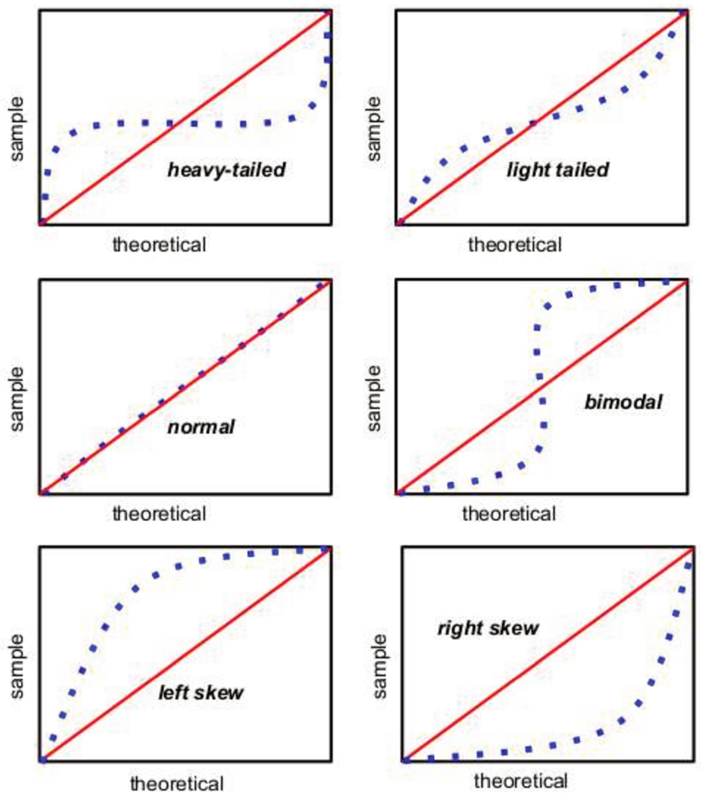


Figure 3 – How information about may be extracted from QQplots.

2.3.4 QoS/QoE Related Metrics

For the point of view of traffic generation, QoS and QoE metrics should present similar values to the ones found in real scenarios. As stated by Magyesi and Szabó [Molnár *et al.* 2013], important QoS/QoE metrics on validation of workload tools are Round trip Time values (RTT), average queue waiting time and, queue size. Still, on queue size, self-similar traffic consumes router buffers faster than Poisson traffic [Cevizci *et al.* 2006].

2.4 Conclusions

In this chapter, we discussed some fundamental concepts of our research: network traffic generators, network traffic modeling, and network traffic generators validation. In section 2.1, we surveyed types of traffic generators and a comparison between their considerable variability of features. It helped us to summarize and have an understanding of what is available nowadays for use, and define the gaps. Also, this chapter helped us to identify what tools and frameworks are available to use. Section 2.2 showed a brief overview of efforts on network traffic modeling and realistic traffic generation. In the modeling issue, was presented a short historical summary of some critical points of network traffic modeling, and on practical traffic generation, discussing some reference tools.

3 SIMITAR: Architecture and Methodology

In this Chapter, we will present our tool which aims to meet the objectives identified in Chapter 1. *SIMITAR*[1] is an acronym for ***SnIffing*[2]*, ModellIng, and TrAffic geneRation***. This acronym summarizes its main fenctional features. SIMITAR is a traffic generator able to ***learn*** features of real traffic automatically, and reproduce synthetic traffic similar to the original. It generates a model for the traffic in an XML[3] file we call the *Compact Trace Descriptor* (CTD). As input data, SIMITAR can use *pcap* files or real-time captures.

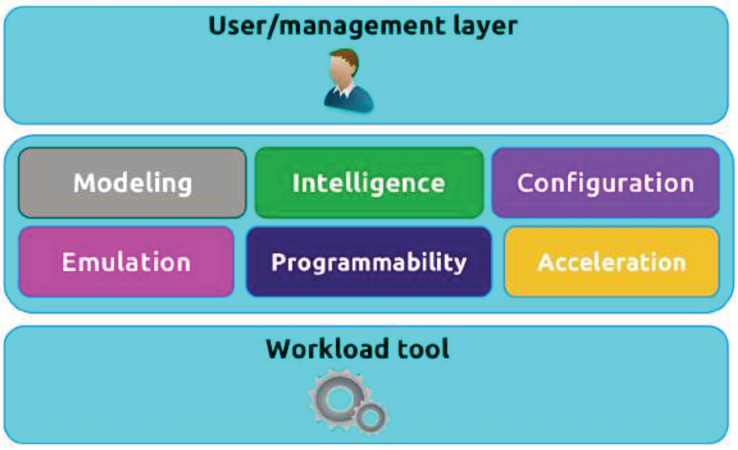


Figure 4 – Main architectural concept of SIMITAR: a tool to automatize many tasks on traffic modelling and generation.

Figure 4 abstracts the stated concepts in a layer model diagram. Our tool works as an intermediate layer which offers traffic ***modeling***, ***configuration***, ***emulation***, and ***programmability***. In the Figure, we also include packet acceleration[4], which is not implemented yet but discussed as future work, in Chapter 6. We are going to refer to the underlying workload tool as the packet generator engine and it is used via API by SIMITAR.

We also introduce the concept of *programmability*. The user may create custom traffic, creating the *Compact Trace Descriptor*, following its template. The idea is that he or she can create custom traffic in a platform agnostic way, without having to study any documentation, and implement any script or program. Using a component methodology, we uncouple the packet

[1] A **Scimitar** or **Scymitar** is curved sword, originating in the Middle East [Burton 2014].
[2] **Sniffing** is the operation of Sniffers. **Sniffers** are tools able to intercept and analyze packets passing over the network. Are also called packet analyzers [Garcia 2008].
[3] Extensible Markup Language (XML) is a markup language that defines rules for storing and processing hierarchical data [W3C 2019].
[4] **Packet acceleration** is a concept introduced by DPDK [DPDK – Data Plane Development Kit 2019], which means kernel by-pass. Packet acceleration optimizes the packet processing, and therefore traffic generation, enabling higher throughput rates.

generation, from the data collection and parameterization process. We developed it using the factory design pattern[5] to make the extension easy for any packet generator engine.

We abstract the whole operation cycle in Figure 5. Our tool collects packet data from live captures or *pcap* files. It then breaks down the traffic into flows and uses the data to generate parameters for our traffic model. Finally, SIMITAR provides these parameters to a packet generator engine and controls the packet injection.

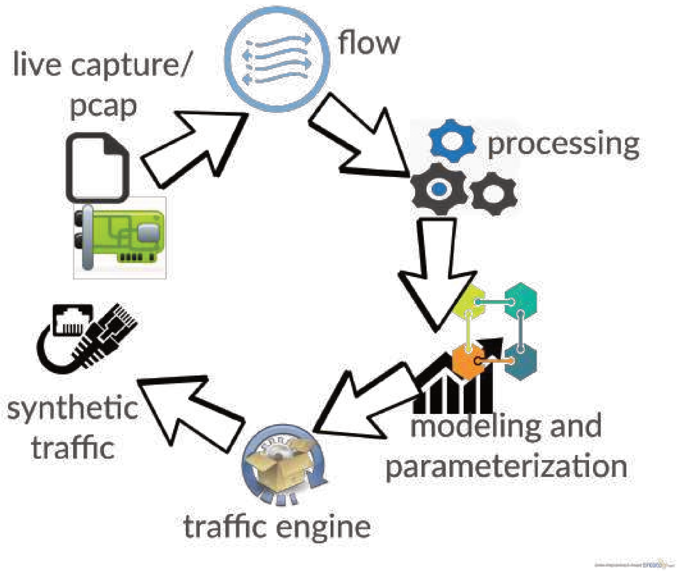


Figure 5 – an operation cycle of SIMITAR, emphasizing each main step: sniffing, flow classification, data storing, data processing and fitting, model parameterization, and synthetic traffic generation.

3.1 System Overview

The SIMITAR architecture is shown in Figure 6. It is composed of four components: a ***Sniffer***, an ***SQLite database***, a ***Trace Analyzer***, a ***Flow Generator***. In the following, We describe each of them.

3.2 *Sniffer*

A sniffer is a tool that can intercept and analyze internet packets from a given network interface. Our *Sniffer* component collects network traffic data and classifies it into flows, storing it on an SQLite database. The flow classification is based on the match of the information on the header fields. It uses the same tracks used by SDN switches [Kreutz *et al.* 2015]. The data used on the matching are the following:

[5] **Design patterns** are abstractions that aim to help the implementation and systems structuring [C++ Programming: Code patterns design - Wikibooks, open books for an open world 2019].

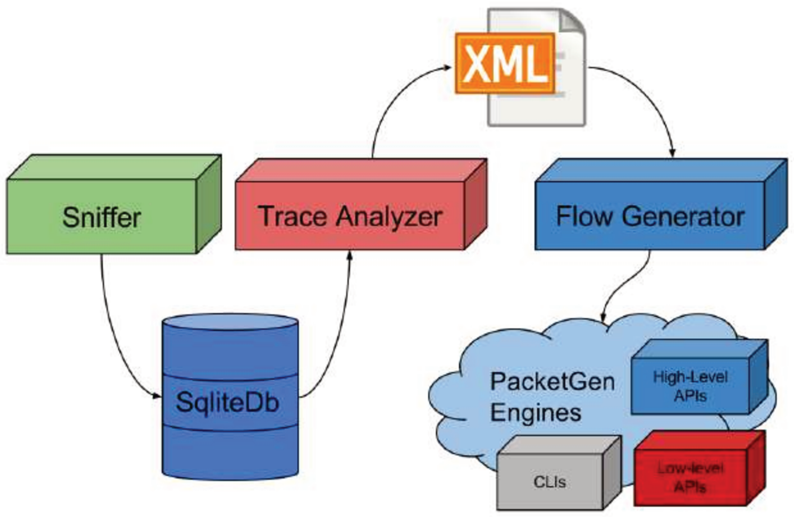


Figure 6 – Architecture of SIMITAR

- Link Protocol
- Network Protocol
- Network Source and Destination Address
- Transport Protocol
- Transport Source and Destination Port

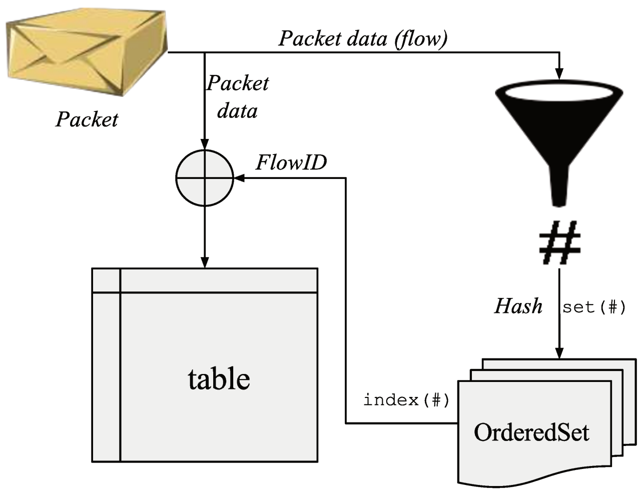


Figure 7 – SIMITAR's sniffer hash-based flow classification

We implemented the first version of this component in Shell Script (Bash). Tshark[6] [tshark - The Wireshark Network Analyzer 3.0.1 2019] was used to extract header fields, and Awk to match the flows, and Sed/Awk to create the SQLite queries. This version was too slow

6 **Tshark** is a command-line dump and analyze network traffic tool.

to operate in real time on ethernet interfaces. On the other hand, this approach was fast to implement and enabled the implementation of the other components.

The second and current version is in Python, and used Pyshark [pyshark · PyPI 2019] as a sniffer library. The *Sniffer* has a data structure we developed called *OrderedSet*. A set is a list of elements with no repetition but does not keep track of the insertion order, but our data structure can keep it. Also, the *OrderedSet* uses a 64 bit hash function from the FNV[7] family. The listed header fields are inputs for a hash function. The hash value is added to the ordered. The operation of insertion returns the insertion position (index of the hash-value on the *OrderedSet*). We use the returned position as flowID.

As future improvements for this component, we propose a more efficient implementation in C++ and data visualization for the collected data. In this way, we can optimize packet processing. We discuss this in more depth in Chapter 6.

3.3 SQLite database

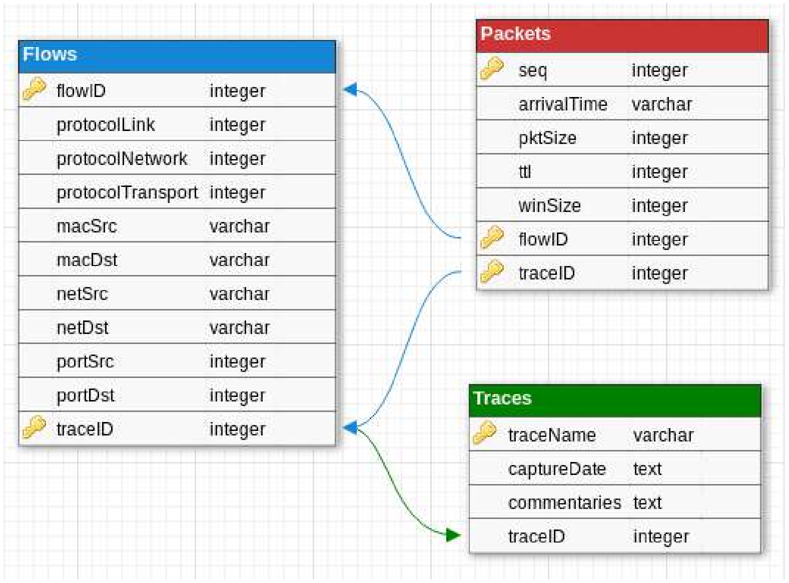
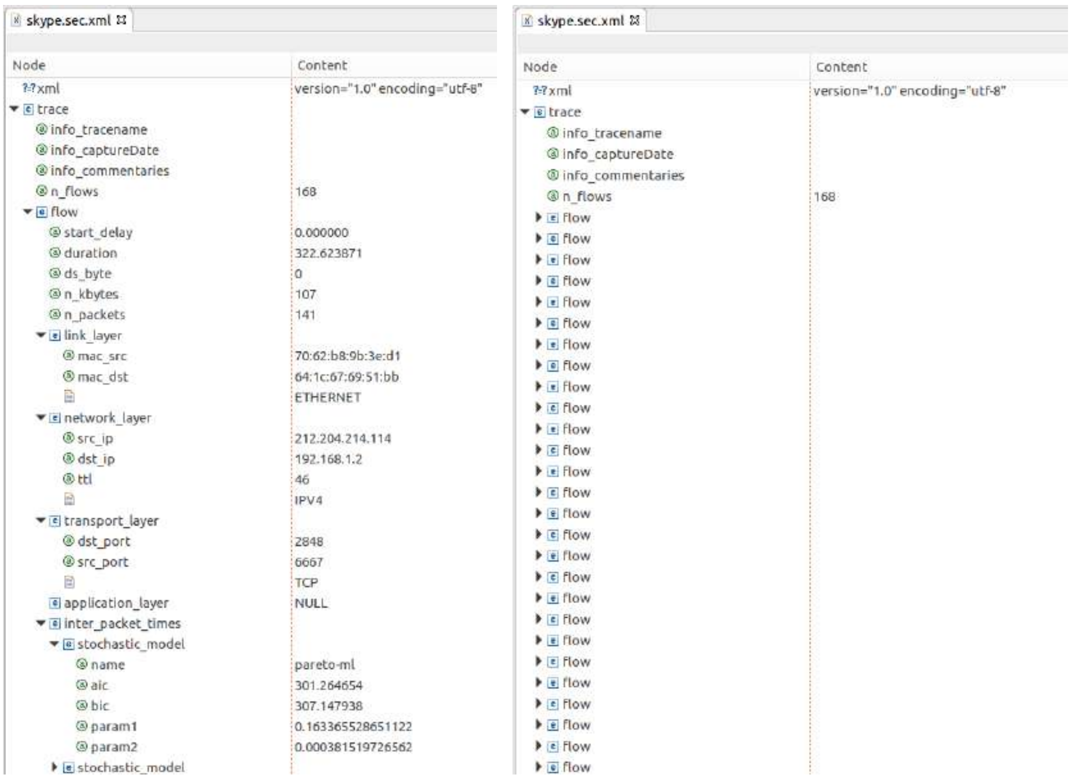


Figure 8 – SIMITAR's SQLite database relational model

The database stores the collected raw data from the traces for further analysis. The *Sniffer* records data on it and the *Trace Analyzer* reads. We choose an SQLite database because its specifications [Appropriate Uses For SQLite 2019] fits our purposes well. It is simple and suitable for an amount of data less than terabytes. In Figure 8, we present the relational model[8] of our database, which contains a set of features extracted from packets, along with the flowID calculated by the *Sniffer* component.

7 The collision probability of a good 64 bits hash function in a table with 10000 items is about of $2.71e-12$.

8 A **Relational Model** is an approach for managing data in a database. Most of relational databases use SQL as a query language [Date 2004].



```
skype.sec.xml
Node                          Content
?-?xml                        version="1.0" encoding="utf-8"
▼ trace
    info_tracename
    info_captureDate
    info_commentaries
    n_flows                   168
  ▼ flow
      start_delay             0.000000
      duration                322.623871
      ds_byte                 0
      n_kbytes                107
      n_packets               141
    ▼ link_layer
        mac_src               70:62:b8:9b:3e:d1
        mac_dst               64:1c:67:69:51:bb
                              ETHERNET
    ▼ network_layer
        src_ip                212.204.214.114
        dst_ip                192.168.1.2
        ttl                   46
                              IPV4
    ▼ transport_layer
        dst_port              2848
        src_port              6667
                              TCP
      application_layer       NULL
    ▼ inter_packet_times
      ▼ stochastic_model
          name                pareto-ml
          aic                 301.264654
          bic                 307.147938
          param1              0.163365528651122
          param2              0.000381519726562
      ▶ stochastic_model
```

(a)

```
skype.sec.xml
Node                          Content
?-?xml                        version="1.0" encoding="utf-8"
▼ trace
    info_tracename
    info_captureDate
    info_commentaries
    n_flows                   168
  ▶ flow
  ▶ flow
  ▶ flow
  ▶ flow
  ▶ flow
  ▶ flow
  ▶ flow
  ▶ flow
  ▶ flow
  ▶ flow
  ▶ flow
  ▶ flow
  ▶ flow
  ▶ flow
  ▶ flow
  ▶ flow
  ▶ flow
  ▶ flow
  ▶ flow
  ▶ flow
  ▶ flow
  ▶ flow
  ▶ flow
  ▶ flow
  ▶ flow
  ▶ flow
  ▶ flow
```

(b)

Figure 9 – Directory diagram of the schema of a Compact Trace Descriptor (CDT) file. On the left, we present a dissected flow, and on the right a set of flows.

3.4 *Trace Analyzer*

This component is the core of our project. It creates a trace model via the analysis of the collected data. The *Trace Analyzer* has the task to learn these features from raw trace data (stored in the SQLite database) and generate an XML file to store a parameterized model. A *Compact Trace Descriptor* (CTD) acts as a human and machine-readable file, which describes a traffic trace through a set of flows, each of them represented by a set of parameters, such as header information and analytical models. In Figure 9 we show a directory diagram of a CDT file. It has many flow fields, and each one contains each estimated parameter . Now we will describe each model part.

3.4.1 Flow features

We measured some flow-features directly from data, namely:

- Flow-level properties like duration of flow, start delay, number of packets per flow, number of KBytes per flow;
- Header fields, like protocols, QoS fields, ports, and addresses.

Each one of these parameters is unique to each flow. Other features like packet-size distribution and inter-packet times follow probability distributions. To represent these characteristics, we used sets of stochastic-based models.

3.4.2 Inter-Packet Times

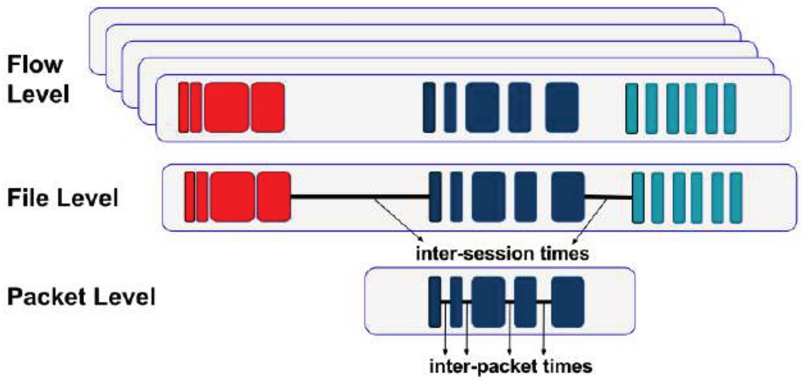


Figure 10 – The schema of the modified version of the Harpoon algorithm we adopt on SIMITAR.

To represent inter-packet times, we adopted a modified version of the Harpoon's traffic model. An in-depth explanation of the original model can be found at [Sommers *et al.* 2004] and [Sommers e Barford 2004]. Harpoon uses a definition of each level, based on the measurement of SYN and ACK TCP flags. It uses TCP flags (SYN) to classify packets at different levels: file, session, and user level. On the other hand, we decide to do not use header fields, but inter-packet times only to make this distinction. In that way, the model is not attached to any protocol.

A graphical representation of our model is in the Figure 10. We defined three different layers of data transference to model and control: *file*, *session*, and *flow*. For SIMITAR, a file is a sequence of consecutive packets transmitted continuously, with inter-packet times small than a threshold. A file can be, for example, packets from a download, a UDP connection or a single ICMP echo packet. A session refers to a sequence of multiple files transmitted between a source and a destination, belonging to the same flow. The flow level refers to the conjunction of flows, as classified by the *Sniffer*. Now, we will explain the SIMITAR operation for each layer.

In the **flow-layer**, the *Trace Analyzer* loads the flow arrival times from the database and calculates the inter-packet times within the flow context; that means as if each flow was independent traffic. At the session layer, we used a deterministic packet train (ON/OFF) model for evaluating file transference times and times between files. We developed an algorithm called `calcOnOff`. It calculates a set of ON (when the flow is transferring packets) and OFF times (when the flow is idle) . It also determines the number of packets and bytes transferred for each file. ON are going to be used as for actual traffic generators on the Flow Generator component, so we defined a minimum acceptable time for on periods equal to 100 ms. ON times can be arbitrary and small, and they could be incompatible with acceptable ON periods for traffic generators. Also in the case of just one packet, the ON time would be zero. So a minimum acceptable time was set to solve these issues. The OFF times, on the other hand, are defined by the constant `session_cut_time`[9]. This is the threshold value mentioned previously wich distinguish files form sessions. If the time between two packets of the same flow is greater than `session_cut_time`, we consider them belonging to a different file, so this time is a session OFF time. In this case, we use the same value of the constant *Request and Response timeout* of Swing [Vishwanath e Vahdat 2009] for the `session_cut_time`: 30 seconds. The *Flow Generator* component[10] is responsible by the control of the ON/OFF periods on the traffic generation.

In the **file-layer**, we modeled the inter-packet times at the file level. We selected all times smaller than `session_cut_time` 9, and all files within the same flow were considered to follow the same model. We delegated the control of the inter-packet times to the underlying packet generator engine. We ordered them, from the best to the worst. Currently, we are using eight different stochastic functions parameterizations. We display each of them in Table 5 .

Table 5 – Functions and parameterizations used by SIMITAR

Function	Linear Regression	Maximum Likelihood	Empirical[11]
Weibull	✓		
Normal			✓
Exponential	✓		✓
Pareto	✓	✓	
Cauchy	✓		
Constant			✓

From the functions presented in the first column in Table 5, Weibull, Pareto, and Cauchy are heavy-tailed (and self-similar processes). However, if the flow has less than 30 packets, just the constant model is evaluated. It is because numerical methods gave poor results if the data sample is small. We sorted these models according to the Akaike Information Criterion (AIC) as default [Varet 2014] [Yang 2005]. This methodology is explained in-depth in Chapter 4 and illustrated in Figure 11. All these constants and modes of operation are modifiable via command-line options.

3.4.3 Packet Sizes

Our approach for the packet size was much simpler. Since the majority of packet size distributions found in real measurements are bi-modal [Castro *et al.* 2010] [Varet 2014]

[9] In the code it is called `DataProcessor::m_session_cut_time`

[10] The class `NetworkFlow` makes this control

[11] Empirical estimation, by calculation of the avarge, and standard deviation

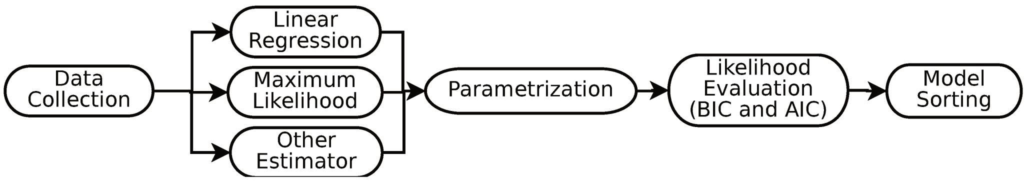


Figure 11 – Diagram of parameterization and model selection for inter-packet times and inter-file times.

[Ostrowsky *et al.* 2007], we first sorted all packet sizes of flow into two modes. We defined a packet-size cut value of 750 bytes, the same value adopted by [Ostrowsky *et al.* 2007].

We knew how many packets each mode has, and then we fitted a model to it. We use three stochastic models: constant, exponential and normal. Since self-similarity does not make sense for packet-sizes, we prefered to use just the simpler models. When there was no packet for a model, we set a flag `NO_MODEL`, and when there was only a single packet, we used the constant model. Then we calculated the *BIC* and *AIC* for each, but we decided to set the constant model as the first.

As is possible to see in many works [Castro *et al.* 2010] [Ostrowsky *et al.* 2007], since the standard deviation of each mode tends to be small, constant fittings give good approximations. Also, it is computationally cheaper for the traffic generated than the other models, since no calculation is needed for each packet sent. Since both *AIC* and *BIC* criteria will always select the constant model as the worst, we decided to ignore this.

3.4.4 *Compact Trace Descriptor*

An example of the final result of all the methods is presented in the XML code below. The code illustrates a single flow from a *Compact Trace Descriptor* (CDT) file. The inter-packet times' models are on tags "`inter_packet_times`" and the packet trains models on tags "`session_times`". All the times are in seconds, and "`inf`" represents infinity. The protocol of each layer is on the data field for each tag.

```
<flow start_delay="0.144400" duration="317.744333" ds_byte="0" n_kbytes="40"
 ↪  n_packets="344">
    <link_layer mac_src="64:1c:67:69:51:bb"
     ↪  mac_dst="70:62:b8:9b:3e:d1">ETHERNET</link_layer>
    <network_layer src_ip="192.168.1.1" dst_ip="192.168.1.2"
     ↪  ttl="64">IPV4</network_layer>
    <transport_layer dst_port="2128" src_port="53">UDP</transport_layer>
    <application_layer>DNS</application_layer>
    <inter_packet_times>
        <stochastic_model name="pareto-ml" aic="-1165.310696" bic="-1157.646931"
         ↪  param1="0.405085202535192" param2="0.002272655895996"/>
        <stochastic_model name="pareto-lr" aic="-454.049749" bic="-446.385984"
         ↪  param1="0.061065000000000" param2="0.002272655895996"/>
```

```
            <stochastic_model name="weibull" aic="-246.882037" bic="-239.218273"
             ↪  param1="0.120355000000000" param2="0.001629000000000"/>
            <stochastic_model name="exponential-me" aic="486.370061" bic="494.033826"
             ↪  param1="1.340057495455104" param2="0.000000000000000"/>
            <stochastic_model name="normal" aic="1629.370900" bic="1637.034665"
             ↪  param1="0.746236637899171" param2="2.626808289821357"/>
            <stochastic_model name="exponential-lr" aic="3166.816047" bic="3174.479812"
             ↪  param1="0.009752000000000" param2="0.000000000000000"/>
            <stochastic_model name="cauchy" aic="31737.418442" bic="31745.082207"
             ↪  param1="0.000000000000194" param2="-3152.827055696396656"/>
            <stochastic_model name="constant" aic="inf" bic="inf"
             ↪  param1="0.746236637899171" param2="0.000000000000000"/>
        </inter_packet_times>
        <session_times on_times="29.22199798,73.40390396,151.84077454"
         ↪  off_times="30.85738373,32.42027283" n_packets="19,103,222"
         ↪  n_bytes="2272,12399,26689"/>
        <packet_sizes n_packets="344" n_kbytes="40">
            <ps_mode1 n_packets="344" n_kbytes="40">
                <stochastic_model name="constant" aic="inf" bic="inf"
                 ↪  param1="120.232558" param2="0.000000"/>
                <stochastic_model name="normal" aic="2926.106952" bic="2933.788235"
                 ↪  param1="120.232558" param2="16.941453"/>
                <stochastic_model name="exponential-me" aic="3987.126362"
                 ↪  bic="3994.807645" param1="0.008317" param2="0.000000"/>
            </ps_mode1>
            <ps_mode2 n_packets="0" n_kbytes="0">
                <stochastic_model name="no-model-selected" aic="inf" bic="inf"
                 ↪  param1="0.000000" param2="0.000000"/>
            </ps_mode2>
        </packet_sizes>
    </flow>
```

3.5 *Flow Generator*

The *Flow Generator* handles the data on the *Compact Trace Descriptor* file, which provides parameters for traffic generation. It crafts and controls each flow in a separate thread. We have already implemented this component using Iperf and Libtins (C++ API) [libtins: packet crafting and sniffing library 2019] as packet generators. It must follow the class hierarchy as presented in Figure 12. This component was designed using the factory design pattern, to simplify its expansion and support[12]

This component itself is a multi-layer workload generator according to the typing

[12] If the user wants to introduce support for a new packet generator engine, he has to implement a derived class of `DummyFlow`, such as in Figure 12. In the current release of SIMITAR, we already have `IperfFlow` and `TinsFlow`, and `DitgFlow`. This new class needs to be static, and the support must be implemented on the factory `NetworkFlowFactory`. For closed loop packet-crafters (the ones that need to establish a connection to generate traffic), two methods must be implemented: `flowGenerate()` and `server()`. `flowGenerate()` is responsible for sending a single file, as defined on de Figure 10. The `server()` methods must implement the reception of n files. For open-loop packet crafters (the ones whose just inject packets but do not establishes a connection), such as the one we implemented using Libtins, does not need the server-side implemented.

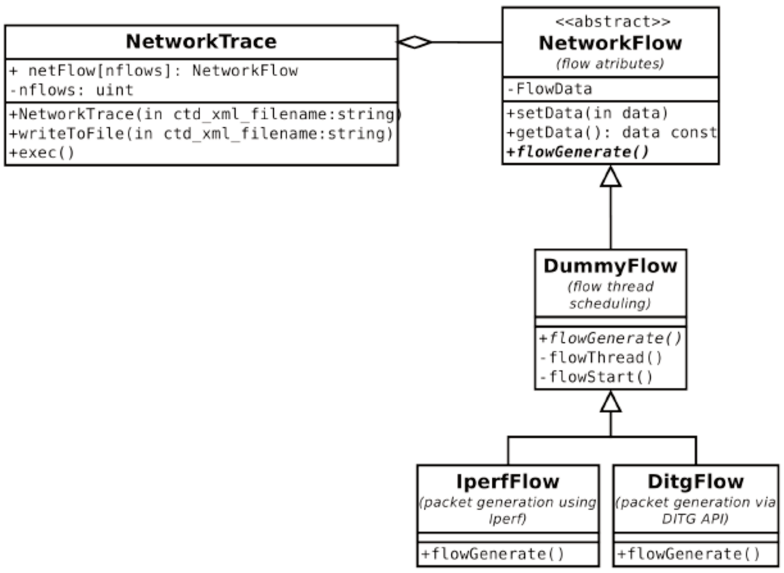


Figure 12 – Class hierarchy of NetworkTrace and NetworkFlow, which enables the abstraction of the traffic generation model of the packet generation engine.

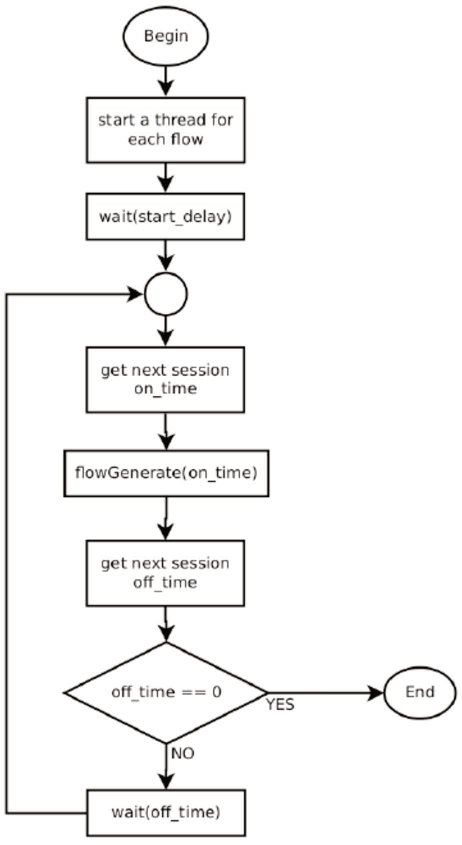


Figure 13 – Simplified-harpoon emission algorithm

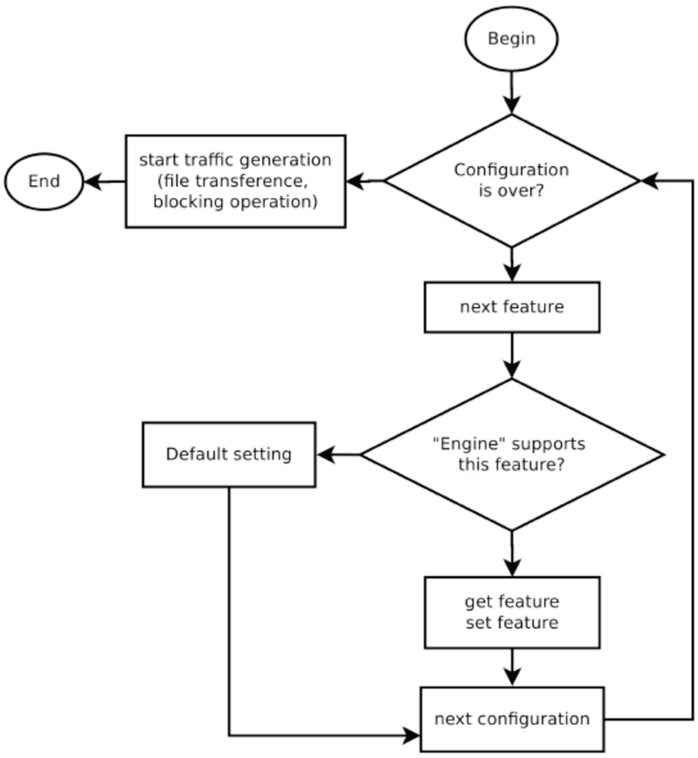


Figure 14 – Packet engine configuration method

introduced in Chapter 2[13]. At the flow-level, SIMITAR controls each flow using the algorithm in Figure 13. This algorithm handles our model as defined in Figure 10. This procedure is independent of the underlying packet crafting tool used. It starts a *thread* for each flow in the *Compact Trace Descriptor*, and then the thread sleeps for `start_delay` seconds. The `start_delay` is the arrival time of the first flow packet. Passed this time, it then calls the underlying packet generator tool (defined as command line argument), and passes to it the flowID, file ON time, number of packets and number of bytes to be sent (file size), and network interface. Then it sleeps until the next session OFF time. When the list of ON/OFF times from the flow is over, the thread ends.

At the packet level, SIMITAR configures the packet-generator tool to send a *file*, as defined in Figure 10. This method must use the available parameters, attributes and *getters* to configure and generate thread-safe traffic using the method `flowGenerate()`. The *file* configuration must follow Figure 14. Down below we present a simple code of how the D-ITG API can be used to generate packet-level traffic. A more complex configuration is possible, but it serves to illustrate the procedure. We call this concept *Flow Generation Programming*. Its API documentation is available at the D-ITG website[14].

```
// This implementation is just a simplified version to illustrate the procedure.
void DitgFlow::flowGenerate(const counter& flowId, const time_sec& onTime, const uint&
 ↪  npackets, const uint& nbytes,  const string& netInterface)
```

[13] Since it works both at packet and flow level, but does not work at application level yet.
[14] http://www.grid.unina.it/software/ITG/manual/index.html#SECTION00047000000000000000

```
{
    // create command to generate the traffic
    std::string strCommand;
    std::string localhost = getNetworkSrcAddr();
    strCommand += " -t " + std::to_string(onTime);
    strCommand += " -k " + std::to_string(nbytes / 1024);
    strCommand += " -a " + getNetworkDstAddr();

    // configure protocol
    if (this->getTransportProtocol() == PROTOCOL__TCP)
        strCommand += " -T TCP -D ";
    else if (this->getTransportProtocol() == PROTOCOL__UDP)
        strCommand += " -T UDP ";
    else if (this->getTransportProtocol() == PROTOCOL__ICMP)
        strCommand += " -T ICMP ";

    //configure inter-packet time model, just Weibull or Constant
    StochasticModelFit idtModel;
    for(uint i = 0;;i++)
    {
        idtModel = this->getInterDepertureTimeModel(i);
        if(idtModel.modelName() == WEIBULL)
        {
            strCommand += " -W " + std::to_string(idtModel.param1()) + " " +
             ↪   std::to_string(idtModel.param2());
            break;
        }
        else if ( idtModel.modelName() == CONSTANT)
        {
            strCommand += " -C " + std::to_string(nbytes/(1024*onTime));
            break;
        }
    }

    // it uses C strings as arguments
    // it is not blocking, so it must block until finishes
    int rc = DITGsend(localhost.c_str(), command.c_str()); // D-ITG API
    usleep(onTime*10e6); // D-ITG uses miliseconds as time unity
    if (rc != 0)
    {
        PLOG_ERROR << "DITGsend() return value was" << rc ; // our log macro for erros
        exit(EXIT_FAILURE);
    }
}
```

3.6 Network Packet Generator

A network packet generator is a tool or library that should provide its API or script interface for the *Flow Generator* component. With this engine, the user must be able to send packets and control attributes such as sending time, bandwidth, number of packets, protocols, and so on. This means, any available parameter form the *Compact Trace Descriptor*.

3.7 Usability

SIMITAR is composed of three main command-line applications, whose give command line access to the *Sniffer*, *Trace Analyzer* and *Flow Generator*, respectively:

- `sniffer-cli.py`;
- `trace-analyzer`;
- `simitar-gen` (the actual traffic generator).

Below we show some usage commands of SIMITAR's components. The `sniffer-cli.py` application creates a new trace entry on the database using the command option new. Then the *Trace Analyzer* can create a *Compact Trace Descriptor* using the same trace entry. We can change the constants used by the *Trace Analyzer* by the command line option. As a traffic generator (`simitar-gen`), SIMITAR may work as a client or a server. Working as a server is necessary for closed-loop packet-generator engines; tools that require establishing a connection before generating the traffic, such as Iperf and D-ITG. It will just work passively. Working as a client it is acting as a traffic emitter. Open loop packet-crafter tools such as Libtins do not require server operation to send the traffic. In the case of closed-loop tools, the destination IP addresses must be explicitly given in the command line by the options `-dst-list-ip` or `-dst-ip`.

```
# @ SIMITAR/, load enviroment variables
source data/config/simitar-workspace-config.sh

# @ SIMITAR/sniffer/, execute to sniff the eth0 interface, and create a trace entry
↪  called "intrig" in the database
./sniffer-cli.py new intrig live eth0

# @ SIMITAR/sniffer/, execute this command to list all traces recorded in the database
./sniffer-cli.py list

# @ SIMITAR/trace-analyzer/, execute this command to create two Compact Trace
↪  Descriptors, called intrig.ms.xml and intrig.sec.xml. The first is parameterized
↪  using milliseconds, and de second uses seconds as time unity.
./trace-analyzer --trace intrig

# @ SIMITAR/simitar-gen/, execute these commands to generate traffic using the
↪  intrig.sec.xml compact trace descriptor. It is stored at the directory
↪  "../data/xml/".
# Libtins
./simitar-gen --tool tins --mode client --ether eth0 --xml ../data/xml/intrig.sec.xml
# Iperf
./simitar-gen --tool iperf --mode client --ether eth0 --xml ../data/xml/intrig.sec.xml
↪  --dst-ip 10.0.0.2
./simitar-gen --tool iperf --mode server --ether eth0 --xml ../data/xml/intrig.sec.xml
```

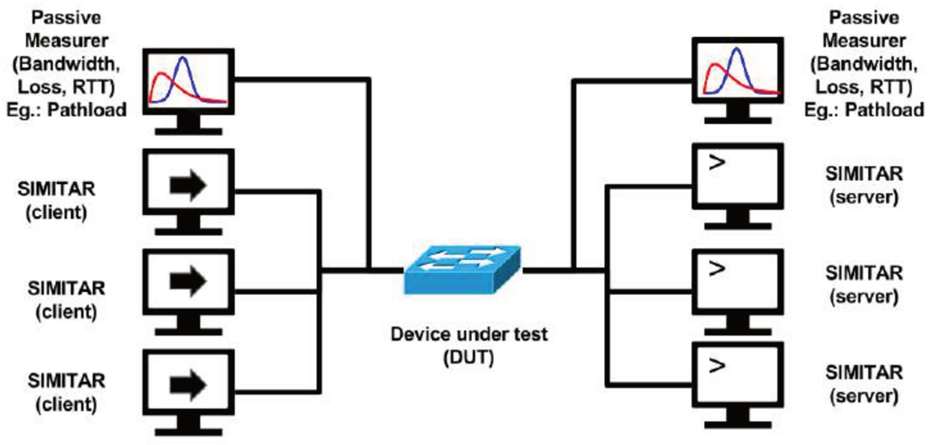


Figure 15 – Use case example for SIMITAR

Figure 15 shows an example of a use case. A device under test can be stressed using a combination of traffic generated by many clients and server pairs. In the case of an open-loop packet generator tool (such as in the Libtins implementation), the servers and clients pairs are not necessary. Using passive network measures, such as Pathload [Pathload – measurement tool for the available bandwidth of network paths 2006] and pathChirp [Vishwanath e Vahdat 2009] [pathChirp 2003], it is possible to measure statistics from the device under tests.

4 Modeling and Algorithms

This chapter describes in-depth the implementation of the traffic modeling algorithms mentioned in Chapter 3. The first section gives a brief survey of other works that presented methodologies for a parameterizing stochastic process to describe internet traffic (inter-packet times and packet-trains). This introduction shows a scenario where there is no "one-fits-all" model. The best model always depends on the type of traffic.

Many consolidate works investigate the nature of internet traffic, and many others on the modeling of stochastic functions for specific scenarios. However, there are not as many on model choice automation. In the second section, we discuss and cross-validate our methodology for automating the choice of inter-packet times processes. We select inter-packet times models using information criteria (*AIC* and *BIC*) to rank the models from the best to the worst. Since, to the best of our knowledge, there was no previous validation of the effectiveness of *AIC* and BIC for ranking information criteria, we developed our cross-validation methodology to test their effectiveness.

In the third section, we present our algorithm called "`calcOnOff`", a method to estimate packet trains periods (ON and OFF times) of an arbitrary flow, which do not rely on header fields, but just times between packets. Finally, the fourth section shows our strategy for guessing application protocols from flows, protocols using common transport header fields.

We use refer to two different cost functions on this chapter. The first is the Gadient Descendent Cost function, we refer as J_∇, since nabla (∇) is the notation used to represent gradients. The second is our cross-validation cost function, we refer as J_M, since it is used to rank stochastic models (M).

4.1 Background

There are many works devoted to studying the nature of the Ethernet traffic [Leland *et al.* 1994]. Classic Ethernet models used Poisson related processes to model traffic[1]. A Poisson process represents the probability of events occur with a known average rate, and independently of the last occurrence [Haight 1967].

However, studies made by Leland et al. showed that the Ethernet traffic has a self-similar and fractal nature. Even if they can represent the randomness of Ethernet traffic, simple Poisson processes cannot express traffic "burstiness" on a long-term timescale, such as traf-

[1] In our study case we used two Poisson related processes, both using exponential distributions: Exponential(Me) and Exponential(LR). Exponential distributions are considered continuous versions of the Poisson process. Since we are using time as a real number, we preferred to use just exponential distributions, for simplicity.

fic "spikes" on long-range "ripples". These characteristics are an indication of the fractal and self-similar nature of the traffic, that usually we express by distributions with infinite variance, called heavy-tailed. Heavy-tail means that a stochastic distribution is not exponentially bounded [Varet 2014], and can guarantee self-similarity via Joseph and Noah effects [Willinger *et al.* 1997]. Examples of heavy-tailed functions are Weibull, Pareto, Cauchy. However, these distributions may guarantee self-similarity, but not necessarily they will ensure other features like good correlation and same average packet rate [Molnár *et al.* 2013].

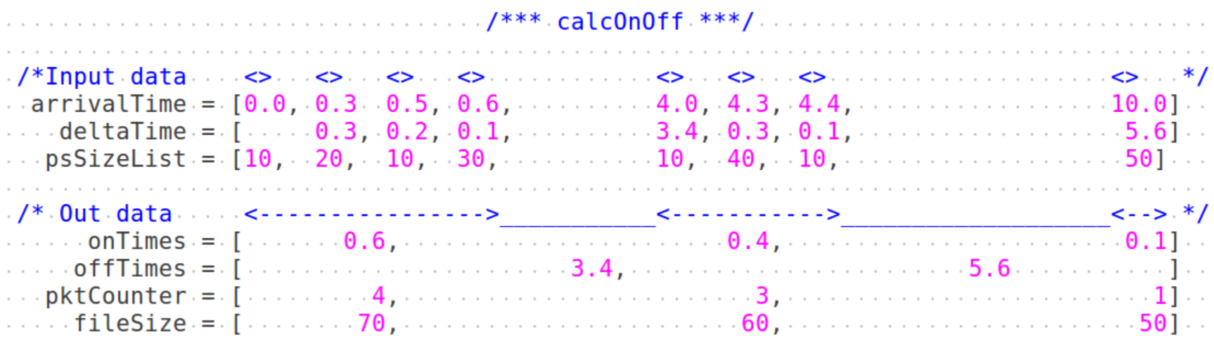
There are plenty of works in the literature which proposes processes and methodologies for modeling times between packets and packet trains [Leland *et al.* 1994] [Ju *et al.* 2009] [Rongcai e Shuo 2010] [Willinger *et al.* 1997] [Cevizci *et al.* 2006] [Markovitch e Krieger 2000] [Field *et al.* 2004] [Kushida e Shibata 2002] [Fiorini 1999] [Kronewitter 2006] [Field *et al.* 2004]. Fiorini [Fiorini 1999] presented a heavy-tailed ON/OFF model, which represented the traffic generated by many sources. The model emulated a multiple source power-tail Markov-Modulated (PT-MMPP) ON/OFF process, where the ON times have been power-tail distributed. They achieve analytical performance measurements using Linear Algebra Queueing Theory.

Kreban and Clearwater [Kleban e Clearwater 2003] presented a model for times between job submissions from multiple users over a supercomputer. They have shown that the Weibull probability functions can express well small and high values of inter-job submission times. They also have tested exponential, lognormal and Pareto distributions. The Exponential distribution has not represented well long-range values because of it had felt-off too fast. On the other hand, the Pareto problem was that it had felt too slow. Lognormal have fitted well small values, but its performance had been weak on larger ones. Kronewitter [Kronewitter 2006] has presented a model of scheduling traffic of many heavy-tail sources. On his work, he had used many Pareto sources to represent the traffic. To estimate Pareto shape (parameter α) he had used linear regression.

4.2 `calcOnOff`: an algorithm for estimating flow packet-train periods

As we discussed in Chapter 3, we developed an algorithm we call `calcOnOff`, listed on Algorithm 1. This procedure estimates the sizes of packet trains, as periods between packet trains in a flow context. This procedure receives as input values:

1. `arrivalTime`: the list of packets arrivals on time on the flow. For example, if we have five packets arriving every two seconds, we would have: `arrivalTime = [0, 2, 4, 6, 8]`;



```
                                /*** calcOnOff ***/

/*Input data    <>   <>   <>   <>           <>   <>   <>                   <>   */
  arrivalTime = [0.0, 0.3  0.5, 0.6,         4.0, 4.3, 4.4,                 10.0]
    deltaTime = [     0.3, 0.2, 0.1,         3.4, 0.3, 0.1,                  5.6]
   psSizeList = [10,  20,  10,  30,          10,  40,  10,                   50]

/* Out data     <----------------->___________<----------->__________________<--> */
      onTimes = [      0.6,                       0.4,                       0.1]
     offTimes = [                      3.4,                        5.6          ]
   pktCounter = [        4,                         3,                         1]
     fileSize = [       70,                        60,                        50]
```

Figure 16 – Textual representation of the input and output data of calcOnOff.

2. `deltaTime`: the list of inter-packet times on the flow. Following the same example presented before, we would have: `deltaTime = [2, 2, 2, 2]`.

3. `cutTime`: the time threshold that defines if we are still in the same train of packets, or a new one.

4. `minOnTime`: is the minimum length of flow ON time. minOnTime were used mainly to avoid ON times of zero seconds, in the case of only one packet, or when the time between packets is smaller than the operational system precision.

5. `psSizeList`: the list of packet sizes in bytes.

For example, suppose a list of arrival times:

`arrivalTime = [0.0, 0.3 0.5, 0.6, 4.0, 4.3, 4.4 , 10.0]`

We would have the follow list of inter-packet times:

`deltaTime = [0.3, 0.2, 0.1, 3.4, 0.3, 0.1, 5.6]`

Suppose the list of packet sizes is:

`psSizeList = [10, 20, 10, 30, 10, 40, 10, 50]`

With a `minOnTime` of 0.1 and a `cutTime` of 3 seconds, we would have the following output:

`onTimes = [0.6, 0.4, 0.1]`

`offTimes = [3.4, 5.6]`

`pktCounter = [4, 3, 1]`

`fileSize = [70, 60, 50]`

Figure 16 shows the same example, but with a text visualization, to simplify the visualization of the grouped data.

Algorithm 1 calcOnOff

```
function calcOnOff($arrivalTime, deltaTime, cutTime, minOnTime$)
    $m = deltaTime.length() - 1$
    $j = 0$
    $lastOff = 0$
    $pktCounterSum = 0$
    $fileSizeSum = 0$
    for $i = 0 : m$ do
        $pktCounterSum = pktCounterSum + 1$
        $fileSizeSum = fileSizeSum + psSizeList[i, 1]$
        if $deltaTime[i] > cutTime$ then
            if $i == 1$ then                                    ▷ if the first is session-off time
                $j++$
                $onTimes.push(minOnTime)$
                $offTimes.push(deltaTime[i])$
                $pktCounter.push(pktCounterSum)$
                $fileSize.push(fileSizeSum)$
                $pktCounterSum = 0$
                $fileSizeSum = 0$
            else                                                ▷ base case
                $pktCounter.push(pktCounterSum)$
                $fileSize.push(fileSizeSum)$
                $pktCounterSum = 0$
                $fileSizeSum = 0$
                if $j == 0$ then
                    $onTimes.push(arrivalTime[i-1])$
                    $offTimes.push(deltaTime[i])$
                else                                            ▷ others on times
                    $onTimes.push(max(deltaTime[i-1] - deltaTime[lastOff], minOnTime))$
                    $offTimes.push(deltaTime[i])$
                end if
                $lastOff = i$
            end if
        end if
    end for
    $pktCounterSum = pktCounterSum + 1$
    $fileSizeSum = fileSizeSum + psSizeList[m]$
    if $lastOff == m - 1$ then                                  ▷ if last is session-off
        $onTimes.push(minOnTime)$
    else                                                        ▷ base last case
        if $lastOff \neq 0$ then
            $onTimes.push(arrivalTime[m] - arrivalTime[lastOff])$
        else
            $onTimes.push(arrivalTime[m])$                      ▷ there was just on time
        end if
    end if
    $pktCounter.push(pktCounterSum)$
    $fileSize.push(fileSizeSum)$
    return $onTimes, offTimes, pktCounter, fileSize$
end function
```

4.3 Typical header fields by Application protocols

We also developed a simple test to guess the application protocol, based on the port numbers and the transport protocol used by each flow. If a flow matches the port number, and the transport protocol, it matches an application protocol, following the rules on Table 6.

Table 6 – Application match table

Application Protocol	Transport Protocols	Transport Ports
HTTPS	TCP	443
FTP	TCP	20, 21
HTTP	TCP	80
BGP	TCP	179
DHCP	UDP	67, 68
SNMP	UDP, TCP	161
DNS	UDP, TCP	53
SSH	UDP, TCP	22
Telnet	UDP, TCP	23
TACACS	UDP, TCP	49

4.4 Automated Selection of Inter-Packet Times

There many consolidate works which have investigated the nature of internet traffic, and many others that had proposed processes to describe network traffic on specific scenarios. But not as many on model choice automation. We propose and evaluate the use of the information criteria *BIC* (Bayesian Information Criterion) and *AIC* (Akaike Information Criterion) as suitable methods for automated model selection for inter-packet times.

4.4.1 Cross-validation Methodology

We then define our cross-validation method based on a cost function J_M, which is an aggregator of traditional and critical metrics used on validation of stochastic models and traffic generators: Correlation(quality of the model), Average inter packet-time (related with the traffic Throughput), and Hurst Exponent (traffic fractal level). J_M assign weights from the best to the worst representation for each property of each trace model by using randomly generated data with our stochastic fittings. Through this process, we choose the best-fitted traffic model under evaluation. Afterward, we compare the results achieved by *AIC*/*BIC* and our cost function. Given the approach mentioned above, we show that *AIC*/*BIC* methods provide an intelligent stochastic process selection strategy for inter-packet times models. We summarize the validation process in the steps below:

1. We have selected many *pcap* files, representing different types of network scenarios. We extracted from these files the list of inter-packet times.

2. Using a set of stochastic functions, and parameterization methods, we defined a list of candidate stochastic processes to represent the inter-packet time's distribution for each dataset.

3. For each dataset, *AIC* and *BIC* were used to rank the processes, from the best to the worst, for each dataset.

4. Using each process from step four, we generated random data and estimated the cost function J. We have repeated the random data generation 30 times, and the parameters used on the cost function had ion had a confidence interval of 95%. Finally, we have ranked the processes from the best to the worst for each dataset.

5. Finally, we had two independent rankings, the one we wanted to validate, and others based on the literature. We compared the results.

We also present some *QQplots* to visually compare the random-generated data and the original data-set.

4.4.2 Datasets

We will use four *pcap* files, where three are publicly available, to enable the reproduction of the simulations described in this chapter. The first is a lightweight Skype capture, found in Wireshark wiki 2, located at https://wiki.wireshark.org/Samp[2], located at https://wiki.wireshark.org/SampleCaptures. The file name is `SkypeIRC.cap`. We will call this trace *skype-pcap*. The second is a CAIDA[3]http://www.caida.org/home/ capture, and we can found it at https://data.caida.org/datasets/passive-2016/equinix-chicago/20160121-130000.UTC. Access to this file requires a login, so is required the creation and approval of a new account. The *pcap*'s file name is `equinix-chicago.dirB.20160121-135641.UTC.anon.pcap.gz`. We call it *wan-pcap*[4].

The third we capture in our laboratory LAN, through a period of 1 hour in a firewall gateway between our local and external network. We call it *lan-firewall-pcap*. The fourth is a capture of a busy private network access point to the Internet, available online on TCPreplay website[5], called `bigFlows.pcap`. We will refer to it *lan-gateway-pcap*.

[2] https://wiki.wireshark.org/
[3] http://www.caida.org/home/
[4] WAN stands for Wide Area Network
[5] http://tcpreplay.appneta.com/wiki/captures.html

We retrieved inter-packet times from the traffic traces and divided them into two equally sized datasets. We split the data based on the index of the array we use to store. Odd-indexed elements have been used as a training dataset, and even-indexed for cross-validation; to avoid data over-fitting.

4.4.3 Stochastic Processes Modeling and Selection

4.4.3.1 Stochastic Processes

We have adopted five stochastic functions (**Weibull**, **Normal**, **Exponential**, **Pareto** and **Cauchy**), and three methods for parameters estimation: **Linear Regression**, **Direct estimation**, and **Maximum Likelihood**, totaling seven stochastic processes :

1. **Weibull** via Linear Regression;
2. **Normal** via Direct Estimation;
3. Exponential via Direct Estimation, we refer as **Exponential(LR)**;
4. Exponential via Linear Regression, we refer as **Exponential(MLH)**;
5. Pareto via Linear Regression, we refer as **Pareto(LR)**;
6. Pareto via Maximum Likelihood, we refer as **Pareto(MLH)**;
7. **Cauchy** via Linear Regression.

The data that have been used for modeling was the training dataset. *AIC*, *BIC* and the cost function have used the cross-validation dataset. Since the time samples resolution used were of 10^{-6}s, all values equal to zero had been set to $5 \cdot 10^{-8}$s, to avoid division by zero. To avoid divergence in tangent operation used Cauchy linearization process, we have floor-limited and upper-limited the inter-packet CDF values by 10^{-6} and 0.999999, respectively. We implemented this prototype using Octave and Python. We upload all the code and data from these experiments on Github [Paschoalon 2019].

4.4.3.2 Linear Regression (Gradient descendant)

Linear regression is a method for estimating the best linear curve in the format:

$$y = ax + b \tag{4.1}$$

to fit a given data set. We can use linear regression to estimate parameters of a non-linear curve expressing it on a linear format. For example, the Weibull CDF for $t > 0$ is:

$$F(t|\alpha,\beta) = F(t) = 1 - e^{-(t/\beta)^{\alpha}} \tag{4.2}$$

Manipulating the equation:

$$\alpha \ln(t) - \alpha \ln(\beta) = \ln(-\ln(1 - F(t))) \tag{4.3}$$

If we call $x = \ln(t)$ and $y = \ln(-\ln(1 - F(t)))$, we found a linear equation, where $a = \alpha$ and $b = -\alpha \ln(\beta)$. Having in hands an estimation of the empirical CDF of our data samples, we apply the x and y definitions to linearize the data.

Using the gradient descendant, we find an estimation of the linear coefficients: $\hat{a}$ and $\hat{b}$. Using the inverse function of linear factors, we see the Weibull estimated parameters $\hat{\alpha}$ and $\hat{\beta}$.

$$\alpha = a \tag{4.4}$$

$$\beta = e^{-(b/a)} \tag{4.5}$$

The gradient descendent consists of minimizing a cost function $J_{\nabla}(\theta)$, whose value decrease if the approximation becomes better. We explain this procedure in the Appendix A. In the Figure 17a we present as examples, the linearized data for the inter arrivals from the *skype-pcap*, and in the Figure 17b the cost convergence. Appendix D presents a complete set of figures.

Applying the inverse equations of the linear coefficients ($\hat{\alpha} = \hat{a}$ and $\hat{\beta} = e^{-(\hat{b}/\hat{a})}$ [6], we can estimate the Weibull distribution parameters. We can summarize this procedure, in these steps:

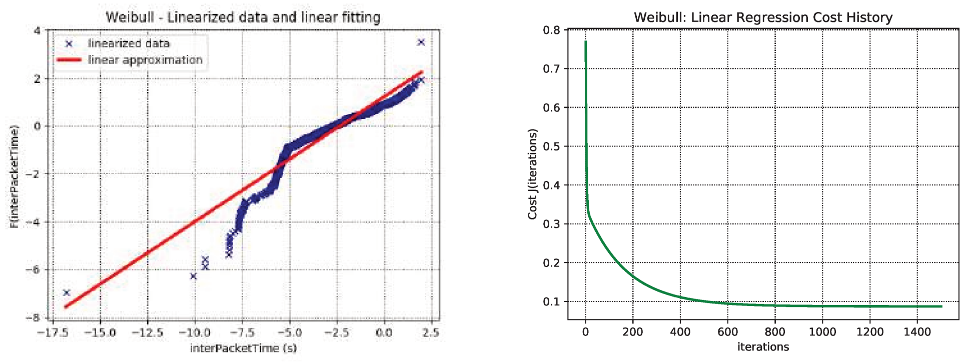
1. Linearize the stochastic CDF function $F(t)$.
2. Apply the linearized $y = y(F(t))$ and $x = x(t)$ on the empirical CDF and times datasets, respectively.
3. Use Gradient Descendant algorithm to find linear coefficients a and b.
4. Apply the inverse equation of the linear coefficients, to determine the stochastic function parameters.

[6] We use the hat symbol ($\hat{\ }$) for estimated parameters

In the parameters estimation (step 4), there is an exception, since the Pareto scale (x_m) is defined by the smaller time allowed. In Table 7 we present a summary of the used equations in the procedure. In this notation, the subscript i means that it is applied to every value measured empirically.

Table 7 – Linearized functions, and parameters estimators, used by the linear regression

Function	Linearized x	Linearized y	Parameters Estimator	
Cauchy	$x_i = t_i$	$y_i = \tan(\pi(F(t_i) - 1/2))$	$\hat{\gamma} = \frac{1}{\hat{a}}$	$\hat{t_0} = -\frac{\hat{b}}{\hat{a}}$
Exponential	$x_i = t_i$	$y_i = \ln(1 - F(t_i)))$	$\hat{\lambda} = -\hat{a}$	
Pareto	$x_i = \ln(t_i)$	$y_i = \ln(1 - F(t_i))$	$\hat{\alpha} = -\hat{a}$	$\hat{x_m} = \min_{i=0,\ldots,m}\{x_i\}$
Weibull	$x = \ln(t)$	$y = \ln(-\ln(1 - F(t)))$	$\hat{\alpha} = \hat{a}$	$\hat{\beta} = e^{-(\hat{b}/\hat{a})}$



(a) Linearized interarrival data (b) Cost function J_∇ of the linear regression

Figure 17 – Linearized data and cost function J_∇ of weibull linear regression

4.4.3.3 Direct Estimation

The expected value $E(X)$ and variance $Var(X)$ of a random variable X of some distributions are closely related to its parameters. Since the average $\bar{\mu}$ and its standard deviation$\bar{\sigma}$ are in general good approximations for the expected value and variance, we use them to estimate parameters.

Following the notation presented at Table 2, we have for the normal distribution the following ordered pair of parameters:

$$(\hat{\mu}, \hat{\sigma}) = (\bar{\mu}, \bar{\sigma}) \tag{4.6}$$

For the exponential distribution $E(X) = \frac{1}{\lambda}$. Therefore:

$$\hat{\lambda} = \frac{1}{\bar{\mu}} \tag{4.7}$$

4.4.3.4 Maximum Likelihood

The maximum likelihood estimation, is a method for estimation of The maximum likelihood estimation, is a method for estimation of The maximum likelihood estimation, is a method for estimation of parameters, winch maximizes the likelihood function. We explain in details this subject in Appendix A. Using this method for the Pareto distribution; it is possible to derive the following equations for its parameters:

$$\hat{x_m} = \min_{i=0,\ldots,m} \{x_i\} \tag{4.8}$$

$$\hat{\alpha} = \frac{n}{\sum_{i=0}^{m}(\ln(x_i) - \ln(\hat{x_m}))} \tag{4.9}$$

where m is the sample(dataset) size.

4.4.3.5 *AIC* and *BIC*

Suppose that we have an statistical model M of some dataset $x = \{x_1, ..., x_n\}$, with n independent and identically distributed observations of a random variable X. This model can be expressed by a PDF $f(x|\theta)$, where θ a vector of parameter of the PDF, $\theta \in \mathbb{R}^k$ (k is the number of parameters). The likelihood function of this model M is given by:

$$L(\theta|x) = f(x_1|\theta) \cdot \ldots \cdot f(x_n|\theta) = \prod_{i=1}^{n} f(x_i|\theta) \tag{4.10}$$

Now, suppose we are trying to estimate the best statistical model, from a set $M_1, ..., M_n$, each one whit an estimated vector of parameters $\hat{\theta}_1, ..., \hat{\theta}_n$. *AIC* and *BIC* are defined by:

$$AIC = 2k - \ln(L(\hat{\theta}|x)) \tag{4.11}$$

$$BIC = k\ln(n) - \ln(L(\hat{\theta}|x)) \tag{4.12}$$

In both cases, the preferred model M_i, is the one with the smaller value of AIC_i or BIC_i.

4.4.4 Cross-validation method: Theoretical Foundation of the Cost Function

To test if *AIC* and *BIC* do perform well, we had to answer two questions:

#1 How do we evaluate if stochastic models and network traffic are playing well according to the expected?

#2 What reliable and trustable methods exist in the literature could we use to compare with *AIC* and *BIC*?

To answer question #1, the first thing is finding how well do a model can explain a given dataset. To answer this question we find two widely adopted alternatives:

- **r-square**: applied just for linear regression, and measures how well a linearization can explain the data
- **Pearson correlation coefficient**: can measures how well two random variables relate to each other.

Therefore, the metric that best satisfies our needs is the Pearson correlation coefficient. The two random variables in question are the cross-validation dataset, and random values generated by the stochastic model. We compared it repeatedly (30 times) to obtain a small enough confidence interval. Its value goes from -1 to +1. +1 means a perfect direct linear correlation. "-1" indicates a perfect inverse linear correlation."0" means no linear correlation. So, as close the result reaches "1", more similar are the inter-packet times to the original values. To estimate it, we use the Octave's function `corr()`.

After addressing the question on the quality of data generated by the model, we need to evaluate the quality for actual, real network traffic, following the guidelines presented on Chapter 2, where we had defined four metric classes:

1. Packet-level metrics
2. Flow level metrics
3. Scaling characteristics
4. QoS/QoE related metrics

Since we do not distinguish the traffic between flows, neither evaluate QoS/QoE metrics, we could not directly extract metrics from these classes. Considering packet sizes in the first category is out of scope for our work and has been already studied in the literature.

On the metric class (1), the authors mention that the most common metric widely adopted is throughput. We can calculate the throughput on packets per second or byte rate. Both measures can be estimated using the mean inter-packet time. The mean packet rate per second can be estimated by:

$$packet\_throughput = 1/(mean\_inter\_packettime) \tag{4.13}$$

So knowing the average packet size, we can estimate the byte by:

$$byte\_throughput = (average\_packet\_size)/(mean\_inter\_packet\_time) \tag{4.14}$$

Therefore a good approximation on the average inter-packet time means a reasonable estimate on the traffic throughout as well. Another metric cited by the authors in the scope is the analysis of inter-packet size distributions. Researchers typically make distribution analyzes via graphical interpretation. Seeing its importance, we show as an example one of our results on that matter, now on Figure 18. This Figure shows the comparison of four CDF functions estimations for the *skype-pcap*, with the cross-validation data. Works such as [Botta *et al.* 2012] [Varet 2014] [Botta *et al.* 2010] [Sommers e Barford 2004] [Emmerich *et al.* 2015] [Field *et al.* 2004] did the same task, for their own purposes using the PDF or CDF distributions. We opted to present the CDF plot because we found it easier for discussion. However, since the goal of our work is to compare *AIC* and *BIC* with other possible rankings inspired in the literature, we had to abstract these plots in a single metric that represents the fitting quality. To this end, we use the (already mentioned) Pearson correlation coefficient.

Finally, there is (3) Scaling characteristics, a well-known aspect of traffic modeling. From the seminal work of Leland et al. [Leland *et al.* 1994], we know that the ethernet traffic has a self-similar and fractal-like behavior. Some of the most important are the estimation of the Hurst exponent (used on the last mentioned paper) [Leland *et al.* 1994] [Cevizci *et al.* 2006] [Abry e Veitch 1998], Wavelet multiresolution analysis [Cevizci *et al.* 2006] [Abry e Veitch 1998] [Vishwanath e Vahdat 2009], power-law and power spectrum analysis, as suggested by [Field *et al.* 2004]. However, to the best of our knowledge, there is no other method widespread as Hurst exponent and Wavelet analysis. The problem of wavelet for our purposes is that its interpretation is inherently graphical, and can be used to identify periodicities, white-noise, and self-similar characteristics on a long-range analysis, depending on the graphical behavior. However, for automatic model ranking, this approach is not practical.

On the other hand, the Hurst exponent ideal for this task, since it is a single value, and is a representation of the fractal dimension of the dataset. Therefore, we choose to rank the scaling characteristics of our models, based on how close they can get from the estimation of the Hurst exponent of the cross-validation. We repeated the estimation for both cross-validation and synthetic dataset 30 times until we got a fair and small error margin. We did not consider in our analysis of other stochastic metrics, such as the standard deviation of the dataset. Although the standard deviation is useful on the understanding of the nature of the traffic, we judge that it be redundant with the Hurst exponent estimation, since it is already a measure of the data variability. To determine this value, we use the function `hurst()` from Octave, which uses the rescaled range method.

Therefore, we got three different rankings based on the literature, each of them giving their estimation of what model performs better. To do not prioritize any of these metrics, and provide a ponderated best result we created the so-called cost function J, which is an aggregator of results. Being Cr the vector of correlations ordered from higher to the smaller, let Me and Hr defined by the absolute difference between average and hurt exponent of the estimated values and the original data-set, both ordered from the small to the high values. Letting $\phi(V,M)$ be an operator which gives the position of a model M in a vector V, we define the cost function J as:

$$J_M(M) = \phi(Cr,M) + \phi(Me,M) + \phi(Hr,M) \tag{4.15}$$

The smaller is the cost J_M; the best is the model. For example, suppose a model m1 that has the best performance on the average inter-packet time estimation, but delivers smaller performance on data correlation and the Hurst exponent. That means, this model was able to overfit the mean inter-packet time representation but does a poor data representation on all the other cases. Since we are comparing seven models, this model can deliver the following cost estimation (counting starts from position 0):

$$J_M(M) = \phi(Cr,m_1) + \phi(Me,m_1) + \phi(Hr,m_1) = 6+0+6 = 12 \tag{4.16}$$

On the other hand, suppose a model m2 that performs as second best on all the tests. This one will have:

$$J_M(M) = \phi(Cr,m_2) + \phi(Me,m_2) + \phi(Hr,m_2) = 1+1+1 = 3 \tag{4.17}$$

Although model m_1, if used for traffic generation would perform well on the throughput representation, it is not fair that it goes ahead of the second model m_2, which performed pretty well on all the metrics. Therefore, we can safely say that m_2 a better representation of inter-packet times, according to metrics typically adopted in the literature of network traffic and stochastic analysis.

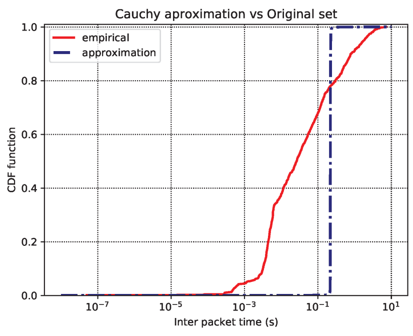
4.5 Results

Here in this chapter we only discuss the approximation and *QQplots* achieved obtained by the *pcap skype-pcap*. All the other comments will be referent to the general results.

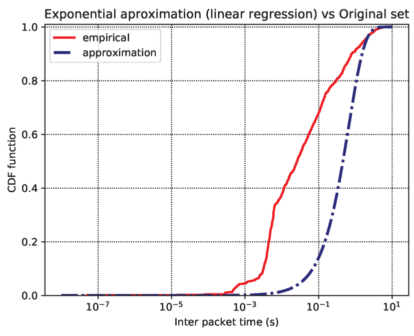
Table 8 – Experimental results, including the estimated parameters and the BIC and AIC values of the four *pcap* traces.

Function	Trace							
	AIC		Parameters		AIC	BIC	Parameters	
	skype-pcap				lan-firewall-pcap			
Cauchy	$6.94E+03$	$6.95E+03$	$\gamma: 1.71E-04$	$x_0: 1.88E-01$	$-2.29E+05$	$-2.29E+05$	$\gamma: 1.93E-02$	$x_0: -4.97E-02$
Exponential(LR)	$-4.70E+01$	$-4.28E+01$	$\lambda: 1.79E+00$		$-2.22E+06$	$-2.22E+06$	$\lambda: 4.05E-01$	
Exponential(Me)	$-2.16E+02$	$-2.12E+02$	$\lambda: 3.45E+00$		$3.63E+05$	$3.63E+05$	$\lambda: 1.13E+02$	
Normal	$1.21E+03$	$1.22E+03$	$\mu: 2.90E-01$	$\sigma: 6.95E-01$	$-1.48E+06$	$-1.48E+06$	$\mu: 8.85E-03$	$\sigma: 3.49E-02$
Pareto(LR)	$3.38E+03$	$3.39E+03$	$\alpha: 4.28E-01$	$x_m: 5.00E-08$	Inf[1]	Inf[1]	$\alpha: 2.51E-01$	$x_m: 5.00E-08$
Pareto(MLH)	$1.88E+02$	$1.97E+02$	$\alpha: 7.48E-02$	$x_m: 5.00E-08$	$-1.80E+06$	$-1.80E+06$	$\alpha: 1.15E-01$	$x_m: 5.00E-08$
Weibull	$-1.15E+03$	$-1.14E+03$		$\beta: 9.68E-02$	$-1.97E+06$	$-1.97E+06$	$\alpha: 3.46E-01$	$\beta: 1.79E-03$
	lan-gateway-pcap				wan-pcap			
Cauchy	$3.65E+06$	$3.65E+06$	$\gamma: 1.95$	$x_0: -4.45E+03$	$2.99E+07$	$2.99E+07$	$\gamma: 8.17E+02$	$x_0: -4.45E+03$
Exponential(LR)	$3.67E+06$	$3.67E+06$	$\lambda: 9.75E-03$		$2.84E+07$	$2.84E+07$	$\lambda: 2.20E-05$	
Exponential(Me)	$-5.44E+06$	$-5.44E+06$	$\lambda: 2.64E+03$		$-3.29E+07$	$-3.29E+07$	$\lambda: 6.58E+05$	
Normal	$-4.67E+06$	$-4.67E+06$	$\mu: 3.79E-04$	$\sigma: 1.00E-06$	$-3.19E+07$	$-3.19E+07$	$\mu: 2.00E-06$	$\sigma: 1.00E-06$
Pareto(LR)	$-5.13E+06$	$-5.13E+06$	$\alpha: 1.49E-01$	$x_m: 5.00E-08$	$4.51E+07$	$4.51E+07$	$\alpha: 4.00E-14$[2]	$x_m: 5.00E-08$
Pareto(MLH)	$-5.13E+06$	$-5.13E+06$	$\alpha: 1.36E-01$	$x_m: 5.00E-08$	$-3.13E+07$	$-3.13E+07$	$\alpha: 3.39E-01$	$x_m: 5.00E-08$
Weibull	$-5.50E+06$	$-5.50E+06$	$\alpha: 2.81E-01$	$\beta: 1.00E-06$	$-2.73E+07$	$-2.73E+07$	$\alpha: 7.64E-02$	$\beta: 1.00E-06$

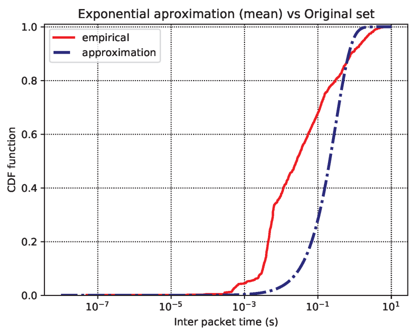
[1] The computation of the likelihood function has exceeded the computational precision used, so it was the highest AIC and BIC for this trace.
[2] The linear regression did not converge to a valid value, so we used a small value instead to perform the computations.



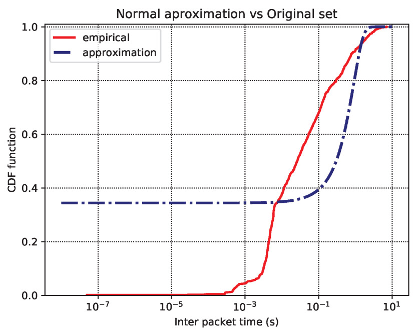
(a) Chauchy



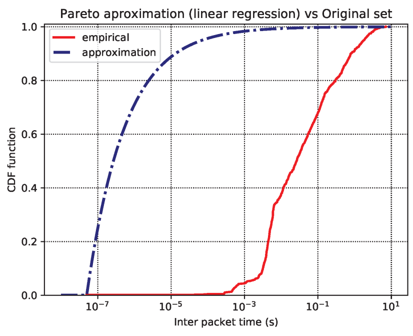
(b) Exponential(LR)



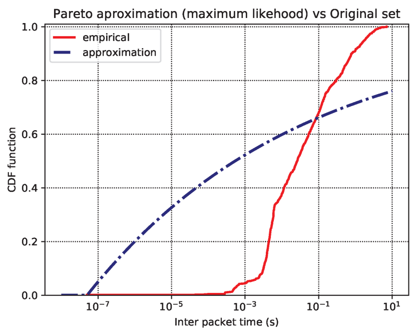
(c) Exponential(Me)



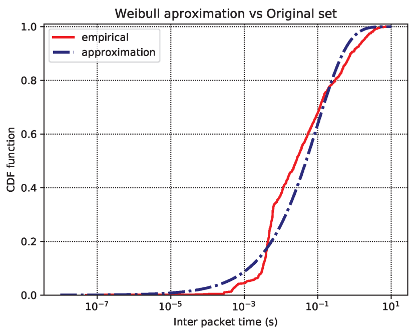
(d) Normal



(e) Pareto(LR)

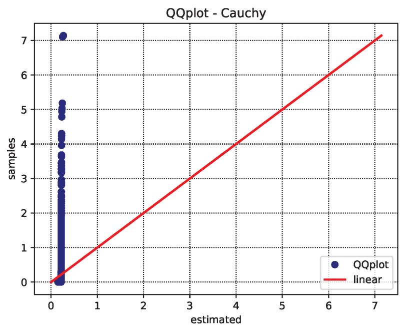


(f) Pareto(MLH)

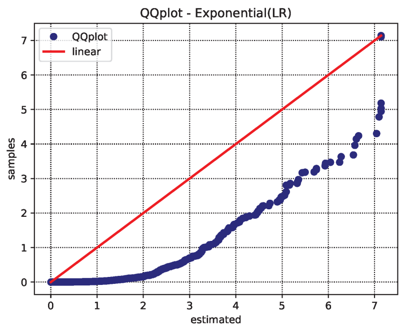


(g) Weibull

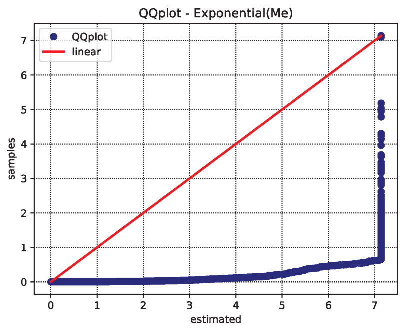
Figure 18 – CDF functions for the approximations of *skype-pcap* inter packet times, of many stochastic functions.



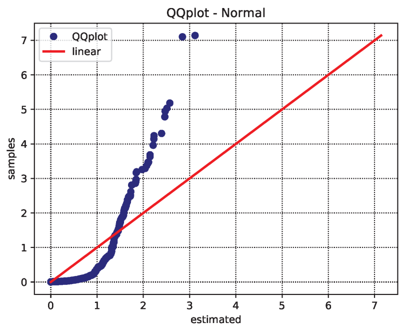
(a) Chauchy



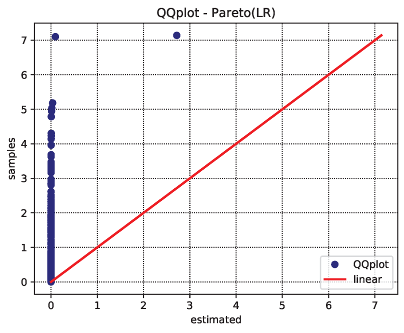
(b) Exponential(LR)



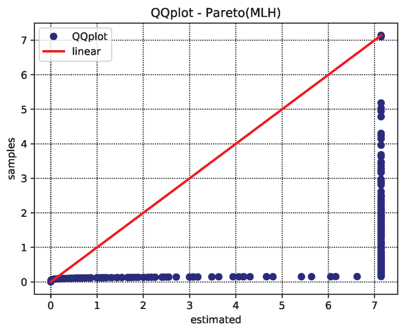
(c) Exponential(Me)



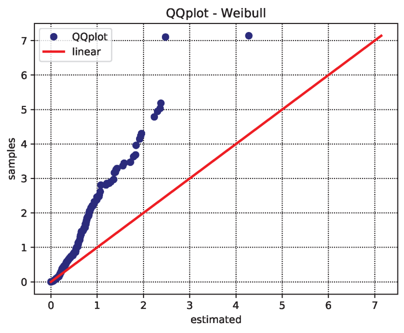
(d) Normal



(e) Pareto(LR)

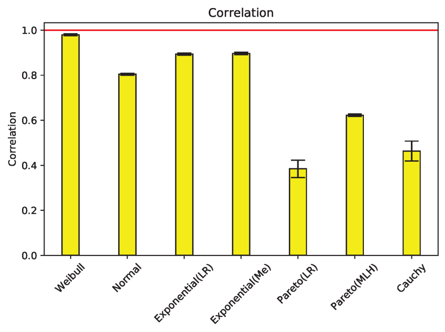


(f) Pareto(MLH)

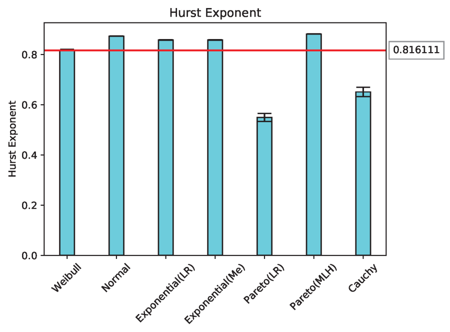


(g) Weibull

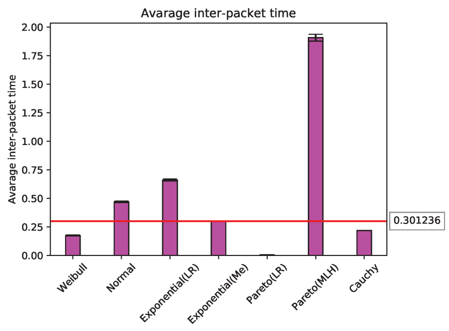
Figure 19 – CDF functions for the approximations of *skype-pcap* inter packet times, of many stochastic functions.



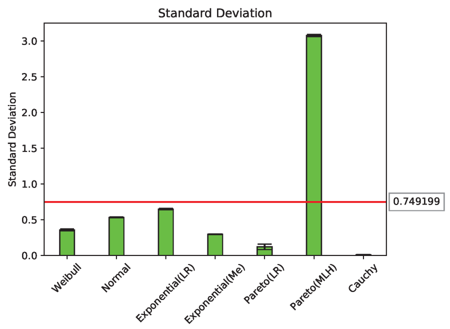
(a) Correlation



(b) Hust Exponent

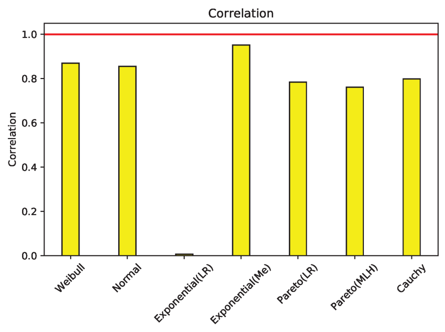


(c) Mean

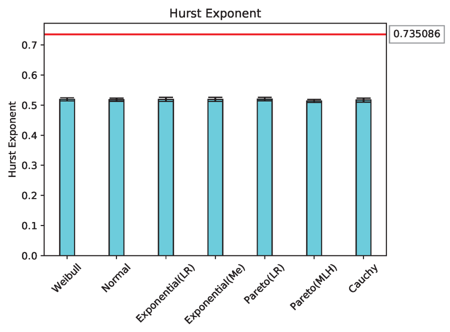


(d) Standard Deviation

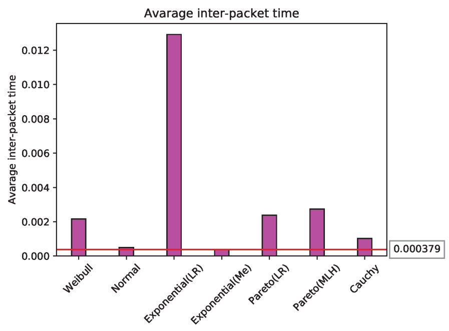
Figure 20 – Statistical parameters of *skype-pcap* and its approximations



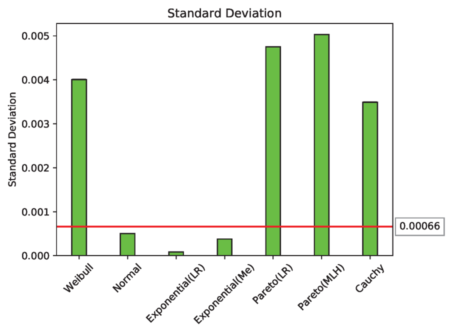
(a) Correlation



(b) Hust Exponent

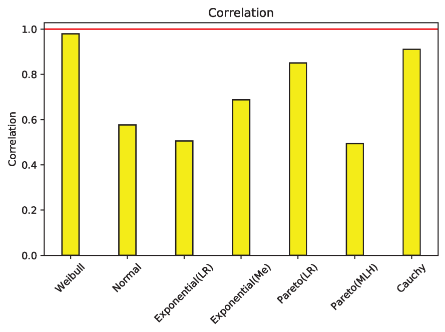


(c) Mean

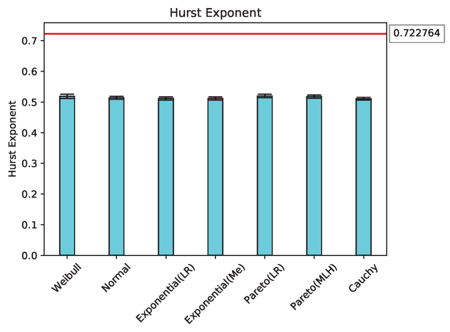


(d) Standard Deviation

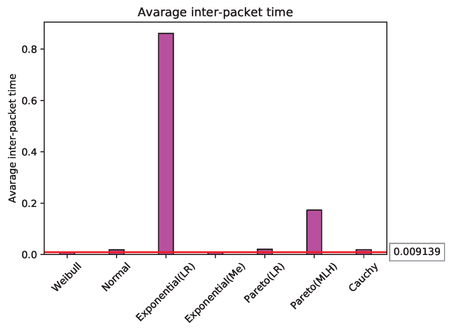
Figure 21 – Statistical parameters of *lan-gateway-pcap* and its approximations



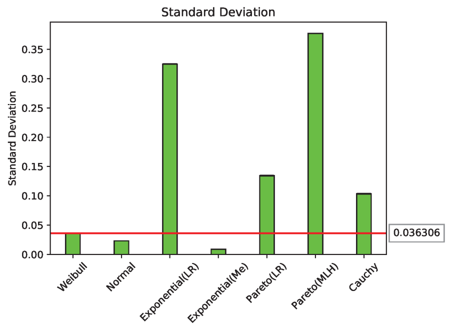
(a) Correlation



(b) Hust Exponent

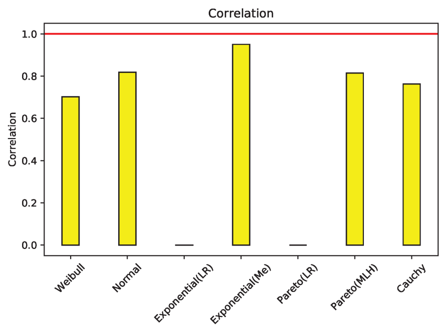


(c) Mean

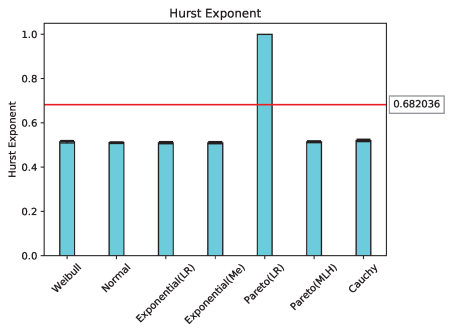


(d) Standard Deviation

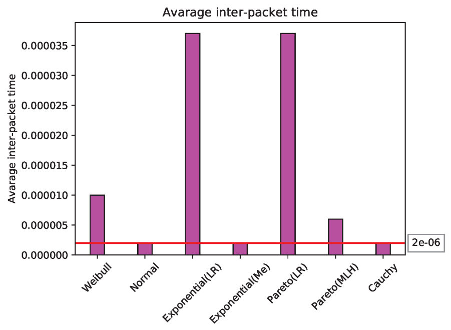
Figure 22 – Statistical parameters of *lan-firewall-pcap* and its approximations



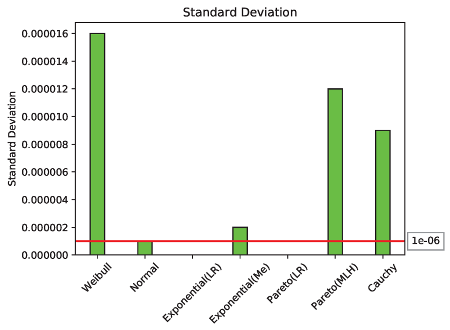
(a) Correlation



(b) Hust Exponent



(c) Mean

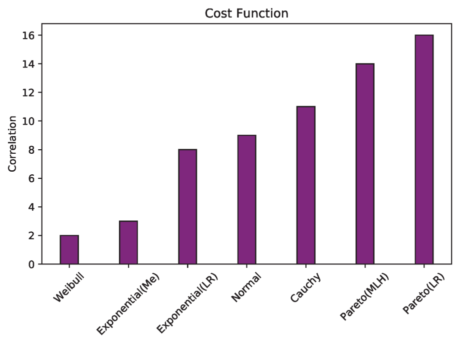


(d) Standard Deviation

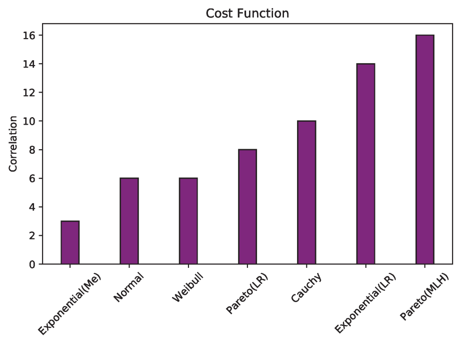
Figure 23 – Statistical parameters of *wan-pcap* and its approximations

Table 9 – Relative difference(%) between *AIC* and *BIC*.

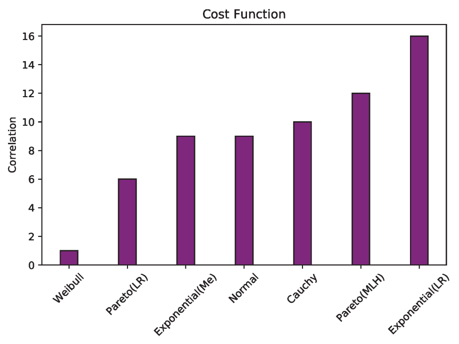
	skype-pcap	lan-gateway -pcap	wan-pcap	lan-firewall -pcap
Weibull	$7.47E-01$	$3.96E-04$	$8.86E-05$	$9.21E-04$
Normal	$7.04E-01$	$4.66E-04$	$7.58E-05$	*NaN*
Exponential(LR)	$9.54E+00$	$2.97E-04$	$4.26E-05$	$2.81E-03$
Exponential(Me)	$2.00E+00$	$2.00E-04$	$3.68E-05$	$6.90E-04$
Pareto(LR)	$2.53E-01$	$4.25E-04$	$5.36E-05$	$1.13E-03$
Pareto(MLH)	$4.45E+00$	$4.25E-04$	$7.74E-05$	$1.04E-03$
Cauchy	$1.23E-01$	$5.97E-04$	$8.08E-05$	$8.90E-03$



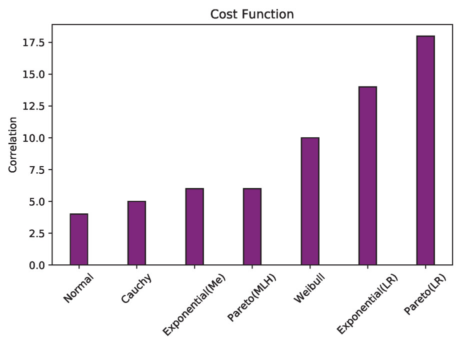
(a) *skype-pcap*



(b) *lan-gateway-pcap*



(c) *lan-firewall-pcap*



(d) *wan-pcap*

Figure 24 – Cost function J_M for each one of the data-sets used in this validation process

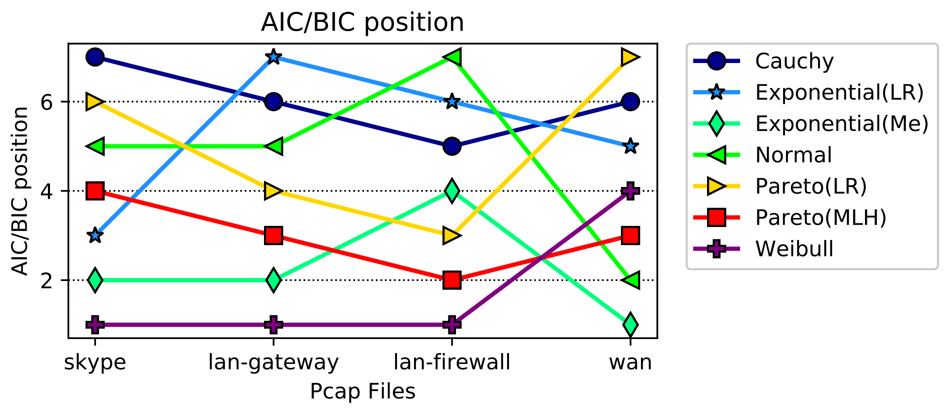


Figure 25 – Comparision of the quality order of each model given by *AIC* and *BIC*

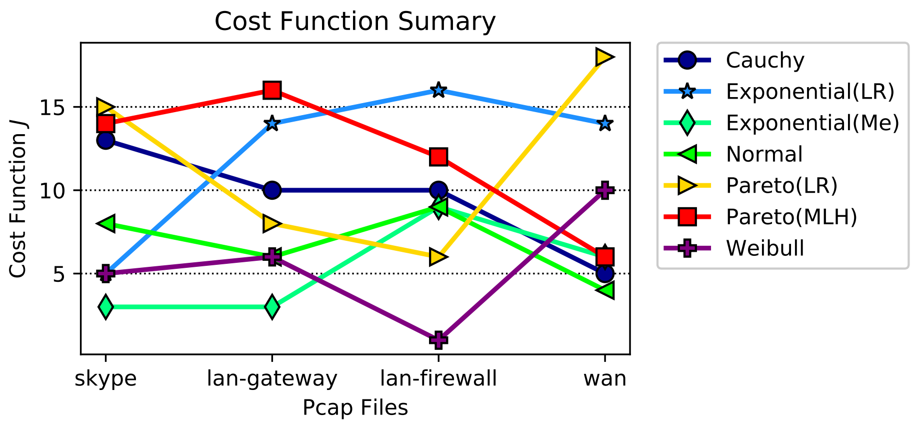


Figure 26 – J_M for each one of the datasets used in this validation process.

The other fitting and QQ plots are provided on Appendix D as a reference. We divided our analysis into 5 steps:

1. *CDF plots*
2. *QQplots*
3. *AIC and BIC*
4. *Correlation, Hurst Exponent, Mean, and Standard Deviation*
5. *AIC and BIC vs.* J_M

CDF plots

Figure 18 shows the CDF plots for *skype-pcap*. They are on log-scale, which provide a better visualization for small time scales.

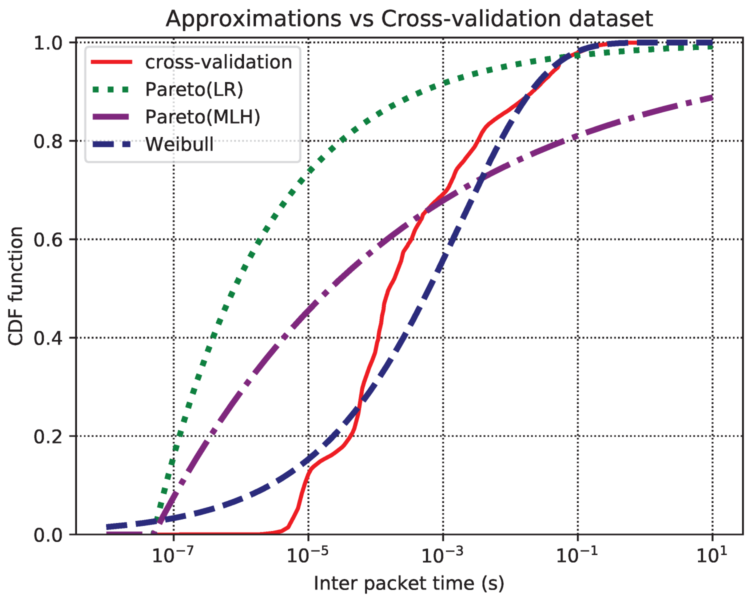


Figure 27 – Inter-packet times CDF function and stochastic models for *firewall-pacap*.

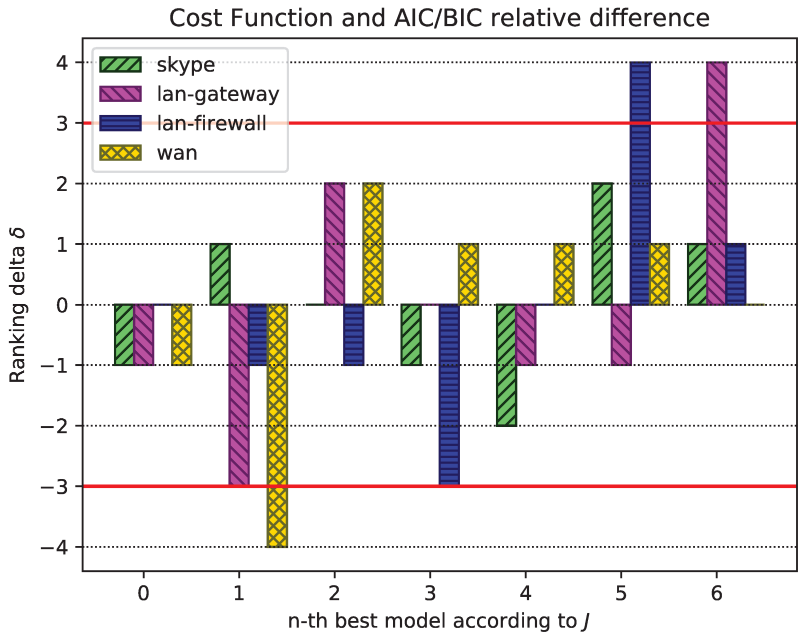


Figure 28 – Comparison of the model selection order for *BIC*/*AIC* and J_M for each *pcap*.

- We initially expected a good performance of heavy-tailed processes, but it depended on each case.
- Visually, the best result have seemed to be the Weibull trough linear regression, followed by Exponential(Me) – what the *AIC*/*BIC* analysis (Figure 25) have confirmed. Also, Fig-

ure 20 show Weibull having the best Correlation and Hurst exponent, and the third best mean; while Exponential(Me) have had the best mean and the second best Correlation and Hurst exponent.

- The Cauchy process have imposed an almost constant traffic, with the average inter-packet time close to the mean. Figure 20 confirms this observation since the process standard deviation is small, and the mean inter-packet time is close to the original (the second best);
- Exponential processes did not represented well small and larger values, compared to heavy-tailed processes, but were good to describe values closer to the average. On the other hand, a self-similar processes, such as Weibull and Pareto, have had a higher dispersion. The random-generated values were more wide-spared over the time – what did not necessarily have implied on quality, as we are going to see. For example, Pareto(MLH) tended to over-estimate the amount of long values of inter-packet times, having a slower convergence of the CDF to one compared to the other processes – about 20% of the values are larger than 10 seconds. Cauchy for most of the case as a good process to represent the mean inter-packet times, but did no performed so well on other metrics (Figures 23, 22, 21, and 20)
- For the normal process, all values smaller than zero were zeroed, what have caused a vertical "off-set" on its CDF distribution.

QQplots

On the *QQplots* (Figure 19) we have observed that the original data had a much more influential heavy-tail effect when compared to the randomly generated on Cauchy, Exponential(Me), Pareto(LR), and Pareto(MLH) (Appendix A). We can observe the The original data(samples) had a much influential heavy-tail effect compared to the estimated data. We verified this by the almost horizontal line made of blue dot-marks. On the other hand, Weibull process have had a much similar *QQplot* compared to the samples. Also we can identify a right-skew for Exponential(LR), Exponential(Me) and Pareto(MLH), and a left-skew for Cauchy, Pareto(LR) and Weibull. On the other hand, the Normal distribution have presented a bi-modal behavior – The first mode is on zero (since we zeroed all values smaller than zero), and the second, is close to the the average.

AIC and BIC

The calculated values for *AIC*, *BIC*, and processes parameters for all the traces are in Table 8. We have verified in all cases that the difference between *BIC* and *AIC* for a given process was always smaller compared to the difference between the values of a same criteria but among different processes to represent the same traffic. In another words, changing the criteria matter less than changing the process distribution. In our study, *AIC* and *BIC* always have pointed to the same model ordering (Figure 25).

To compare these values of the information criteria, we calculated on the Table 9 the relative difference between *AIC* and *BIC*[7]. The table indicate that the *AIC* and *BIC* converged to a common value as the dataset size increasead[8]. This result have indicated that, if the dataset is large enough, *AIC* and *BIC* will point to the same result. As Table 9 shows, 9.54% for Exponential(LR) was the larger difference between the information criteria. For the other *pcaps*, with a greater dataset, the differences were always small than 0.001%. But, even the *lan-firewall-pcap* being the largest one, the relative difference between its *AIC* and *BIC* is small than the ones form *wan-pcap* and *lan-gateway-pcap*. However, we have that information criteria depends on the likelihood value, and this on the individual probability of each value. We note that the standard deviation for packet times in *lan-firewall-pcap* is considerably larger 22. This may be one of the possible causes for this observed behavior.

Correlation, Hurst Exponent, Mean, and Standard Deviation

In Figures 20, 21, and 23, and 22 we show the results of properties calculated for each of the simulations of the stochastic processes: Correlation, Exponent of Hurst, Mean and Standard deviation. Each figure represents a different *pcap* file. The red horizontal line represents the value of the original traffic. Analyzing the quality of *AIC* and *BIC* on *skype-pcap*, based on the figure results we see that concerning Correlation and Self-similarity it picked the right model: Weibull. Also regarding the average packet rate and dispersion, it is still one of the best choices. On regarding to the Mean, Exponential(Me) and Cauchy had the best performances. Regarding to the Standard Deviation, Exponential(Me) and Exponential(LR) had the best performance. Not by chance, Exponential(Me) *AIC* and *BIC* presented as the second best option. Comparing the results of the figures with the order suggested by AIC to BIC (Figure 25), we noticed that the best ranked models showed a good performance in the metrics in general. We show the cost function J_M, able to summarize all the primary metrics in a single number, in Figure 24. All these results are abstracted by the cost function J_M.

AIC and BIC vs. J_M

Table 8 summarizes the estimates obtained for *AIC*, *BIC*, and the stochastic process estimated parameters for all *pcap* traces. Each model order is graphically presented in Figure 25. For all *pcap* experiments, we verify that the difference between *BIC* and *AIC* for a given function is always smaller than its value among different distributions. As shown in the Table 8, *AIC* and *BIC* criteria always pointed to the same model ordering. Table 9 presents the

[7] To calculate the relative difference $r_{\%}$ shown on Table 9 we used this formula:

$$r_{\%} = \frac{|V_1 - V_2|}{\frac{(V_1+V_2)}{2}} \cdot 100 \qquad (4.18)$$

[8] from the small to the larger dataset we have *skype-pcap*, *lan-gateway-pcap*, *wan-pcap*, and *lan-firewall-pcap*

percentage difference between the obtained values. We verify that their values tend to converge when the dataset increases.

Figure 24 illustrates the cost function values for all the models on each *pcap* file. For example, for *skype-pcap*, *BIC* and *AIC* points that Weibull and Exponential(Me) are the best representation for the traffic. The cost function used for cross-validation points both as best options, along with Exponential(LR). To simplify the visualization and comparison of the differences between the rankings given by both methodologies, we created the plot 28. The Figure 28 presents a chart with the relative differences from the order of each model. Taking as a reference the position of each model given by J, we sorted them from the better to the worst (0 to 6, on the x-axis), and measured the position distance with the ones given by the information criteria. Since the worst case for this value is 6 (opposite correspondence), we draw a line on the average: the expected value in the case no positive or negative correspondence existed between both information criteria and J. Using the ϕ operator, as defined before, we can calculate the ranking delta, as explained, for the *i-th* model by:

$$\delta(m_i) = \phi(Jv, m_i) - \phi(IC, m_i) \tag{4.19}$$

Where Jv and IC are the ordered pairs vectors on models and cost functions/information criteria, from the best to the worst, respectively. We can observe that for mos to the use cases, the information criteria and the cost function had chosen models in a similar order. A hypothesis ranked well by one, tend to be ranked as good on the other. For the 28 possible study cases, 19 (68%) had the same, or one-position difference on the ranking. Also, can point that *AIC*/*BIC* tended to prioritize most of the heavy-tailed processes, such as Weibull and Pareto (except by Cauchy). It is a useful feature when the scaling and long-range characteristics of the traffic have to be prioritized by the selected model.

Finally, we point out the *AIC* and *BIC* presented a bias in favor of Pareto(MLH). Even though it was never ranked as the best model, it was always better positioned by *AIC* and *BIC* than by J. We explain this result by the fact that *AIC* and *BIC* calculation uses the model likelihood, and the Pareto(MLH) maximizes it. This effect is clear on the *lan-firewall-pcap*. On the Figure 27 we plot the cross-validation dataset, the best fitting pointed by both methods (Weibull), and the second-best indicated by J (Pareto(LR)) and by *AIC*/*BIC*(Pareto(MLH)). Even though Pareto(MLH) has a good performance representing small values, about 10% of the inter-packet times are higher than 10 seconds, a prohibitive high value. It makes the Pareto(LR) overhaul performance better.

4.6 Conclusions

In this section, we explained on details methods mentioned in Chapter 3. After, revisiting some classical works on modeling inter-packet times and packet trains, we resent the methods used to estimate packet trains periods (Algorithm 1) and make the application classification (Table 6). We then presented the use of information criteria (*AIC* and *BIC*) as tools for automatic selection of stochastic processes for inter-packet times. Since information criteria have not been tested for our study-case yet, were never tested for our study case, we developed an independent cross-validation method, based on traditional metrics for traffic validation. Since both procedures have converged to similar results, we conclude that *AIC* and *BIC* are reliable tools. However, we have to pay attention to processes parameterized the maximum likelihood method, since they tend to be prioritized, even performing poorly. We abstracted this method for automatically select stochastic processes in Algorithm 2.

We introduced and evaluated a method based on *BIC* and *AIC* for analytic selection criteria of the best stochastic process to model network traffic. We use a cross-validation methodology based on random data generation following the selected models and cost function measurements. We observe that both *AIC* or *BIC* and the cost function were able to pick the first models in the same order, corroborating to our hypothesis of Akaike and Bayesian information criteria as reliable model selectors for network inter-packet times. We can conclude that *BIC* and *AIC* are suitable alternatives to derive realistic network traffic models for synthetic traffic generation. The only cave we point is on the use of the Maximum Likelihood method, that can over-prioritized some models over others with better performance. We summarize the implementation used on SIMITAR on Algorithm 2.

Algorithm 2 stochasticModelFitting

```
1: function STOCHASTICMODELFITTING($interArrivalData, criterion$)
2:     $m = interArrivalData.size$
3:     $interArrivalData = interArrivalData + MIN\_TIME$
4:     if $m < MINIMUM\_AMOUNT\_OF\_PACKETS$ then
5:         $model\_list = \{constant\}$
6:     else
7:         $model\_list = \{weibull, pareto\_lr, pareto\_mlh, exponential\_me, exponential\_lr, normal,$
8:         $cauchy, constant\}$
9:     end if
10:    for $model$ in $model\_list$ do
11:        $model.fitting\_model(interArrivalData)$
12:    end for
13:    $model\_list.sort(criterion)$
14:    return $model\_list$
15: end function
```

5 Proof of Concept Evaluation

5.1 Testbed

As the experimental platform for validation we use Mininet-based emulated scenarios. For reproducibility purposes, Table 10 presents all relevant experimental details. All the required scripts are available on the SIMITAR code repository [Paschoalon 2019]. We use SIMITAR v0.4.2 (Eulemur rubriventer)[1], as tagged at the GitHub repository. We explore a tree topology (Figure 29), and a one-hop connection (Fig. 30). Both scenarios as SDN networks with an OpenDayLight (Beryllium) controller. We use two *pcap* files. The first is a Skype capture (*skype-pcap*), and the second (*lgw10s-pcap*) corresponds to the first ten seconds of a gateway capture[2]. Host host *h1* (IPv4 address 10.0.0.1) has generated the traffic, and was captured by TCPDump[3] on *pcap* format.

5.2 Methodology

To compare the degree of realism of the generated traffic, we use the flows' cumulative distribution function (CDF) [Sommers e Barford 2004], and the Wavelet multi-resolution analysis [Vishwanath e Vahdat 2009]. On both analysis, the closer the plots are, the more realistic is the traffic generated by SIMITAR. The flow's cumulative distribution measures the ingress of new flows over time, and it is a measure similarity and evolution of the traffic at the flow-level. On the other hand, the wavelet multi-resolution analysis can extract traffic scaling characteristics and is a measure of similarity at the packet-level. If the curve decreases, this indicates a periodicity on that time scale exists. If the curve remains approximately constant, it indicates similarity to white-noise. Finally, if the traffic has self-similar characteristics around a particular time scale, its curve increases linearly. Table 12 presents a compendium of metrics extracted from the traffic, including the Hurst exponent, which is a metric of the traffic fractal level. Self-similar processes, such as the network traffic, have its value between 0.5 and 1.0 ($0.5 < H < 1.0$) [Leland *et al.* 1994].

[1] We label the tags of SIMITAR control version on GitHub as lemurs species names (https://en.wikipedia.org/wiki/List_of_lemur_species)

[2] *skype-pcap*: available at <https://wiki.wireshark.org/SampleCaptures>, named *SkypeIRC.cap*; lgw10s-pcap avaiable at <http://tcpreplay.appneta.com/wiki/captures.html> named *bigFlows.pcap*

[3] TCPDump is a tool for monitoring and capturing network packets [TCPDUMP/LIBPCAP public repository 2019].

Table 10 – Experiments specification table

Processor	Intel(R) Core(TM) i7-4770, 8 cores, CPU @ 3.40GHz
RAM	15.5 GB
HD	1000 GB
Linux	4.8.0-59-generic
Ubuntu	Ubuntu 16.10 (yakkety)
SIMITAR	v0.4.2 (Eulemur rubriventer)
Mininet	2.3.0d1
Iperf	iperf version 2.0.9 (1 June 2016) pthreads
Libtins	3.4-2
OpenDayLight	0.4.0-Beryllium
Octave	4.0.3
Pyshark	0.3.6.2
Wireshark	2.2.6+g32dac6a-2ubuntu0.16.10
Tcudump	4.9.0
libpcap	1.7.4

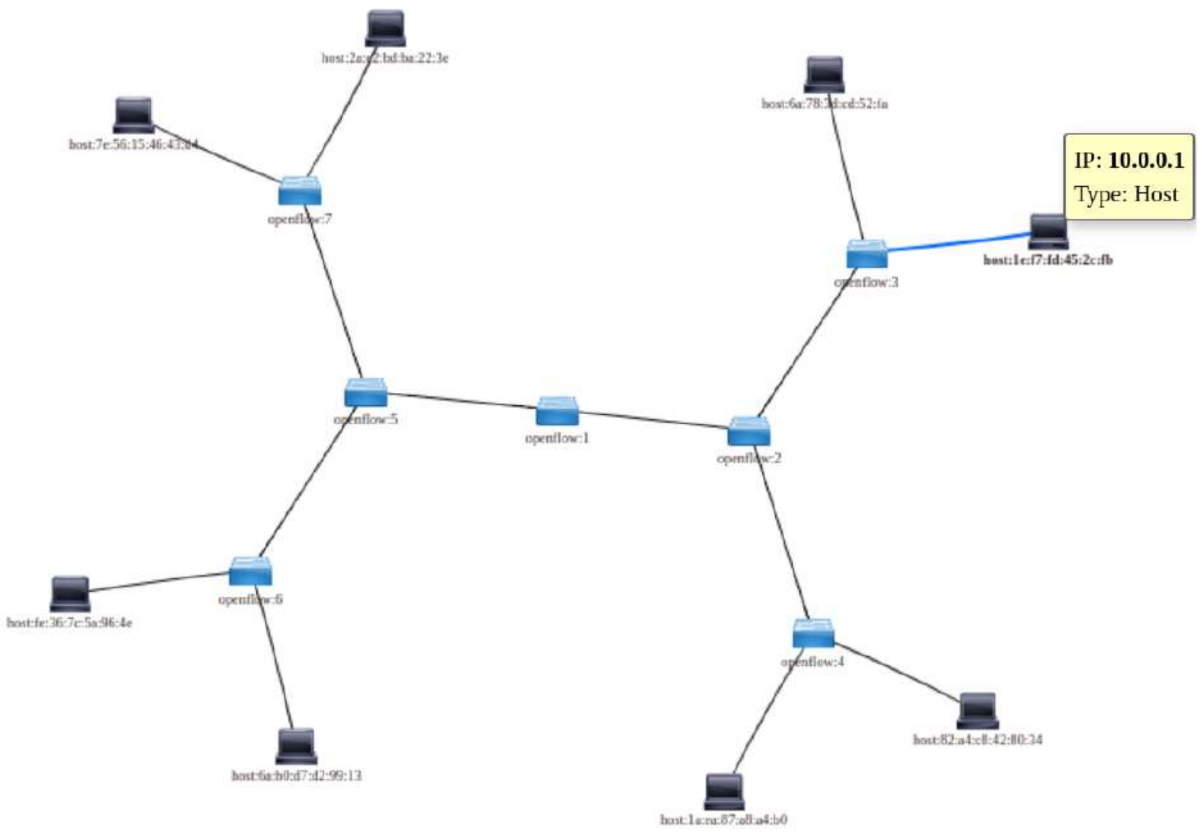


Figure 29 – Tree SDN topology emulated by mininet, and controlled by OpenDayLight Beryllium

5.3 Results

Figures 33, 34 and Table 12 show the obtained results when comparing the original and the synthetic traffic generated by SIMITAR. We also illustrate the bandwidth traffic for one

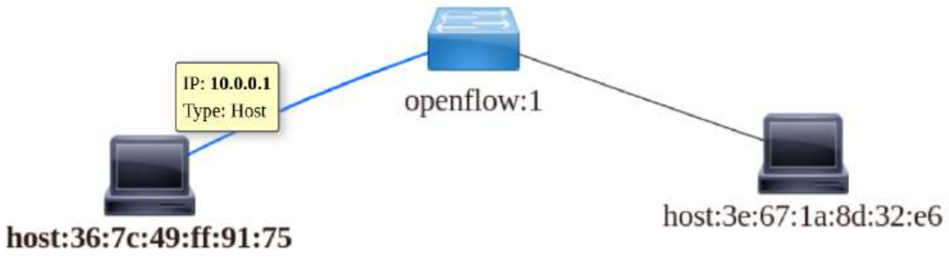


Figure 30 – Single hop SDN topology emulated by mininet, and controlled by OpenDayLight Beryllium

Table 11 – Performed validations

Metric Type	**Validations**
Packet Based Metrics	Data bit rate (kbps), Average packet rate (packets/s), Average packet size (bytes), Number of packets, Number of packets, Bandwidth over time
Flow Based Metrics	Number of flows, Flows per second, Flows CDF distributions
Fractal and Scaling Characteristics	Hurst Exponent, Wavelet Multiresolution Analisis

of the use cases in Figure 31. As we can see the generated traffics are not identical regarding bandwidth. However, both present similar fractal-like shape. The Hurst exponent of inter-packet times in every case has an error smaller than 10% compared to the original in every case, i.e., the fractal-level of each synthetic traffic is indeed similar to the original trace.

The plot of flows per second seems much more accurate (Fig. 32), since most of the peaks match. Indeed, no visual lag between the plots. Even though the generated traffic is not identical to the original, the cumulative flow distribution obtained for every study-case is almost identical on every plot (Fig. 33). The small differences on the curves result from threads and process concurrence for resources, in addition to noise from the sleep/wake processes on the thread signals. Since the operating system made the packet capture and timing, the packet capture buffer queue may have contributed as well. This result was our most significant achievement in our implementation. This result shows that our method of flow scheduling and independent traffic generation was effective and efficient in replicating the original traffic at the flow-level. However, the actual number of flows was more significant when SIMITAR used Iperf as the traffic generator API and slightly smaller when using libtins. This discrepancy can be explained because Iperf establishes additional connections to control the signaling and traffic statistics for every connection. On the other hand, with libtins, the number of flows is small, since a flow generation is aborted if the NetworkFlow flow class fails to create a new traffic flow.

The results from the Wavelet multi-resolution analysis of inter-packet times vary in each case. The time resolution chosen was ten milliseconds, and it is represented in log 2 scale. The equation can calculate the time of each time-scale j:

$$t = \frac{2^j}{100}[s] \tag{5.1}$$

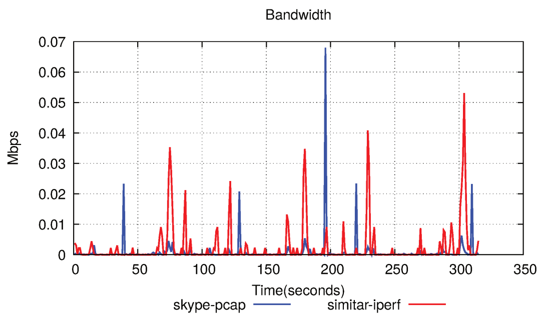
In the first case (Fig. 33a), SIMITAR reproduced Skype traffic, using Iperf in a single-hop scenario. On small time scales, both curves increased linearly, which indicates a fractal shape. However, at this point, they exhibit different slopes with the synthetic traces behaving closer to a white-noise shape. After the time scales 5 and 6 (300-600 milliseconds) scale, the error between the curves becomes almost negligible. We also observe a periodicity pattern at the time-scale of 9 seconds. Vishwanath and Vahdat [Vishwanath e Vahdat 2009] measured the same periodicity pattern; which appears to be an intrinsic characteristic of TCP traffic. We observe some periodicity at 11 and 13 time-scales (20 and 80 seconds).

In the second case (Fig. 33b), on a tree topology on small time scales, we identify behavior closer to white-noise on small scales, and similar results, but with more substantial energy levels on greater time scales. The diversity introduced by the topology and the concurrent signaling traffic caused by the other hosts and switches does explain the observed behavior since node signaling tends to be more randomized than user-generated traffic. Indeed, as we can see in Table 10, there are two hundred more packets captured on the client interface in the tree topology compared to the one-hop scenario.

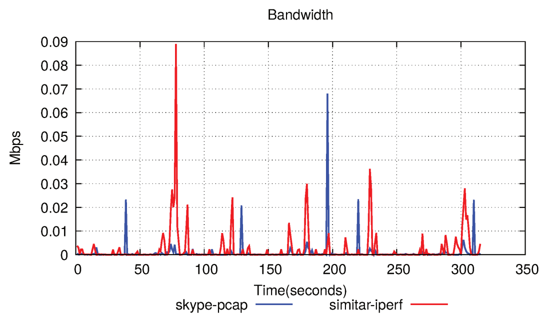
In the last two plots (Figures 33c and 33d), where we use libtins as the packet crafter, the energy level is higher, and the curves are less correlated. SIMITAR, in the current implementation, is not modeling inter-packet with libtins and sends packets as fast as possible, which explains this discrepancy. However, in the last scenario, due to the higher average throughput, the observed performance was better.

Table 12 – Sumary of results comparing the original traces (italic) and te traffic generated by SIMITAR, with the description of the scenario.

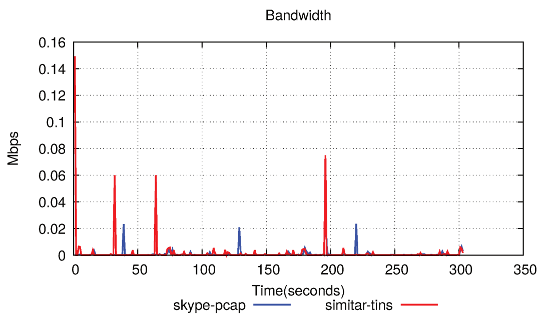
	skype-pcap	skype, one-hop, iperf	skype, tree, iperf	skype, one-hop, libtins	*lgw10s-pcap*	lgw10s, one-hop, libtins
Hurst Exponent	0.601	0.618	0.598	0.691	0.723	0.738
Data bit rate (kbps)	7	19	19	12	7252	6790
Average packet rate (packets/s)	3	4	5	6	2483	2440
Average packet size (bytes)	260,89	549,05	481,14	224,68	365,00	347,85
Number of packets	1071	1428	1604	2127	24 k	24 k
Number of flows	167	350	325	162	3350	3264



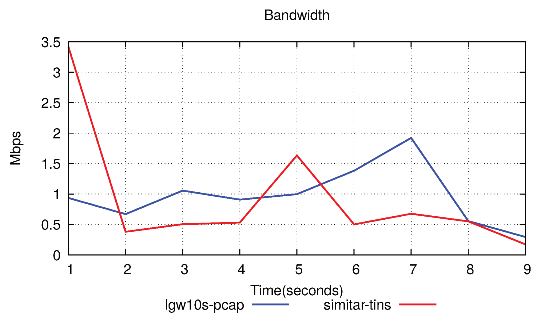
(a) *Iperf, single-hop, skype-pcap*



(b) *Iperf, tree, skype-pcap*

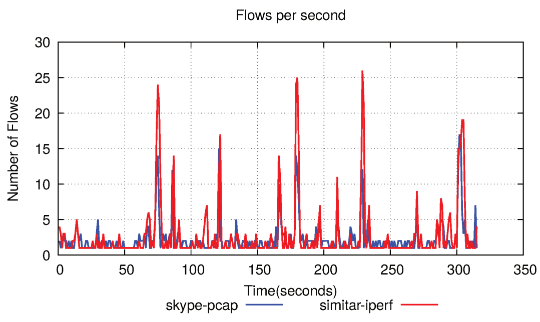


(c) *libtins, single-hop, skype-pcap*

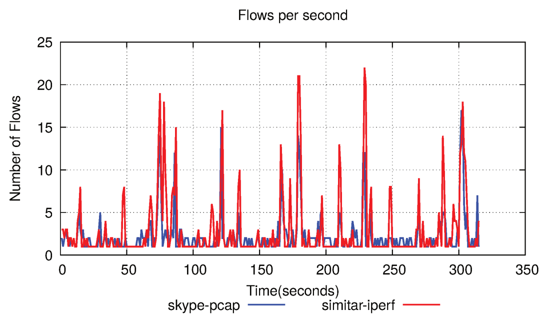


(d) *libtins, single-hop, langw10s-pcap*

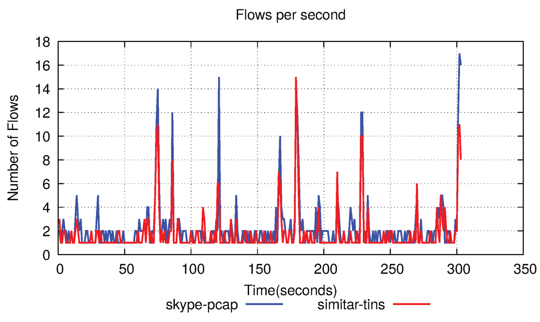
Figure 31 – Traces bandwidth.



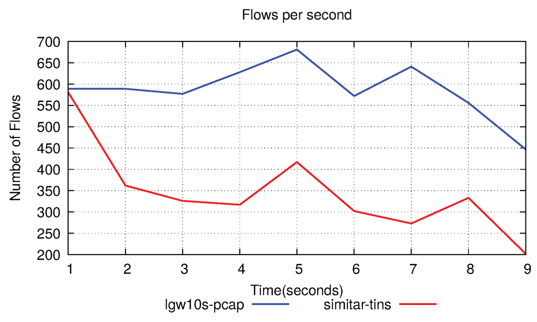
(a) *Iperf, single-hop, skype-pcap*



(b) *Iperf, tree, skype-pcap*

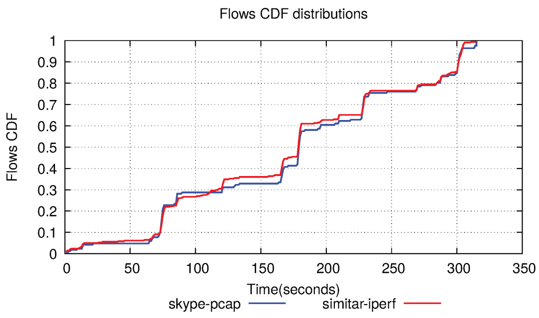


(c) *libtins, single-hop, skype-pcap*

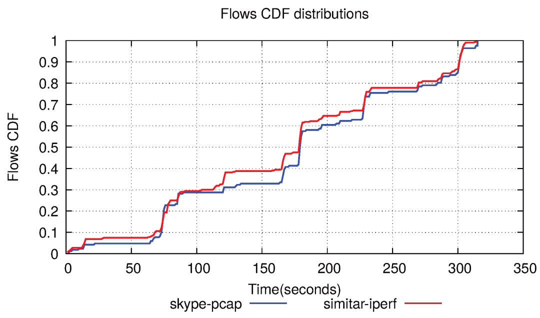


(d) *libtins, single-hop, langw10s-pcap*

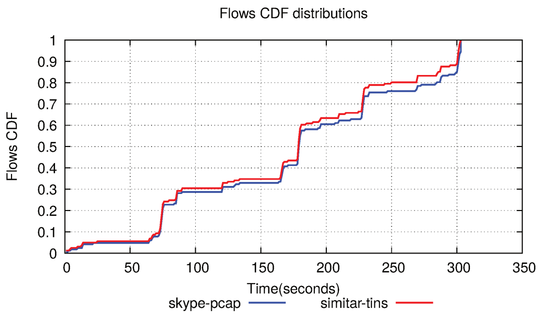
Figure 32 – Flow per seconds



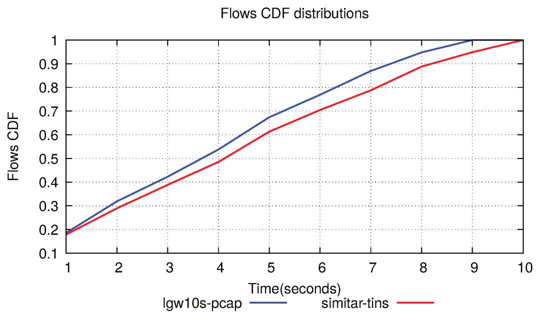
(a) *Iperf, single-hop, skype-pcap*



(b) *Iperf, tree, skype-pcap*

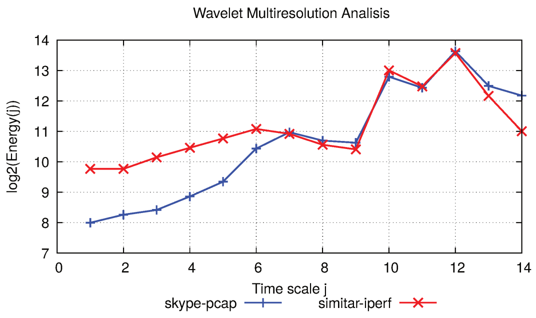


(c) *libtins, single-hop, skype-pcap*

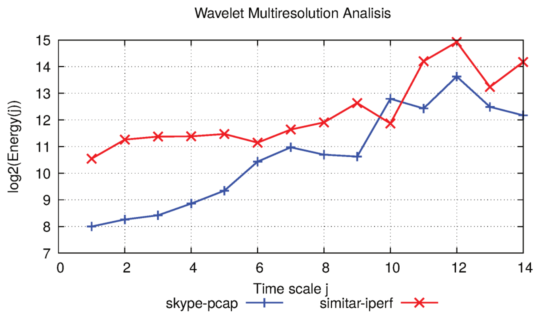


(d) *libtins, single-hop, langw10s-pcap*

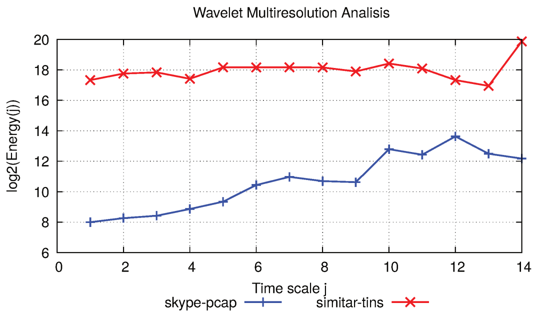
Figure 33 – Flows cumulative distributions.



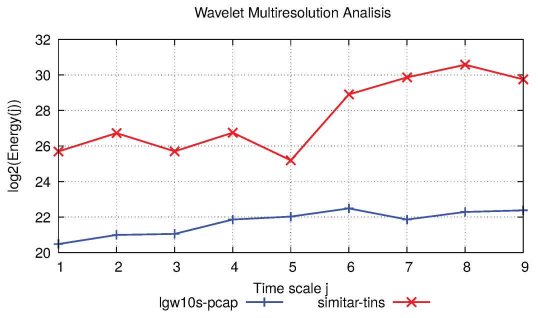
(a) *Iperf, single-hop, skype-pcap*



(b) *Iperf, tree, skype-pcap*



(c) *libtins, single-hop, skype-pcap*



(d) *libtins, single-hop, langw10s-pcap*

Figure 34 – Wavelet multiresolution energy analysis.

5.4 Conclusions

We present SIMITAR as a tool and methodology to attend the evolving needs of rich and realistic network traffic experiments working at both flow- and packet-level. At the flow-level, our methodology already achieves high fidelity results. The cumulative distribution of flows is almost identical in each case. From the perspective of benchmarking of a middle-boxes or SDN switches, this is a valuable result, since their performance, especially in SW implementations, largely depend on the number and characteristics of the stimuli flows. However, because of packets exchanged by background signaling connections, the traffic generated by Iperf, even following the same cumulative flow distribution, ended up creating more streams then expected.

At the packet level, the current results with Iperf replicate with high accuracy the scaling characteristics of the first traffic, and the number of generated packets are not far than the expected. Despite all identified optimization, the results are more than satisfactory and prove the potential of the proposed methodology. At the flow-level, our results are at least as good as those achieved by best-of-breed related work like Harpoon and Swing. On the scaling characteristics, using lightweight traces, the results have been of comparable in quality.

6 Future Work

Table 13 lists a set of ideas for future work. The sections A to D comprehend topics on the evolution of SIMITAR. Section E mentions new ideas of research that can contribute to SIMITAR evolution. Also, some items can be a starting point for new tools.

Table 13 – Overview of future work topics

A	Performance	
	1	Modeling Optimizations
	2	TinyFlows and flow merging
	3	Smarter Flow Scheduler and thread management
	4	DPDK KNI Interfaces
	5	Multi-thread C++ Sniffer
B	Tool Support	
	1	Inter-packet times on TinsFlow
	2	D-ITG Flow Generator: DitgFlow
	3	DPDK Flow Generator: DpdkFlow
	4	Ostinato Flow Generator: OstinatoFlow
	5	ZigBee protocol Support
C	Calibration	
	1	min_time
	2	min_on_time
	3	session_cut_time
D	New Components	
	1	Traffic Measurer
	2	Pcap files crafter
	3	Python/Lua Flow Generator
E	New Research Topics	
	1	Automated Selection of Inter-packet times models 2.0
	2	How how to craft malicious flows?
	3	Markovian-based traffic models
	4	Envelope-process based traffic models
	5	Relationship between Hurst and Hölder exponent, and stochastic processes.
	6	Hurst-exponent feedbakc controll system for ON/OFF times
	7	Traffic generation based on Generative Adversarial Networks (GANs)
	8	Realistic WAN, WiFi, and IoT traffic
	9	SIMITAR vs Harpoon
	10	How well traffic generators simulate reproduce stochastic processes?
	11	Traffic Generator Tools Survey

6.1 Performance

6.1.1 Modeling optimizations

[A.1] The primary issue of SIMITAR now is optimizing data processing for creating the Compact Trace Descriptor. The performance becomes an issue when processing large *pcap* files with more dozens of thousands of flows. The time expended for processing traces, in this case, is exceeding tens of hours. In the current implementation, the linear regression execution is mono-thread, and the stop criterion is just the number of iterations. Parallel processing, and creating stop criteria based on convergence in addition to the number of iterations will improve the performance, along with some code optimizations. Make the XML less verbose will help as well.

6.1.2 TinyFlows and flow merging

[A.2] Crating an option for merging flows is a possibility to improve the performance of traffic with several thousands of flows and Gigabits of bandwidth, such as from WAN captures. A merge criterion, for example, is to consider just network headers on flow's classification. Also, the usage of simpler models for flows with a small number of packets (a "TinyFlow"), would improve the processing speed.

6.1.3 Smarter Flow scheduler and thread management

[A.3] Currently, SIMITAR instantiates all the flow threads once the traffic generations start. A smarter traffic generation where SIMITAR instantiates each thread when the traffic, and join when it is inactive should reduce the overhead for traces with a large number of flows.

6.1.4 DPDK KNI Interfaces

[A.4] One possibility to improve the traffic generation performance issue DPDK Kernel NIC Interface (KNI interfaces) 2. DPDK KNI interfaces allow applications from the user's space to interact with DPDK ports. In this way, we may achieve a faster packet processing.

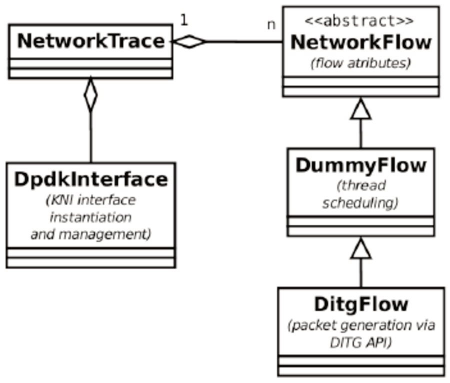


Figure 35 – Usage of DPDK KNI interfaces.

6.1.5 Multi-thread C++ Sniffer

[A.5] Implementing a C/C++ multi-thread sniffer will improve the processing time of packets.

6.2 Tool Support

6.2.1 Inter-packet times on TinsFlow

[B.1] SIMITAR's current implementation using libtins to generate the packets does not model inter-packet times. Modeling this feature will improve the scaling characteristics of libtins traffic.

6.2.2 D-ITG, Ostinato, and DPDK Flow Generators: DitgFlow, OstinatoFlow, DdpkFlow

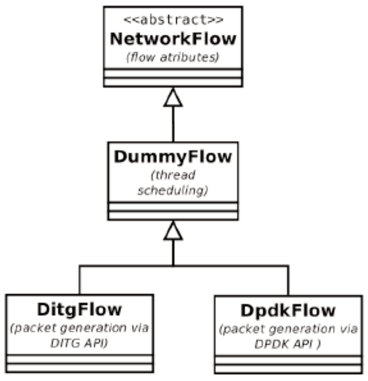


Figure 36 – DddkDlow and DitgFlow

[B.2-4] Expand SIMITAR to other traffic generator tools (Figure 36). D-ITG offers many stochastic functions for customization of inter-packet times, Ostinato provides a rich set of headers and protocols, and DPDK a high performance on packet generation. Each tool can offer a different result on traffic generation, each with their benefits.

6.2.3 ZigBee protocol Support

[B.5] Finally, to apply SIMITAR on IoT scenarios, we will have to provide support for new protocols, such as ZigBee [Ramya *et al.* 2011].

6.3 Calibration

6.3.1 `min_time`

[C.1] Calibrate the constant `DataProcessor::min_time`: smallest time considered for inter-packet times. We use this value to avoid inter-packet times equals to zero due to the sniffer resolution. Today, this value is $5e^{-8}$.

6.3.2 `min_on_time`

[C.2] Calibrate the constant `DataProcessor::m_min_on_time`: this value controls the small ON time that a file can have. It can change the generated traffic precision. Currently, this value is 0.1s.

6.3.3 `session_cut_time`

[C.3] Calibrate the constant `DataProcessor::m_session_cut_time`. `calcOnOff` uses this value to defines whatever a file transference still active or has ended. This constant affects performance on traffic realism.

6.4 New Components

6.4.1 Traffic Measurer

[D.1] Develop a component able to extract useful QoS information from the generated traffic is essential on the applicability and utility of our tool (Figure 37). This can be done by:

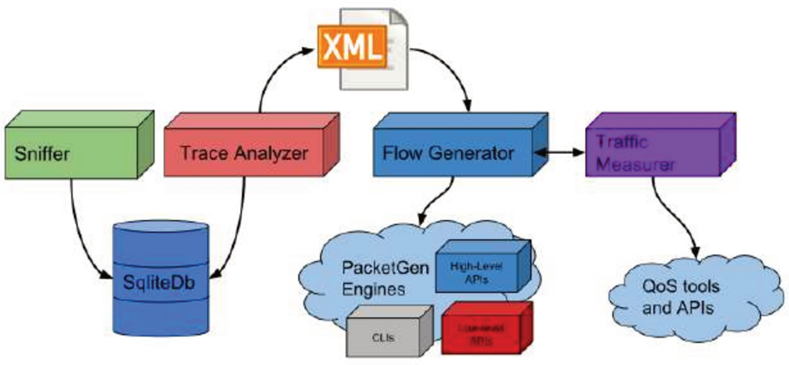


Figure 37 – Component for measurement of traffic statistics: packet-loss, throughput, available bandwidth, delay, RTT, and jitter.

- Counting the transmitted and received packets, to estimate the ***packet-loss*** and the ***throughput***.
- Apply techniques of passive measurement of ***available bandwidth***, such as the ones used by Pathload [Pathload – measurement tool for the available bandwidth of network paths 2006] and pathChirp [Ribeiro *et al.* 2003] [pathChirp 2003].
- Create signaling channels to estimate the ***delay***, ***RTT*** and ***jitter***.

6.4.2 Pcap files crafter

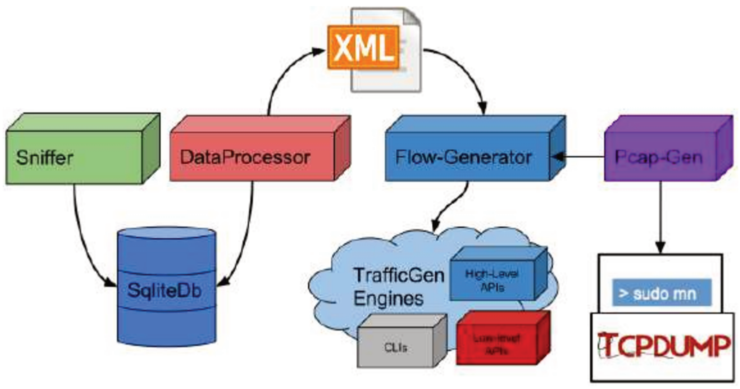


Figure 38 – Using SIMITAR for generation synthetic *pcap* files, CTD files: a component schema

[D.2] PcapGen, a *pcap* generator tool: Create a component capable of generating synthetic *pcap* files, using Compact Trace Descriptors files, using TCPDump [TCPDUMP/LIBPCAP public repository 2019] and Mininet [Mininet – An Instant Virtual Network on your Laptop (or other PC) 2019]. We present a diagram of this idea in Figure 38. This expansion would enable SIMITAR to work as trace library for *pcap-based* benchmark tools.

6.4.3 Python/Lua Flow Generator

[D.3] Currently, SIMITAR only enables the programming of flow traffic generation in C++. Adding Python and Lua support for the Flow Generator component, we can allow expansion for Python/Lua traffic generation APIs (such as Ostinato and MoonGen APIs), without creating C++ wrappers.

6.5 New Research Topics

6.5.1 Automated Selection of Inter-packet times models 2.0

[E.1] Expand our work made on the validation of information criteria on the automated selection of stochastic models.

- A deeper analysis of the impact on the maximum likelihood method in comparison to the others;
- An analysis of the effectiveness of each parameterization method, and the best performance of each metric of quality measurement: correlation, mean inter-packet time and Hurst exponent;
- Use of new stochastic methods functions, such as Log-Normal, Gamma, Poison, Binomial, Beta, and Chi-squared;
- Use of Markovian-chain and Envelope processes;
- Use other information criteria, such as AICc, MDL, nMDL [Tune *et al.* 2016] and DIC [Spiegelhalter *et al.* 2014].

6.5.2 How how to craft malicious flows?

[E.2] Research features used on by intrusion detection and intrusion prediction systems, and develop a method to mimic malicious flows, creating a "MaliciousFlow" model. At the same time, improve and evolve the current flow modeling, to ensure regular flows are not labeled as malicious flows by the same systems. In that way, SIMITAR will be able to craft malicious traces, an important achievement in network security research.

6.5.3 Envelope and Markovian-based traffic models

[E.3-4] Evolve SIMITAR traffic model, and try the application of Markovian and Envelope processes on the modeling of inter-packet times and ON/OFF times.

6.5.4 Fractal and multi-fractal modeling: models, Hurst exponent and Hölder exponent.

[E.5] Another proposal is to deepen the studies on Fractal and multi-fractal modeling.

- Study a larger set of fractal models, including the processes mentioned above: Envelope and Markovian.
- Study the Hölder exponent [Yu e Qi 2011], a generalization over time of the Hurst exponent.
- Study the viability of parameterizing a process to have a specific value of Hust exponent. In other words, have the Hust exponent as a constraint for the model.

6.5.5 Hurst-exponent feedback control system for ON/OFF times

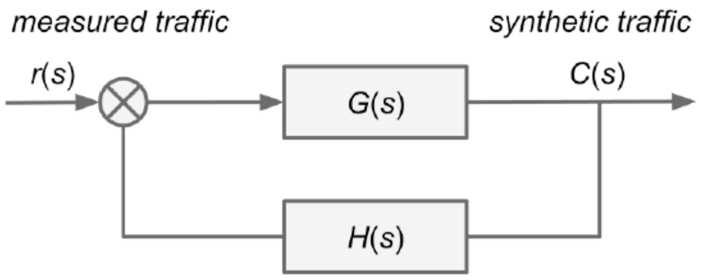


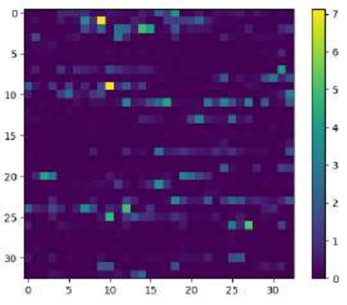
Figure 39 – Schematic of a feedback control system applied on synthetic traffic generation.

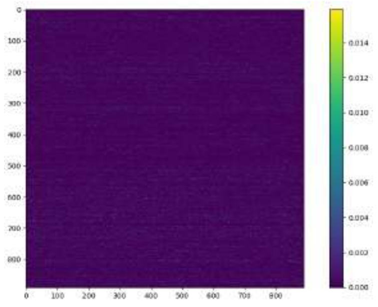
[E.6] Study the viability of the application of a feedback system to control the parameters of the synthetic traffic; for example, Mean inter-packet time and Hurst Exponent (Figure 39).

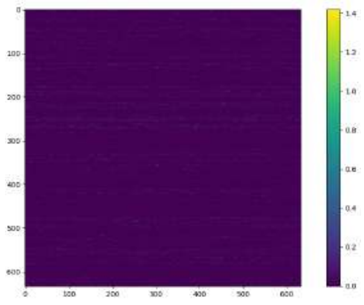
6.5.6 Traffic generation based on Generative Adversarial Networks (GANs)

[E.7] Other promising technology on controlling features can be the usage of Generative Adversarial Networks, also called GANs [Huang *et al.* 2018]. GAN is a type of neural network used usually used on image synthesis, as we can see in Figure 40. We could apply the GAN to generate matrixes of features: inter-packet times, flows, protocols, packet sizes and so on (Figure 41). Then, apply this matrix of features to synthesize network traffic.

Figure 40 – Example of GANs application. GANs are commonly used for image synthesis. Source: [Wu *et al.* 2017].


(a) *skype-pcap*


(b) *gateway-pcap*


(c) *firewall-pcap*

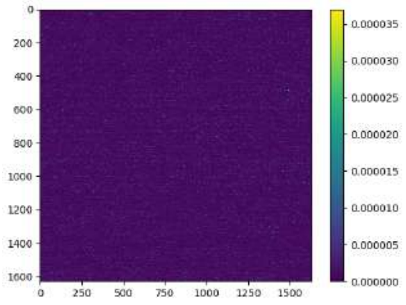

(d) *wan-pcap*

Figure 41 – Color-map of inter-packet times from pcaps used on Chapter 4.

6.5.7 Realistic WAN, Wifi and IoT traffic

[E.8] Expand the support for protocols and API of traffic generation to create synthetic traffic in different environments:

- Using DPDK, and improving the processing performance can be able to recreate WAN synthetic traffic.
- Support for new protocols, such as IEEE 802.11 (Wifi) and ZigBee, to create synthetic Wifi and IoT traffic.

6.5.8 SIMITAR vs Harpoon

[E.9] A "trial by fire", after improving the tools with these new features, compare the performance on realism and throughput of SIMITAR and Harpoon.

6.5.9 How well traffic generators simulate reproduce stochastic processes?

[E.10] A parallel research topic. Evaluate how close traffic generators that use the stochastic process to model inter-packet times can reproduce the theoretically expected values.

6.5.10 Traffic Generator Tools Survey

[E.11] The idea is to consolidate the already collected knowledge on traffic generators (Chapter 2 and Appendix C), with other topics of relevance in these subjects, inspired in the same structure used by other surveys.

7 Final Conclusions

In this dissertation, we discuss our proposed process of conceiving, researching, definition and development and validation of a realistic traffic generator to fill gaps in the state of the art, achieving results comparable to open-source projects. We followed a spiral process, as defined in the introduction (Figure 1, Chapter 1). After defining the scope of our research, we researched the literature and could not find any publicly available traffic generation solution, which at the same time is:

- Auto-configurable;
- Produces Realistic Traffic;
- Enables Traffic Customization;
- Extensible.

At this point, we identified a set of requirements, we designed an architecture using UML, and we implemented an initial prototype. Next, we continue to search for more techniques and methods that could be applied to solve our tasks and improve the results. We researched many topics that at the end did not fit our intentions, such as Machine Learning and Neural Networks. However, others such as Linear regression, maximum likelihood, and information criteria were indeed satisfactory. Many ideas were inspired by previous researches, such as Harpoon, Swing, sourcesOnOff, and LegoTG. Again, we planned a methodology, procedure, and validate these ideas, and codded. SIMITAR evolution continued with incremental upgrades until we reach the process presented in the Chapter 3. Also, along with the developed software, we have documented our findings over the literature and open-source community. The more-relevant subjects for the understanding of our research; to mention (i) traffic generator tools, (ii) network traffic modeling; (iii) validation of traffic generators have been documented in Chapter 2. Although, whether necessary, concepts were introduced in other parts of the text – especially Chapter 4. Also, we have saved the cut-content on Appendix C.

In Chapter 4, we achieve a significant contribution to our work, which shows that the information criteria AIC and BIC are efficient analytic methods for select models for inter-packet times. Both can infer a good model, even evaluated according to different types of metrics, without any simulation. Also, we show evidence that for Ethernet traffic of data, choosing AIC and BIC make almost no difference. As far as is our knowledge, this is a complete study on the use of AIC and BIC on inter-packet times modeling of Ethernet traffic.

Our tests performed in Chapter 4 intend to focus on packet-level, flow-level, and scaling metrics. The results were notably good at the flow-level. SIMITAR were able to replicate the flow-cumulative distribution with high accuracy, and with libtins, the number of flows as well. When Iperf was the traffic generator API, the number of flows was more substantial, because it establishes additional connections for signaling. On scaling and packet metrics, the results still have to be improved. Iperf as the traffic generator tool, still being limited, has proved to be efficient on replication the scaling characteristics for the Skype traffic. Since it establishes socket connections to generate the traffic, we believe that this fact makes it accurate on replication traffic from applications.

In the end, we were able to implement a functional implementation of the solution proposed in Chapter 1.

- SIMITAR was able to create synthetic traffic, based on our models, replicating wich good results flow-level characteristics, fractal and scaling characteristics as well;
- SIMITAR is auto-configurable, sparing user time on conceiving parameterization, validation, and implementation of a good traffic model;
- SIMITAR enables flexible traffic customization. The user may program his custom traffic, creating a custom Compact Trace Descriptor, without having to use any Traffic Generation API;
- SIMITAR decouples the modeling and traffic generation layer completely. SIMITAR is fully extensible, relying on the implementation of just a C++ class. Our current implementation use two very distinct packet-crafters: libtins, a C++ library designed for the application of sniffers and traffic generators, and Iperf, a traffic generator used to bandwidth measurements.

Finally, we identified a list of improvements and future works (Chapter 6), aiming the development of the software, including, performance better processing time and packet generation, and the realism on the traffic generated. The higher bottleneck of the project resides on processing performance. For processing huge *pcap* files, it still takes a prohibitive amount of time. The results on realism, even with much room for improvement, already have a good quality. With the proposed future works, we believe to overcome the identified limitations.

Bibliography

ABRY, P.; VEITCH, D. Wavelet analysis of long-range-dependent traffic. *IEEE Transactions on Information Theory*, v. 44, n. 1, p. 2–15, Jan 1998. ISSN 0018-9448.

ANTICHI, G.; PIETRO, A. D.; FICARA, D.; GIORDANO, S.; PROCISSI, G.; VITUCCI, F. Bruno: A high performance traffic generator for network processor. In: *2008 International Symposium on Performance Evaluation of Computer and Telecommunication Systems*. [S.l.: s.n.], 2008. p. 526–533.

Antichi, G.; Shahbaz, M.; Geng, Y.; Zilberman, N.; Covington, A.; Bruyere, M.; Mckeown, N.; Feamster, N.; Felderman, B.; Blott, M.; Moore, A. W.; Owezarski, P. Osnt: open source network tester. *IEEE Network*, v. 28, n. 5, p. 6–12, Sep. 2014. ISSN 0890-8044.

APPROPRIATE Uses For SQLite. 2019. <https://www.sqlite.org/whentouse.html>. (Accessed on 04/17/2019).

BARFORD, P.; CROVELLA, M. Generating representative web workloads for network and server performance evaluation. *SIGMETRICS Perform. Eval. Rev.*, ACM, New York, NY, USA, v. 26, n. 1, p. 151–160, jun. 1998. ISSN 0163-5999. Disponível em: <http://doi.acm.org/10.1145/277858.277897>.

BARTLETT, G.; MIRKOVIC, J. Expressing different traffic models using the legotg framework. In: *2015 IEEE 35th International Conference on Distributed Computing Systems Workshops*. [S.l.: s.n.], 2015. p. 56–63. ISSN 1545-0678.

BOOCH, G. *The unified modeling language user guide*. Upper Saddle River, NJ: Addison-Wesley, 2005. ISBN 0321267974.

BOTTA, A.; DAINOTTI, A.; PESCAPE, A. Do you trust your software-based traffic generator? *IEEE Communications Magazine*, v. 48, n. 9, p. 158–165, Sept 2010. ISSN 0163-6804.

BOTTA, A.; DAINOTTI, A.; PESCAPé, A. A tool for the generation of realistic network workload for emerging networking scenarios. *Computer Networks*, v. 56, n. 15, p. 3531 – 3547, 2012. ISSN 1389-1286. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1389128612000928>.

BOX, G. E. P.; JENKINS, G. M.; REINSEL, G. C. *Time Series Analysis: Forecasting and Control*. 3. ed. Englewood Cliffs, NJ: Prentice Hall, 1994.

BURTON, R. *The book of the sword : a history of daggers, sabers, and scimitars from ancient times to the modern day*. New York: Skyhorse Publishing, Inc, 2014. 130 p. ISBN 978-1626364011.

C++ Programming: Code patterns design - Wikibooks, open books for an open world. 2019. <https://en.wikibooks.org/wiki/C%2B%2B_Programming/Code/Design_Patterns>. (Accessed on 04/21/2019).

CAI, Y.; LIU, Y.; GONG, W.; WOLF, T. Impact of arrival burstiness on queue length: An infinitesimal perturbation analysis. In: *Proceedings of the 48h IEEE Conference on Decision* *and Control (CDC) held jointly with 2009 28th Chinese Control Conference*. [S.l.: s.n.], 2009. p. 7068–7073. ISSN 0191-2216.

CAIDA Center for Applied Internet Data Analysis. 2019. http://www.caida.org/home/. [Online; accessed January 11th, 2017].

CASTRO, E.; KUMAR, A.; ALENCAR, M. S.; E.FONSECA, I. A packet distribution traffic model for computer networks. In: *Proceedings of the International Telecommunications Symposium – ITS2010*. [S.l.: s.n.], 2010.

CEVIZCI, I.; EROL, M.; OKTUG, S. F. Analysis of multi-player online game traffic based on self-similarity. In: *Proceedings of 5th ACM SIGCOMM Workshop on Network and System Support for Games*. New York, NY, USA: ACM, 2006. (NetGames '06). ISBN 1-59593-589-4. Disponível em: <http://doi.acm.org/10.1145/1230040.1230093>.

Covington, G. A.; Gibb, G.; Lockwood, J. W.; Mckeown, N. A packet generator on the netfpga platform. In: *2009 17th IEEE Symposium on Field Programmable Custom Computing Machines*. [S.l.: s.n.], 2009. p. 235–238.

CSIKOR, L.; SZALAY, M.; SONKOLY, B.; TOKA, L. Nfpa: Network function performance analyzer. In: *Network Function Virtualization and Software Defined Network (NFV-SDN), 2015 IEEE Conference on*. [S.l.: s.n.], 2015. p. 15–17.

D-ITG, Distributed Internet Traffic Generator. 2015. http://traffic.comics.unina.it/software/ITG/. [Online; accessed May 14th, 2016].

DATE, C. J. *An introduction to database systems*. Boston: Pearson/Addison Wesley, 2004. ISBN 978-0321197849.

DPDK – Data Plane Development Kit. 2019. http://dpdk.org/. [Online; accessed May 14th, 2016].

EMMERICH, P.; GALLENMüLLER, S.; RAUMER, D.; WOHLFART, F.; CARLE, G. Moongen: A scriptable high-speed packet generator. In: *Proceedings of the 2015 ACM Conference on Internet Measurement Conference*. New York, NY, USA: ACM, 2015. (IMC '15), p. 275–287. ISBN 978-1-4503-3848-6. Disponível em: <http://doi.acm.org/10.1145/2815675.2815692>.

FENG, W.-c.; GOEL, A.; BEZZAZ, A.; FENG, W.-c.; WALPOLE, J. Tcpivo: A high-performance packet replay engine. In: *Proceedings of the ACM SIGCOMM Workshop on Models, Methods and Tools for Reproducible Network Research*. New York, NY, USA: ACM, 2003. (MoMeTools '03), p. 57–64. ISBN 1-58113-748-6. Disponível em: <http://doi.acm.org/10.1145/944773.944783>.

FIELD, A. J.; HARDER, U.; HARRISON, P. G. Measurement and modelling of self-similar traffic in computer networks. *IEE Proceedings - Communications*, v. 151, n. 4, p. 355–363, Aug 2004. ISSN 1350-2425.

FIORINI, P. M. On modeling concurrent heavy-tailed network traffic sources and its impact upon qos. In: *1999 IEEE International Conference on Communications (Cat. No. 99CH36311)*. [S.l.: s.n.], 1999. v. 2, p. 716–720 vol.2.

GARCIA, L. M. Programming with libpcap - sniffing the network from our own application. *Hakin 9.*, v. 3, n. 2, p. 38–46, jan 2008. ISSN 1733-7186. Disponível em: <http://recursos.aldabaknocking.com/libpcapHakin9LuisMartinGarcia.pdf>.

GEN_SEND, gen_recv: A Simple UDP Traffic Generater Application. 2019. http://www.citi.umich.edu/projects/qbone/generator.html. [Online; accessed January 11th, 2017].

GENSYN - generator of synthetic Internet traffic. 2019. http://www.item.ntnu.no/people/personalpages/fac/poulh/gensyn. [Online; accessed May 14th, 2016].

GETTING Started with Pktgen. 2015. http://pktgen.readthedocs.io/en/latest/getting_started.html. [Online; accessed May 14th, 2016].

Ghobadi, M.; Salmon, G.; Ganjali, Y.; Labrecque, M.; Steffan, J. G. Caliper: Precise and responsive traffic generator. In: *2012 IEEE 20th Annual Symposium on High-Performance Interconnects*. [S.l.: s.n.], 2012. p. 25–32. ISSN 1550-4794.

GRIGORIU, M. Dynamic systems with poisson white noise. *Nonlinear Dynamics*, v. 36, n. 2, p. 255–266, 2004. ISSN 1573-269X. Disponível em: <http://dx.doi.org/10.1023/B:NODY.0000045518.13177.3c>.

HAIGHT, F. A. *Handbook of the Poisson Distribution*. New York: John Wiley & Son, 1967.

HAN, B.; GOPALAKRISHNAN, V.; JI, L.; LEE, S. Network function virtualization: Challenges and opportunities for innovations. *Communications Magazine, IEEE*, v. 53, n. 2, p. 90–97, Feb 2015. ISSN 0163-6804.

HEEGAARD, P. Gensyn-a java based generator of synthetic internet traffic linking user behaviour models to real network protocols. *ITC Specialist Seminar on IP Traffic Measurement, Modeling and Management*, 01 2000.

HTTPERF(1) - Linux man page. 2019. https://linux.die.net/man/1/httperf. [Online; accessed January 11th, 2017].

HUANG, H.; YU, P. S.; WANG, C. An introduction to image synthesis with generative adversarial nets. *CoRR*, abs/1803.04469, 2018. Disponível em: <http://arxiv.org/abs/1803.04469>.

HUANG, M.; WANG, X.; LI, K.; DAS, S. K. A comprehensive survey of network function virtualization. *Computer Networks*, v. 133, p. 212–262, 2018.

HUANG, P.; FELDMANN, A.; WILLINGER, W.; ARCHIVES, T. P. S. U. C. A non-intrusive, wavelet-based approach to detecting network performance problems. unknown, 2001. Disponível em: <http://citeseer.ist.psu.edu/453711.html>.

INTRODUCTION to Mininet · mininet/mininet Wiki. 2019. <https://github.com/mininet/mininet/wiki/Introduction-to-Mininet#what>. (Accessed on 04/21/2019).

IPERF - The network bandwidth measurement tool. 2019. https://iperf.fr/. [Online; accessed May 14th, 2016].

JPERF. 2015. https://github.com/AgilData/jperf. [Online; accessed May 14th, 2016].

JPERF. 2019. https://sourceforge.net/projects/jperf/. [Online; accessed Apr 14th, 2019].

JU, F.; YANG, J.; LIU, H. Analysis of self-similar traffic based on the on/off model. In: *2009 International Workshop on Chaos-Fractals Theories and Applications*. [S.l.: s.n.], 2009. p. 301–304.

KHAYARI, R. E. A.; RUCKER, M.; LEHMANN, A.; MUSOVIC, A. Parasyntg: A parameterized synthetic trace generator for representation of www traffic. In: *Performance Evaluation of Computer and Telecommunication Systems, 2008. SPECTS 2008. International Symposium on*. [S.l.: s.n.], 2008. p. 317–323.

KLEBAN, S. D.; CLEARWATER, S. H. Hierarchical dynamics, interarrival times, and performance. In: *Supercomputing, 2003 ACM/IEEE Conference*. [S.l.: s.n.], 2003. p. 28–28.

KOLAHI, S. S.; NARAYAN, S.; NGUYEN, D. D. T.; SUNARTO, Y. Performance monitoring of various network traffic generators. In: *Computer Modelling and Simulation (UKSim), 2011 UkSim 13th International Conference on*. [S.l.: s.n.], 2011. p. 501–506.

KREUTZ, D.; RAMOS, F.; VERISSIMO, P. E.; ROTHENBERG, C. E.; AZODOLMOLKY, S.; UHLIG, S. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, v. 103, n. 1, p. 14–76, Jan 2015. ISSN 0018-9219.

KRONEWITTER, F. D. Optimal scheduling of heavy tailed traffic via shape parameter estimation. In: *MILCOM 2006 - 2006 IEEE Military Communications conference*. [S.l.: s.n.], 2006. p. 1–6. ISSN 2155-7578.

Ku, C.; Lin, Y.; Lai, Y.; Li, P.; Lin, K. C. Real traffic replay over wlan with environment emulation. In: *2012 IEEE Wireless Communications and Networking Conference (WCNC)*. [S.l.: s.n.], 2012. p. 2406–2411. ISSN 1558-2612.

KUROSE, J. *Computer networking : a top-down approach*. Boston: Pearson, 2017. ISBN 9780133594140.

KUSHIDA, T.; SHIBATA, Y. Empirical study of inter-arrival packet times and packet losses. In: *Proceedings 22nd International Conference on Distributed Computing Systems Workshops*. [S.l.: s.n.], 2002. p. 233–238.

KUTE – Kernel-based Traffic Engine. 2007. http://caia.swin.edu.au/genius/tools/kute/. [Online; accessed May 14th, 2016].

LELAND, W. E.; TAQQU, M. S.; WILLINGER, W.; WILSON, D. V. On the self-similar nature of ethernet traffic (extended version). *IEEE/ACM Transactions on Networking*, v. 2, n. 1, p. 1–15, Feb 1994. ISSN 1063-6692.

LIBTINS: packet crafting and sniffing library. 2019. http://libtins.github.io/. [Online; accessed May 30th, 2017].

MARKOVITCH, N. M.; KRIEGER, U. R. Estimation of the renewal function by empirical data-a bayesian approach. In: *Universal Multiservice Networks, 2000. ECUMN 2000. 1st European Conference on*. [S.l.: s.n.], 2000. p. 293–300.

MAWI Working Group Traffic Archive. 2019. http://mawi.wide.ad.jp/mawi/. [Online; accessed January 11th, 2017].

MELO, C. A.; FONSECA, N. L. da. Envelope process and computation of the equivalent bandwidth of multifractal flows. *Computer Networks*, v. 48, n. 3, p. 351 – 375, 2005. ISSN 1389-1286. Long Range Dependent Traffic. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1389128604003305>.

MINERVA ABYI BIRU, D. R. R. *Towards a definition of the Internet of Things (IoT)*. 2015. <https://iot.ieee.org/images/files/pdf/IEEE_IoT_Towards_Definition_Internet_of_Things_Revision1_27MAY15.pdf>. (Accessed on 04/21/2019).

MININET – An Instant Virtual Network on your Laptop (or other PC). 2019. http://mininet.org/. [Online; accessed Apr 6th, 2019].

MOLNáR, S.; MEGYESI, P.; SZABó, G. How to validate traffic generators? In: *2013 IEEE International Conference on Communications Workshops (ICC)*. [S.l.: s.n.], 2013. p. 1340–1344. ISSN 2164-7038.

MOONGEN. 2019. https://github.com/emmericp/MoonGen. [Online; accessed May 14th, 2016].

MULTI-GENERATOR (MGEN). 2019. http://www.nrl.navy.mil/itd/ncs/products/mgen . [Online; accessed May 14th, 2016].

MXTRAF. 2019. http://mxtraf.sourceforge.net/. [Online; accessed January 11th, 2017].

NETFPGA. 2019. https://github.com/NetFPGA/netfpga/wiki/PacketGenerator. [Online; accessed December 12th, 2016].

NETSPEC – A Tool for Network Experimentation and Measurement. 2019. http://www.ittc.ku.edu/netspec/. [Online; accessed May 14th, 2016].

NG, A. *Aprendizagem Automática | Coursera*. 2019. <https://pt.coursera.org/learn/machine-learning>. (Accessed on 04/23/2019).

NPING. 2019. https://nmap.org/nping/ . [Online; accessed May 14th, 2016].

OSNT Traffic Generator. 2019. https://github.com/NetFPGA/OSNT-Public/wiki/OSNT-Traffic-Generator. [Online; accessed December 12th, 2016].

OSTINATO Network Traffic Generator and Analyzer. 2016. http://ostinato.org/. [Online; accessed May 14th, 2016].

OSTROWSKY, L. O.; FONSECA, N. L. S. da; MELO, C. A. V. A traffic model for udp flows. In: *2007 IEEE International Conference on Communications*. [S.l.: s.n.], 2007. p. 217–222. ISSN 1550-3607.

PACKETH. 2015. http://packeth.sourceforge.net/packeth/Home.html. [Online; accessed May 14th, 2016].

PASCHOALON, A. *AndersonPaschoalon/aic-bic-paper*. 2019. <https://github.com/AndersonPaschoalon/aic-bic-paper>. (Accessed on 04/22/2019).

PASCHOALON, A. *Simitar*. 2019. <https://github.com/AndersonPaschoalon/Simitar>. [Online; accessed May 30th, 2017].

Paschoalon, A. d. S.; Rothenberg, C. E. Towards a flexible and extensible framework for realistic traffic generation on emerging networking scenarios. *IX DCA/FEEC/University of Campinas (UNICAMP) Workshop (EADCA)*, p. 1–4, September 2016. Disponível em: <https://pdfs.semanticscholar.org/0c25/504a9a78ceca227b95e775e3cf9735c83fec.pdf>.

Paschoalon, A. d. S.; Rothenberg, C. E. Using bic and aic for ethernet traffic model selection. is it worth? *X DCA/FEEC/University of Campinas (UNICAMP) Workshop (EADCA)*, p. 1–4, October 2017. Disponível em: <https://www.fee.unicamp.br/sites/default/files/departamentos/dca/eadca/eadcax/trabalhos/artigo_22_Using_BIC_AID_Ethernet_Traffic_Anderson_Prof_Christian.pdf>.

Paschoalon, A. d. S.; Rothenberg, C. E. Automated selection of inter-packet time models through information criteria. *IEEE Networking Letters*, p. 1–1, 2019. ISSN 2576-3156.

PATHCHIRP. 2003. http://www.spin.rice.edu/Software/pathChirp/. [Online; accessed Apr 14th, 2019].

PATHLOAD – measurement tool for the available bandwidth of network paths. 2006. https://www.cc.gatech.edu/ dovrolis/bw-est/pathload.html. [Online; accessed Apr 14th, 2019].

PRECISETRAFGEN. 2019. https://github.com/NetFPGA/netfpga/wiki/PreciseTrafGen. [Online; accessed December 12th, 2016].

PYSHARK · PyPI. 2019. <https://pypi.org/project/pyshark/>. (Accessed on 04/17/2019).

QIAN, B.; RASHEED, K. Hurst exponent and financial market predictability. *Proceedings of the Second IASTED International Conference on Financial Engineering and Applications*, 01 2004.

QUANTILE-QUANTILE Plot. 2019. http://mathworld.wolfram.com/Quantile-QuantilePlot.html. [Online; accessed Nov 2nd, 2018].

Q–Q plot. 2019. <https://ipfs.io/ipfs/QmXoypizjW3WknFiJnKLwHCnL72vedxjQkDDP1mXWo6uco/wiki/QQ_plot.html>. (Accessed on 05/05/2019).

Ramya, C. M.; Shanmugaraj, M.; Prabakaran, R. Study on zigbee technology. In: *2011 3rd International Conference on Electronics Computer Technology*. [S.l.: s.n.], 2011. v. 6, p. 297–301.

RIBEIRO, V.; RIEDI, R.; NAVRáTIL, J.; COTTRELL, L. pathchirp: Efficient available bandwidth estimation for network paths. *Proceedings of Passive and Active Measurement Workshop*, 04 2003.

ROLLAND, C.; RIDOUX, J.; BAYNAT, B. Litgen, a lightweight traffic generator: Application to p2p and mail wireless traffic. In: *Proceedings of the 8th International Conference on Passive and Active Network Measurement*. Berlin, Heidelberg: Springer-Verlag, 2007. (PAM'07), p. 52–62. ISBN 978-3-540-71616-7. Disponível em: <http://dl.acm.org/citation.cfm?id=1762888.1762896>.

RONGCAI, Z.; SHUO, Z. Network traffic generation: A combination of stochastic and self-similar. In: *Advanced Computer Control (ICACC), 2010 2nd International Conference on*. [S.l.: s.n.], 2010. v. 2, p. 171–175.

ROSS, S. M. *Introduction to Probability Models, Ninth Edition*. Orlando, FL, USA: Academic Press, Inc., 2006. ISBN 0125980620.

RUDE & CRUDE. 2002. http://rude.sourceforge.net/. [Online; accessed December 12th, 2016].

SCAPY – Packet crafting for Python2 and Python3. 2019. https://scapy.net/. [Online; accessed Apr 6th, 2019].

SEAGULL – Open Source tool for IMS testing. 2006. http://gull.sourceforge.net/doc/WP_Seagull_Open_Source_tool_for_IMS_testing.pdf. [Online; accessed May 14th, 2016].

SEAGULL: an Open Source Multi-protocol traffic generator. 2009. http://gull.sourceforge.net/. [Online; accessed May 14th, 2016].

SHORT User's guide for Jugi's Traffic Generator (JTG). 2019. http://www.netlab.tkk.fi/ jmanner/jtg/Readme.txt. [Online; accessed January 11th, 2017].

SOCKETS. 2019. https://www.gnu.org/software/libc/manual/html_node/Sockets.html. [Online; accessed May 30th, 2017].

Soltanmohammadi, E.; Ghavami, K.; Naraghi-Pour, M. A survey of traffic issues in machine-to-machine communications over lte. *IEEE Internet of Things Journal*, v. 3, n. 6, p. 865–884, Dec 2016. ISSN 2327-4662.

SOMMERS, J.; BARFORD, P. Self-configuring network traffic generation. In: *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement*. New York, NY, USA: ACM, 2004. (IMC '04), p. 68–81. ISBN 1-58113-821-0. Disponível em: <http://doi.acm.org/10.1145/1028788.1028798>.

SOMMERS, J.; KIM, H.; BARFORD, P. Harpoon: A flow-level traffic generator for router and network tests. *SIGMETRICS Perform. Eval. Rev.*, ACM, New York, NY, USA, v. 32, n. 1, p. 392–392, jun. 2004. ISSN 0163-5999. Disponível em: <http://doi.acm.org/10.1145/1012888.1005733>.

SOMMERVILLE, I. *Software Engineering*. Addison-Wesley, 2007. (International computer science series). ISBN 9780321313799. Disponível em: <https://books.google.com.br/books?id=B7idKfL0H64C>.

SOURCESONOFF. 2019. http://www.recherche.enac.fr/ avaret/sourcesonoff. [Online; accessed December 19th, 2016].

SPIEGELHALTER, D. J.; BEST, N. G.; CARLIN, B. P.; LINDE, A. van der. The deviance information criterion: 12 years on. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, v. 76, n. 3, p. 485–493, 2014. Disponível em: <https://rss.onlinelibrary.wiley.com/doi/abs/10.1111/rssb.12062>.

SRIVASTAVA, S.; ANMULWAR, S.; SAPKAL, A. M.; BATRA, T.; GUPTA, A. K.; KUMAR, V. Comparative study of various traffic generator tools. In: *Engineering and Computational Sciences (RAECS), 2014 Recent Advances in*. [S.l.: s.n.], 2014. p. 1–6.

TCPDUMP & Libpcap. 2019. http://www.tcpdump.org/. [Online; accessed May 14th, 2016].

TCPDUMP/LIBPCAP public repository. 2019. http://www.tcpdump.org/. [Online; accessed May 30th, 2017].

TCPIVO: A High Performance Packet Replay Engine. 2019. http://www.thefengs.com/wuchang/work/tcpivo/. [Online; accessed May 14th, 2016].

TCPREPLAY home. 2019. http://tcpreplay.appneta.com/. [Online; accessed May 14th, 2016].

THE OpenDayLight Platform. 2019. https://www.opendaylight.org/. [Online; accessed May 29th, 2017].

THE Swing Traffic Generator. 2019. http://cseweb.ucsd.edu/ kvishwanath/Swing/. [Online; accessed May 14th, 2016].

TRAFFIC Generator. 2011. http://www.postel.org/tg/. [Online; accessed May 14th, 2016].

TSHARK - The Wireshark Network Analyzer 3.0.1. 2019. <https://www.wireshark.org/docs/man-pages/tshark.html>. (Accessed on 04/17/2019).

Tune, P.; Roughan, M.; Cho, K. A comparison of information criteria for traffic model selection. In: *2016 10th International Conference on Signal Processing and Communication Systems (ICSPCS)*. [S.l.: s.n.], 2016. p. 1–10.

UNDERSTANDING Q-Q Plots | University of Virginia Library Research Data Services + Sciences. 2019. <https://data.library.virginia.edu/understanding-q-q-plots/>. (Accessed on 05/05/2019).

VARET, N. L. A. Realistic network traffic profile generation: Theory and practice. *Computer and Information Science*, v. 7, n. 2, 2014. ISSN 1913-8989.

VISHWANATH, K. V.; VAHDAT, A. Evaluating distributed systems: Does background traffic matter? In: *USENIX 2008 Annual Technical Conference*. Berkeley, CA, USA: USENIX Association, 2008. (ATC'08), p. 227–240. Disponível em: <http://dl.acm.org.ez88.periodicos.capes.gov.br/citation.cfm?id=1404014.1404031>.

VISHWANATH, K. V.; VAHDAT, A. Swing: Realistic and responsive network traffic generation. *IEEE/ACM Transactions on Networking*, v. 17, n. 3, p. 712–725, June 2009. ISSN 1063-6692.

W3C. *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. 2019. <https://www.w3.org/TR/REC-xml/#sec-intro>. (Accessed on 04/21/2019).

WEISSTEIN, E. W. Self-similarity. *From MathWorld–A Wolfram Web Resource*, 2019. Disponível em: <http://mathworld.wolfram.com/Self-Similarity.html>.

WELCOME to BRUTE homepage! 2003. http://wwwtlc.iet.unipi.it/software/brute/ . [Online; accessed May 14th, 2016].

WELCOME to the Netperf Homepage. 2019. http://www.netperf.org/netperf/. [Online; accessed May 14th, 2016].

WILK, M. B.; GNANADESIKAN, R. Probability plotting methods for the analysis for the analysis of data. *Biometrika*, v. 55, n. 1, p. 1–17, 03 1968. ISSN 0006-3444. Disponível em: <https://doi.org/10.1093/biomet/55.1.1>.

WILLINGER, W.; TAQQU, M. S.; SHERMAN, R.; WILSON, D. V. Self-similarity through high-variability: statistical analysis of ethernet lan traffic at the source level. *IEEE/ACM Transactions on Networking*, v. 5, n. 1, p. 71–86, Feb 1997. ISSN 1063-6692.

Wu, X.; Xu, K.; Hall, P. A survey of image synthesis and editing with generative adversarial networks. *Tsinghua Science and Technology*, v. 22, n. 6, p. 660–674, December 2017. ISSN 1007-0214.

YANG, Y. Can the strengths of aic and bic be shared? a conflict between model indentification and regression estimation. *Biometrika*, v. 92, n. 4, p. 937, 2005. Disponível em: <+http://dx.doi.org/10.1093/biomet/92.4.937>.

Yu, L.; Qi, D. Hölder exponent and multifractal spectrum analysis in the pathological changes recognition of medical ct image. In: *2011 Chinese Control and Decision Conference (CCDC)*. [S.l.: s.n.], 2011. p. 2040–2045. ISSN 1948-9439.

A Probability and Math Revision

A.1 Random variable

We call random variable X a measurable real-valued function of possible outcomes (Ω) defined on a sample space(E).

$$X : \Omega \rightarrow E \tag{A.1}$$

A.2 Probability Density Function (PDF)

Variable X is a continuous random variable if if there is a function $f(x)$, that satisfies for a set $B = \{b \in \mathbb{R} | b_1 \leq b \leq b_2\}$, defined for all $x = \{x \in \mathbb{R} | -\infty \leq x \leq +\infty\}$, having the property:

$$P(X \in B) = \int_{b_1}^{b_2} f(x)dx \tag{A.2}$$

Where P is the probability function of the random variable x. $f(x)$ is called probability density function of the random variable X [Ross 2006].

A.3 Cumulative Distribution Function (CDF)

The Cumulative Distribution Function of a real-valued random variable X, is a function $F(x)$ defined by [Ross 2006]:

$$F(x) = P(X \leq x) = \int_{-\infty}^{x} f(x)dx \tag{A.3}$$

Where $f(x)$ is the probability density function (PDF) of X.

A.4 Expected value, Mean, Variance and Standard Deviation

Let X be a constinuous real-valued random variable, and $f(x)$ be its probability density function (PDF). Then the expected value of X is defined by:

$$E[X] = \int_{-\infty}^{+\infty} xf(x)dx \tag{A.4}$$

For a random variable normally distributed X_{normal} the result of this definition is equals to its mean μ of the distribution.

$$E[X_{normal}] = \mu \tag{A.5}$$

For an exponential distribution is equals to the inverse of its rate:

$$E[X_{exponential}] = \frac{1}{\lambda} \tag{A.6}$$

The variance of a random variable X, denoted by $Var(X)$, is defined by:

$$var(X) = E[X^2] - (E[x])^2 \tag{A.7}$$

For a random variable X normally distributed, the variance is equal to its standard deviation [Ross 2006]:

$$var(X) = \sigma^2 \tag{A.8}$$

For a finite data $X = \{x_1, x_2, ..., x_n\}$set we can estimate the mean and standard deviation using the follow equations:

$$\mu = \frac{1}{n} \sum_{n}^{i=1} X_i \tag{A.9}$$

$$\sigma = \sqrt{\frac{1}{n}[(x_1 - \mu)^2 + (x_2 - \mu)^2 + ... + (x_n - \mu)^2]} \tag{A.10}$$

A.5 Stochastic Process

A stochastic process of a random variable represented by $\{X(t) | t \in T\}$ is a collection of random variables. Since t is often interpreted as time, $X(t)$ is usually referred as the state of the process at a given time t [Ross 2006].

A.6 Correlation (Pearson correlation coefficient)

Letting (X, Y) be a pai of real-valued random variables, the covariance is defined by:

$$cov(X,Y) = E[(X - E[X])(Y - E[Y])] \tag{A.11}$$

And the Pearson's correlation coefficient is defined by:

$$cor(X,Y) = \frac{cov(X,Y)}{\sigma_X \sigma_Y} \tag{A.12}$$

Where:

$$\sigma_X = \sqrt{E[X^2] - E[X]^2} \tag{A.13}$$

A.7 Autocorrelation of a finite time series

The autocorrelation function measures the correlation between data samples y_t and y_{t+k}, where $k = 0, ..., K$, and the data sample $\{y\}$ is generated by a stochastic process.

According to [Box *et al.* 1994], the autocorrelation for a lag k is:

$$r_k = \frac{c_k}{c_0} \tag{A.14}$$

where

$$c_k = \frac{1}{T-1} \sum_{t=1}^{T-k} (y_t - \bar{y})(y_{t+k} - \bar{y}) \tag{A.15}$$

and c_0 is the sample variance of the time series.

A.8 Self-similarity

A self similar object has the property of looking "roughly" the same at any scale. Self-similar objects are described by the power law:

$$N = s^d \tag{A.16}$$

where

$$d = \frac{\ln N}{\ln s} \tag{A.17}$$

is the dimension of the scaling law, called Hausdorff dimension [Weisstein 2019].

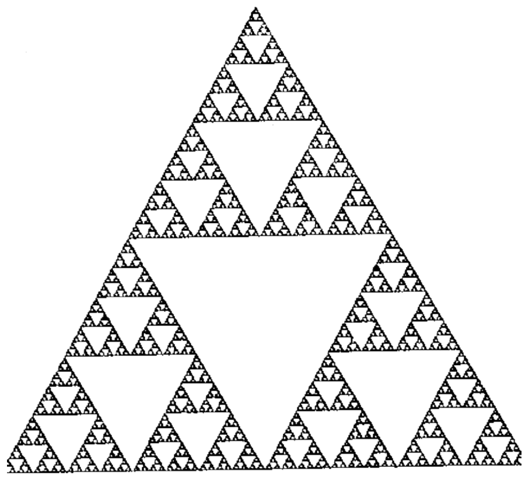


Figure 42 – This is a classical example of a self-similar figure, caled Sierpinski triangle.

A.9 Hurst Exponent

For a time-series $X = \{X_1, X_2, ..., X_n\}$, letting m be the time series mean:

$$\mu = \frac{1}{n} \sum_{n}^{i=1} X_i \tag{A.18}$$

We can calculate the adjusted series Y by:

$$Y = \{Y_t\} = \{X_t - \mu\} \tag{A.19}$$

for $t = 1, 2, ..., n$. We can calculate the cumulative deviate series Z by:

$$Z_t = \sum_{t}^{1} Y_i, \quad t = 1, 2, ..., n \tag{A.20}$$

We can than calculate the time series range by:

$$R(n) = max(Z_1, Z_2, ..., Z_n) - min(Z_1, Z_2, ..., Z_n) \tag{A.21}$$

And its standard deviation by:

$$S(n) = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (X_i - \mu)^2} \tag{A.22}$$

Letting $E[x]$ be the expected value of a real-valued random variable X, and C a constant, the Hurst Exponent H is defined by [Qian e Rasheed 2004]:

$$E\left[\frac{R(n)}{S(n)}\right] = Cn^H, \quad n \to \infty \tag{A.23}$$

A.10 Heavy-tailed distributions

Heavy tailed distributions are probability distributions whose tails are not exponentially bounded. A distribution of of a real-valued random variable X, with cumulative distribution $F(x)$, is said to be heavy tailed, if it satisfies this condition for all $\lambda \in \mathbb{R}$:

$$\lim_{x \to \infty} e^{\lambda x}(1 - F(x)) = \infty \tag{A.24}$$

A.11 *QQplot* analysis

QQplot is used to test if two data-sets comes from a common distribution [Quantile-Quantile Plot 2019] [WILK e GNANADESIKAN 1968] [Understanding Q-Q Plots | University of Virginia Library Research Data Services + Sciences 2019] [Q–Q plot 2019]. In our study case, we used to compare empirical data with theoretical given by model approximations. We show down below the image presented in Chapter 2 for reference. Looking on how the dot plot behaves compared to the linear line, we can see how well the theoretical plot (the estimated data, on the horizontal axis) represents the actual values (sample data, vertical axis):

- **Light-tailed**: the samples still hold a slight heavy-tail effect compared to the estimated by the theoretical values.
- **Heavy-tailed**: the samples have a predominant heavy-tail effect compared to the estimated by the theoretical values.
- **Linear**: the samples match the theoretical values.
- **Bimodal**: samples present a bimodal pattern.
- **Left skew**: small values are under-represented by the model (44).
- **Right skew**: larger values are under-represented by the model (44).

As an example, we created a *QQplot* (Figure 45), where we used as samples randomly generated data, generated by a Cauchy, and theoretical values, normal random data. Comparing with the Figure 43, we can identify a heavy tail behavior on the sample data.

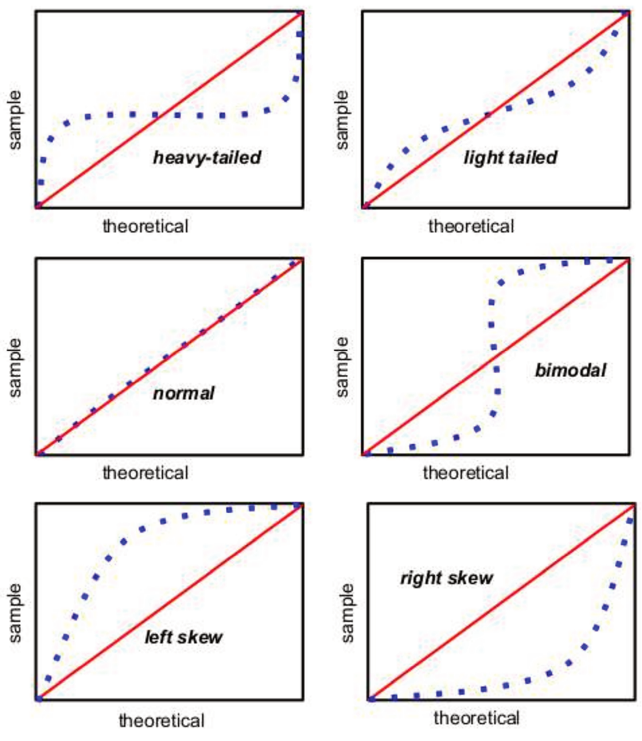


Figure 43 – How information about data samples can be extracted from *QQplots*. Depending on the shape of the dot plot,

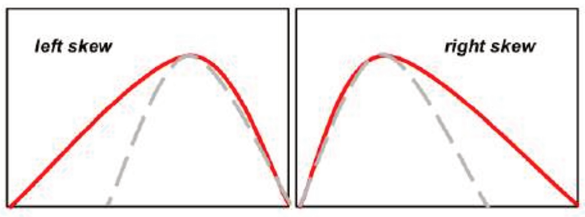


Figure 44 – Shape of a distribution with right and left skew.

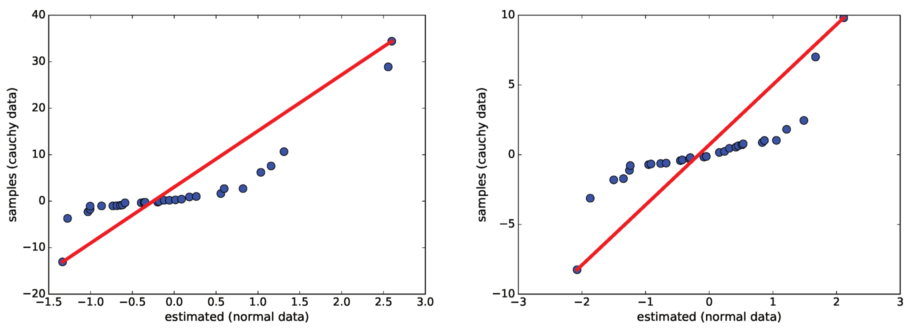


Figure 45 – QQplot of randomly generated data of a Cauchy process as samples and a normal process as theoretical. We can identify a heavy-tail behavior on the samples, compared to the theoretical.

To generate the plots in Figure 45, we used Python and the libraries matplotlib and numpy. The code used is shown down below:

```
import numpy as np
import matplotlib.pyplot as plt

nn = sorted(np.random.standard_normal(30))
cc = sorted(np.random.standard_cauchy(30))
nn_max = max(nn)
nn_min = min(nn)
cc_max = max(cc)
cc_min = min(cc)
yy = np.linspace(cc_min, cc_max, num=10)
xx = np.linspace(nn_min, nn_max, num=10)
fig, ax = plt.subplots()
ax.plot(nn, cc, 'bo', markersize=10.0)
ax.plot(xx, yy, 'r-', linewidth=4.0)
plt.xlabel('estimated (normal data)')
plt.ylabel('samples (cauchy data)')
plt.tick_params(labelsize=14)
plt.tight_layout()
plt.show()
```

A.12 Akaike information criterion (AIC) and Bayesian information criterion (BIC)

Suppose that we have an statistical model M of some dataset $x = \{x_1, ..., x_n\}$, with n independent and identically distributed observations of a random variable X. This model can be expressed by a PDF $f(x|\theta)$, where θ a vector of parameter of the PDF, $\theta \in \mathbb{R}^k$ (k is the number of parameters). The likelihood function of this model M is given by:

$$L(\theta|x) = f(x_1|\theta) \cdot ... \cdot f(x_n|\theta) = \prod_{i=1}^{n} f(x_i|\theta) \tag{A.25}$$

Now, suppose we are trying to estimate the best statistical model, from a set $M_1, ..., M_n$, each one whit an estimated vector of parameters $\hat{\theta}_1, ..., \hat{\theta}_n$. *AIC* and *BIC* are defined by:

$$AIC = 2k - \ln(L(\hat{\theta}|x)) \tag{A.26}$$

$$BIC = k\ln(n) - \ln(L(\hat{\theta}|x)) \tag{A.27}$$

In both cases, the preferred model M_i, is the one with the smaller value of AIC_i or BIC_i.

A.13 Gradient Descendent Algorithm

Given a linear hypothesis for a dataset:

$$h_\theta = \theta^T x \tag{A.28}$$

were $h_\theta, \theta, x \in \mathbb{R}^m$. If $m = 2$ we will just have a simple linear equation $h_\theta(x) = \theta_0 + \theta_1 x$.

The goal of the gradient descendent is to minimize the cost function $J_\nabla(\theta)$, defined by:

$$J_\nabla(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)} - y^{(i)})^2 \tag{A.29}$$

To do this, we initialize a θ_j vector (usually with zeros), and repeat this procedure, until θ_j converges:

$$\theta_{j+1} := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)} - y^{(i)})x_j^i \tag{A.30}$$

where α is the step value, typically a small positive number. All values of θ_j must be updated simultaneously [Ng 2019].

B Computer Networks Review

B.1 Network Stack

Network Stack [Kurose 2017] or Protocol stack is the implementation of computer networks, where a known set of protocols are responsible for delivering the data. The Stack is composed of five layers: Application, Transport, Network, Link and Physical layer.

- ***Application-layer***: This layer is responsible for delivery to the processes the data.This layers deliver **Data**. Some protocols are HTTP, HTTPS, Telnet, DNS, FPT, and SMTP.
- ***Transport-layer***: It is responsible for establishing the communication between hosts (end-to-end communication) and deliver reliability to the data. This layer delivers **Segments**. The main protocols are TCP and UDP.
- ***Newtwork-layer***: It is responsible for the path determination and addressing between the end-points. This layer delivers **Packets**. As examples of protocols we have IP (IPv4 and IPv6), and ICMP.
- ***Link-layer***: It is responsible for the communication and data delivery between hosts and adjacent nodes on LANs and WANs. This layer delivers **Frames**. Some protocols are ARP, MAC (Ethernet), and Wi-Fi (IEEE 802.11) protocols, Bluetooth protocols, and ZigBee (IEEE 802.15) protocols.
- ***Physical-layer***: This layer is the hardware implementation of the Link-layer protocols, and is responsible for the bit transmission.

B.2 Software Defined Networking (SDN)

Software Defined Network [Kreutz *et al.* 2015] is a network achitecture where the forwarding plane (switches and routers, the data plane), and the network control logic (networking policies, the control plane) are separated, introducing the ability of program the network. SDN architecture has four main pillars:

i The control plane and the data plane are decoupled: control functionalities are removed from network devices;

ii The forwarding policies are flow-based, instead of destination-based;

iii The logic resides on an external entity, the Network Operational System(NOS);

iv The network in programmable through applications that runs on the NOS.

The most consolidate protocol that does the communication between the control plane and the data plane is OpenFlow. As examples of Controllers or NOS, we have OpenDayLight and Beacon.

B.3 Network Function Virtualization (NFV)

Network Function Virtualization (NFV) [Huang *et al.* 2018] is a concept and architecture that leveraging IT virtualization technologies, aims to consolidate proprietary and hardware-based middleboxes, such as firewalls, WAN accelerators, routers, and load-balancers into commodity hardware, such as x86 servers, and high-volume switches and storages. NFV architecture has three main layers:

- ***NFVI (NFV Infrastructure)***: This layer comprehends the actual physical network infrastructure, a virtualization layer, and virtualized resources of computing, storage, and networking;
- ***VNF-layer (Virtual Network Functions Layer)***: this is the layer where the virtualized network functions run, and consume resources provided by the NFVI.
- ***MANO (Management and Orchestration)***: On this layer resides the NFV Orchestrator, the VNF manager, and the Virtualized Infrastructure Manager.

B.4 Internet of Things (IoT)

Internet of Things(IoT) [Minerva Abyi Biru 2015] can be defined as "A network of items – each embedded with sensors – which are connected to the Internet." [Minerva Abyi Biru 2015]. The architecture of IoT has three main layers: Applications, Networking and Data-communication, and Sensing.

C Traffic Generators Survey

C.1 Introduction

This appendix contains the cut-content of chapter 2, serving now as complementary material. In the first section we show an extensive survey on traffic generator tools, and on the second, some use-cases of traffic-generators validation.

C.2 Traffic generator tools

In this section, we present a short review of many open-source tools available for synthetic traffic generation and benchmark. The aim in this section is to present the most mentioned tools in the literature and the most recent and advanced ones. On tables are presented a survey of the main features of such tools. Some free, but not open-source traffic generators are listed as well. Before present our survey, we refer to some tools mentioned in the literature, but we could not find source code and manual.

BRUNO [Antichi *et al.* 2008] is traffic generator implemented aiming performance and accuracy on timings. It has many configurable parameters that allow emulation of many web server scenarios. **Divide and conquer** [Molnár *et al.* 2013]: is a replay engine that works in a distributed manner. It can split traces among multiple commodity PCs, and reply packets, to produce realistic traffic.

Some others mentioned tools [D-ITG, Distributed Internet Traffic Generator 2015] we were not able to find any reference of available features are: **UDPgen**, **Network Traffic Generator**, **Packet Shell**, **Real-Time Voice Traffic Generator**, **PIM-SM Protocol Independent Multicast Packet Generator**, **TTCP**, **SPAK, Packet Generator**, **TfGen**, **TrafGen** and **Mtools**. Table 18 presents an updated list of links for download.

C.2.1 Traffic Generators - Feature Survey

Tables 14, 15, 16, and 17 is presented a survey of of the main features of such tools, such as support for Operational systems, protocols, stochastic functions available for traffic generation, and traffic generator class. Some free, but not open-source, traffic generators are listed as well.

Table 14 – Summary of packet-level traffic generators.

Packet-level Traffic Generators				
Traffic Generator	**Operating System**	**Network Protocols**	**Available stochastic distributions**	**Interface**
D-ITG	Linux, Windows, Linux Familiar, Montavista, Snapgear	IPv4-6, ICMP, TCP UDP, DCCP, SCTP	constant, uniform, exponential, pareto, cauchy, normal, poisson, gamma	CLI, Script, API
Ostinato	Linux, Windows, FreeBDS	Ethernet/802.3/LLC, SNAP; VLAN, (with QinQ); ARP, IPv4-6-Tunnelling; TCP, UDP, ICMPv4, ICMPv6, IGMP, MLD; HTTP, SIP, RTSP, NNTP, custom protocol, etc...	constant	GUI, CLI, script, API
PackETH	Linux, MacOS, Windows	Ehernet II, ethernet 802.3, 802.1q, QinQ, ARP, IPv4-6, UDP, TCP, ICMP, ICMPv6, IGMP	constant	CLI, GUI
Seagull	Linux, Windows	IPv4-6, UDP, TCP, SCTP, SSL/TLS and SS7/TCAP. custom protocol	constant, poisson	CLI, GUI
Iperf	Windows, Linux, Android, MacOS X, FreeBSD, OpenBSD, NetBSD, VxWorks, Solaris	IPv4-6, UDP, TCP, SCTP	constant	CLI, API
BRUTE	Linux	IPv4-6, UDP, TCP	constant, poisson, trimodal, exponential	CLI, script
SourcesOnOff	Linux	IPv4, TCP, UDP	weibull, pareto, exponential, normal	CLI
TG	Linux, FreeBSD, Solaris SunOS	IPv4, TCP, UDP	constant, uniform, exponential	CLI
Mgen	Linux(Unix), Windows	IPv4-6, UDP, TCP, SINK	constant, exponential,	CLI, Script
KUTE	Linux 2.6	UDP	constant	kernel module
RUDE & CRUDE	Linux, Solaris SunOS, FreeBSD	IPv4, UDP	constant	CLI
NetSpec	Linux	IPv4,UDP, TCP	uniform, normal, log-normal, exponential, poisson, geometric, pareto, gamma	script
Nping	Windows, Linux, Mac OS X	TCP, UDP, ICMP, IPv4-6, ARP	constant	CLI
MoonGen	Linux	IPv4-6, IPsec, ICMP, UDP, TCP	constant, poisson	scipt API
Dpdk Pktgen	Linux	IPv4, IPv6, ARP, ICMP, TCP, UDP	constant	CLI, script API
LegoTG	Linux	(depend on underlying tool)	(depend on underlying tool)	CLI, script
gen_send/ gen_recv	Solaris, FreeBSD, AIX4.1, Linux	UDP	constant	CLI
mxtraf	Linux	TCP, UDP, IPv4	constant	GUI, script
Jigs Traffic Generator (JTG)	Linux	TCP, UDP, IPv4-6	constant	CLI

C.2.2 Packet-level traffic generators

D-ITG [Botta *et al.* 2012] [D-ITG, Distributed Internet Traffic Generator 2015]: D-ITG (Distributed Internet Traffic Generator) is a platform capable to produce IPv4 and IPv6 traffic defined by IDT and PS probabilistic distributions such as constant, uniform, Pareto, Cauch, Normal, Poisson, Gamma, Weibull, and On/Off; both configurable and pre-defined for many applications, from Telnet, through online games. It provides many flow-level options of customization, like duration, start delay and number of packets, support to many link-layer and

Table 15 – Summary of multi-level and flow-level traffic generators.

Flow and Multi-level Traffic Generators				
Traffic Generator	**Operating System**	**Network Protocols**	**Model**	**Interface**
Swing	Linux	IPv4, TCP, UDP, HTTP, NAPSTER, NNTP and SMTP	Multi-level auto-configurable Ethernet	CLI
Harpoon	FreeBSD, Linux, MacOS X, Solaris	TCP, UDP, IPv4, IPv6	Flow-level auto-configurable Ethernet	CLI
LiTGen	-	-	Multi-level Wifi	-
EAR	Linu	IEEE 802.11, ICMP, UDP, TCP, TFTP, Telnet	"Event Reproduction Ratio" techinique - wireless IEEE 802.11	-

Table 16 – Summary of application-level traffic generators.

Application-level Traffic Generators			
Traffic Generator	Operating System	Model	Interface
GenSyn	Java Virtual Machine	User-behavior emulation	GUI
D-ITG	Linux, Windows, Linux Familiar, Montavista, Snapgear	Telnet, DNS, Quake3, CounterStrike (active and inactive), VoIP (G.711, G.729, G.723)	CLI
Surge	Linux	Client/Server	CLI
Httperf	Linux	HTTP/1.0, HTTP/1.1	CLI
VoIP Traffic Generator	-	VoIP	CLI
ParaSynTG	-	HTTP workload properties	CLI
NetSpec	LInux	HTTP, FTP, Telnet, Mpev video, voice and video teleconference	CLI

Table 17 – Summary of replay-engines traffic generators.

Replay-Engines Traffic Generators		
Traffic Generator	**Operating System**	**Implementation**
Ostinato	Linux, Windows, FreeBDS	Software-only
PackETH	Linux, MacOS, Windows	Software-only
BRUNO	Linux	hardware-dependent
TCPReplay	Linux	Software-only
TCPivo	Linux	Software-only
NetFPGA PacketGenerator	Linux	Hardware
NetFPGA Caliper	Linux	Hardware
NetFPGA OSTN	Linux	Hardware
MoonGen	Linux	Hardware-dependent
DPDK Pktgen	Linux	Hardware-dependent
NFPA	Linux	hardware-dependent

transport-layer protocols, options, sources and destinations addresses/ports. It has support for NAT traversal, so it is possible to make experiments between two different networks separated by the cloud. D-ITG can also be used to measure packet loss, jitter, and throughput. D-ITG may be used through a CLI, scripts, or a API, that can be used to create applications and remotely control other hosts through a daemon.

Ostinato [Ostinato Network Traffic Generator and Analyzer 2016]: Ostinato is a packet crafter, network traffic generator and analyzer with a friendly GUI (“Wireshark in reverse” as the documentation says) and a Python API. This tool permits craft and sends packets of different protocols at different rates. Support Server/Client communication and a vast variety of protocols, from the link layer (such as 802.3 and VLAN) to the application layer (such HTTP and IP). It is also possible to add any unimplemented protocols, through scripts defined by the user.

Seagull [Seagull – Open Source tool for IMS testing 2006] [Seagull: an Open Source Multi-protocol traffic generator 2009]: an Open Source Multi-protocol traffic generator 2009]: Seagull is a traffic generator and test open-source tool, released by HP. It has support of many protocols, from link layer to application layer, and its support is easily extended, via XML dictionaries. As the documentation argues, the protocol extension flexibility is one of the main features. It supports high speeds, and is reliable, being tested through hundreds of hours. It can also generate traffic using three statistical models: uniform (constant), best-effort and Poisson.

BRUTE [Welcome to BRUTE homepage! 2003]: Is a traffic generator that operates on the top of Linux 2.4-6 and 2.6.x, not currently being supported on newer versions. It also supports some stochastic functions (constant, poison, trimodal) for departure time burst, and can simulate VoIP traffic.

PackETH [PACKETH 2015]: PackETH is GUI and CLI stateless packet generator tool for ethernet. It supports many adjustments of parameters, and many protocols as well, and can set MAC addresses.

Iperf [iPerf - The network bandwidth measurement tool 2019]: Ipef is a network traffic generator tools, designed for the measure of the maximum achievable bandwidth on IP networks, for both TCP and UDP traffic, but can evaluate delay, windows size, and packet loss. It has a GUI interface, called Jperf [JPerf 2019]. There is also a JavaAPI, for automating tests s [jperf 2015]. Support IPv4 and IPv6.

NetPerf [Welcome to the Netperf Homepage 2019]: Netperf is a benchmark tool that can be used to measure the performance of many types of networks, providing tests for both unidirectional throughput and end-to-end latency. It has support for TCP, UDP, and SCTP, both over IPv4 and IPv6.

sourcesOnOff [Varet 2014] [sourcesonoff 2019]: sourcesOnOff is a new traffic generator released on 2014, that aims to generate realistic synthetic traffic using probabilistic models to control on and off time of traffic flows. As shown on the paper, it is able to guarantee self-similarity, an has support to many probabilistic distributions for the on/off times: Weibull, Pareto, Exponential and Gaussian. Supports TCP and UDP over IPv4.

TG [Traffic Generator 2011]: TG is a traffic generator that can generate and receive one-way packet streams transmitted from the UNIX user level process between source and traffic sink nodes. A simple specification language controls it, that enables the craft of different lengths and interarrival times distributions, such as Constant, uniform, exponential and ON/OFF (markov2).

[1]**MGEN** [Multi-Generator (MGEN) 2019]: MGEN (Multi-Generator) is a traffic generator developed by the Naval Research Laboratory (NRL) PROTocol Engineering Advanced Networking (PROTEAN) Research Group. It can be used to emulate the traffic patterns of unicast and/or multicast UDP and TCP IP applications. It supports many different types of stochastic functions, nominated periodic, Poisson, burst jitter and clone which can control inter-departure times and packet size.

KUTE [KUTE – Kernel-based Traffic Engine 2007]: KUTE is a kernel level packet generator, designed to have a maximum performance traffic generator and receiver mainly for use with Gigabit Ethernet. It works in the kernel level, sending packets as fast as possible, direct to the hardware driver, bypassing the stack. However, KUTE works only on Linux 2.6, and has only be tested on Ethernet Hardware. Also, it only supports constant UDP traffic.

RUDE & CRUDE [RUDE & CRUDE 2002]: RUDE (Real-time UDP Data Emitter) and CRUDE (Collector for RUDE), are small and flexible programs which run on user-level. It has a GUI called GRUDE. It works only with UDP protocol.

[2]**NetSpec** [NetSpec – A Tool for Network Experimentation and Measurement 2019]: NetSpec is a tool designed to do network tests, as opposed to doing point to point testing. NetSpec provides a framework that allows a user to control multiple processes across multiple hosts from a central point of control, using daemons that implement traffic sources and sinks, along with measurement tools. Also, it can model many different traffic patterns and applications, such as maximum host rate, Constant Bit Rate (CBR), WWW, World Wide Web, FTP, File Transfer Protocol, telnet, MPEG video, voice, and video teleconference.

Nping [Nping 2019]: active hosts, as a traffic generator for network stack stress testing, ARP poisoning, Denial of Service attacks, route tracing, etc. Nping CLI permits the users control over protocols headers.

TCPreplay [Tcpreplay home 2019]: TCPreplay is a user-level replay engine, that can use *pcap* files as input, and then forward, packets in a network interface. It can modify some header parameters as well.

TCPivo [Feng *et al.* 2003] [TCPivo: A High Performance Packet Replay Engine 2019]: TCPivo is a high-speed traffic replay engine that is able to read traffic traces, and replay packets in a network interface, working at the kernel level. It is not currently supported kernel versions greater than 2.6.

NetFPGA PacketGenerator [NetFPGA 2019]: NetFPGA Packet Generator is a hardware-based traffic generator and capture tool, build over the NetFPGA 1G, and open FPGA platform with 4 ethernet interfaces of 1 Gigabit of bandwidth each. It is a replay engine tool

[1] not open-source
[2] not open-source

which uses as input *pcap* files. It is able to accurately control the delay between the frames, with the default delay being the same in the *pcap* file. It is also able to capture packets and report statistics of the traffic.

NetFPGA Caliper [PreciseTrafGen 2019]: is a hardware-based traffic generator, build on NetFPGA 1G, built over NetThreads platform, an FPGA microprocessor which support threads programming. Different form NetFPGA Packet Generator, Caliper can produce live packets. It is written in C.

NetFPGA OSNT [OSNT Traffic Generator 2019]: OSNT (Open Source Network Tester) is hardware based network traffic generator built over the NetFPGA 10G. As NetFPGA 1G, NetFPGA 10G is an FPGA platform with 4 ethernet interfaces, but with 10 Gigabits of bandwidth. OSNT is a replay engine and is loaded with *pcap* traces.

Dpdk Pktgen [Getting Started with Pktgen 2015]: Pktgen is a traffic generator measurer built over DPDK. DPDK is a development kit, a set of libraries and drivers for fast packet processing. DPDK was designed to run on any processor but has some limitation in terms of supported NICs, that can be found on its website.

MoonGen [Emmerich *et al.* 2015] [MoonGen 2019]: MoonGen is a scriptable high-speed packet generator built over DPDK and LuaJIT. It can send packets at 10 Gbit/s, even with 64 bytes packets on a single CPU core. MoonGen can achieve this rate even if each packet is modified by a Lua script. Also, it provides accurate timestamping and rate control. It is able to generate traffic using several protocols (IPv4, IPv6, IPsec, ARP, ICMP, UDP, and TCP), and can generate different inter-departure times, like a Poisson process and burst traffic.

gen_send/gen_recv [gen_send, gen_recv: A Simple UDP Traffic Generater Application 2019]: gen_send and gen_recv are simple UDP traffic generator applications. It uses UDP sockets. gen_send can control features like desired data rate, packet size and inter-packet time.

mxtraf [mxtraf 2019]: mxtraf enables a small number of hosts to saturate a network, with a tunable mixture of TCP and UDP traffic.

Jigs Traffic Generator (JTG) [Short User's guide for Jugi's Traffic Generator (JTG) 2019]: is a simple, accurate traffic generator. JTG process only sends one stream of traffic, and stream characteristics are defined only by command line arguments. It also supports IPv6.

C.2.3 Application-level/Special-scenarios traffic generators

[3]**ParaSynTG** [Khayari *et al.* 2008]: application-level traffic generator configurable by input parameters, which considers most of the observes www traffic workload properties.

[3] not open-source

[4]**GenSyn** [GenSyn - generator of synthetic Internet traffic 2019]: network traffic generator implemented in Java that mimics TCP and UDP connections, based on user behavior.

Surge [Barford e Crovella 1998]: Surge is an application level workload generator which emulates a set of real users accessing a web server. It matches many empirical measurements of real traffic, like server file distribution, request size distribution, relative file popularity, idle periods of users and other characteristics.

Httperf [httperf(1) - Linux man page 2019]: Is an application lever traffic generator to measure web server performance. It uses the protocol HTTP (HTTP/1.0 and HTTP/1.1), and offer many types of workloads while keeping track of statistics related to the generated traffic. Its most basic operation is to generate a set of HTTP GET requests and measure the number of replies and response rate.

VoIP Traffic Generator: it is a traffic generator written in Perl that creates multiple streams of traffic, aiming to simulate VoIP traffic.

C.2.4 Flow-level and multi-level traffic generators

Harpoon [Sommers *et al.* 2004]: Harpoon is a flow-based traffic generator, that can automatically extract form Netflow traces parameters, in order to generate flows that exhibit the same statistical characteristics measured before, including temporal and spatial characteristics.

Swing [Vishwanath e Vahdat 2009] [The Swing Traffic Generator 2019]: Swing is a closed-loop multi-layer and network responsive generator. It can read capture traces and captures the packet interactions of many applications, being able to models distributions for the user, application, and network behavior, stochastic and responsively. Swing can model user behavior, REEs, connection, packets, and network.

[5]**LiTGen** (Lightweight Traffic Generator) [Rolland *et al.* 2007] is an open-loop, multilevel traffic generator. It can model wireless network traffic in a peer user and application basis. This tool model the traffic in three different levels: packet level, object level (smaller parts of an application session), and session level.

[6]**EAR** [Ku *et al.* 2012]: traffic generator that uses a technique called“EventReproduction Ratio” to mimic wireless IEEE 802.11 protocol behavior.

[4] not open-source
[5] not open-source
[6] not open-source

C.2.5 Others traffic generation tools

NFPA [Csikor *et al.* 2015]: NFPA is a benchmark tool based on DPDK Pkgen, specialized in executing and automatize performance measurements over network functions. It works being directly connected to a specific device under tests. It uses built-in, and user-defined traffic traces and Lua scripts control and collect information of DPDK Pktgen. It has a command line and Web interface, and automatically plot the results.

LegoTG [Bartlett e Mirkovic 2015]: LegoTG is a modular framework for composing custom traffic generation. It aims to simplify the combination on the use of different traffic generators and modulators on different testbeds, automatizing the process of installation, execution, resource allocation, and synchronization via a centralized orchestrator, which uses a software repository. It already has support to many tools, and to add support to new tools is necessary to add and edit two files, called TG block, and ExFile.

C.2.6 Traffic Generators – Repository Survey

C.3 Validation of Ethernet traffic generators: some use cases

In this section, we list some use cases of validation of Ethernet traffic generators. Our validation methods used in Chapter 4 and 5 were based on them. We are going to present seven different study cases on validation of related traffic generators. They are Swing, Harpoon [Sommers e Barford 2004], D-ITG [Botta *et al.* 2012], sourcesOnOff [Varet 2014], MoonGen [Emmerich *et al.* 2015], LegoTG [Bartlett e Mirkovic 2015] and NFPA [Csikor *et al.* 2015].

C.3.1 Swing

Swing [Vishwanath e Vahdat 2009] is at present, one of the primary references of realistic traffic generation. The authors extracted bidirectional metrics from a network link of synthetic traces. Their goals were to get realism, responsiveness, and randomness. They define realism as a trace that reflects the following characteristics of the original:

Packet inter-arrival rate and burstiness across many time scales;

- Packet size distributions;
- Flow characteristics as arrival rate and length distributions;
- Destination IPs and port distributions.

The traffic generator uses a structural model the account interactions between many layers of the network stack. Each layer has many control variables, which is randomly generated

Table 18 – Links for the traffic generators repositories

Traffic Generator	Repository
D-ITG	http://traffic.comics.unina.it/software/ITG/
Ostinato	http://ostinato.org/
Seagull	http://gull.sourceforge.net/
BRUTE	http://wwwtlc.iet.unipi.it/software/brute/
PackETH	http://packeth.sourceforge.net/packeth/Home.html
Iperf	https://iperf.fr/
NetPerf	http://www.netperf.org/netperf/
sourcesOnOff	http://www.recherche.enac.fr/ avaret/sourcesonoff
TG	http://www.postel.org/tg/
MGEN*	http://www.nrl.navy.mil/itd/ncs/products/mgen
KUTE	http://caia.swin.edu.au/genius/tools/kute/
RUDE & CRUDE	http://rude.sourceforge.net/
Pktgen	http://www.linuxfoundation.org/collaborate/workgroups/networking/pktgen
NetSpec	http://www.ittc.ku.edu/netspec/
Nping	https://nmap.org/nping/
TCPreplay	http://tcpreplay.appneta.com/
TCPivo	http://www.thefengs.com/wuchang/work/tcpivo/
NetFPGA PacketGenerator	https://github.com/NetFPGA/netfpga/wiki/PacketGenerator
NetFPGA Caliper	https://github.com/NetFPGA/netfpga/wiki/PreciseTrafGen
NetFPGA OSNT	https://github.com/NetFPGA/OSNT-Public/wiki/OSNT-Traffic-Generator
DPDK Pktgen	http://pktgen.readthedocs.io/en/latest/getting_started.html
MoonGen	https://github.com/emmericp/MoonGen
gen_send/gen_recv	http://www.citi.umich.edu/projects/qbone/generator.html
mxtraf	http://mxtraf.sourceforge.net/
JTG	https://sourceforge.net/projects/iperf/files/ https://github.com/AgilData/jperf
GenSyn	http://www.item.ntnu.no/people/personalpages/fac/poulh/gensyn
SURGE	http://cs-www.bu.edu/faculty/crovella/surge_1.00a.tar.gz
Httperf	https://linux.die.net/man/1/httperf
VoIP Traffic Generator	https://sourceforge.net/projects/voiptg/
Harpoon	http://cs.colgate.edu/ jsommers/harpoon/
Swing	http://cseweb.ucsd.edu/ kvishwanath/Swing/
NFPA	
LegoTG	

by a stochastic process. They begin the parameterization, classifying [Tcpdump & Libpcap 2019] *pcap* files with the data; they can estimate parameters.

They validate the results using public available traffic traces, from [MAWI Working Group Traffic Archive 2019] and CAIDA [CAIDA Center for Applied Internet Data Analysis 2019]. On the paper, the authors focuses on the validation metrics below:

- Comparison of estimated parameters of the original and swing generated traces;

- Comparison of aggregate and per-application bandwidth and packets per seconds;
- QoS metrics such as two-way delay and loss rates;
- Scaling analysis, via Energy multiresolution energy analysis.

To the vast majority of the results, both original and swing traces results were close to each other. Thus, Swing was able to match aggregate and burstiness metrics, per byte and per packet, across many timescales.

C.3.2 Harpoon

Harpoon [Sommers e Barford 2004] [Sommers *et al.* 2004] is a traffic generator able to generate representative traffic at IP flow level. It can generate TCP and IP with the same byte, packet, temporal and spatial characteristics measured at routers. Also, Harpoon is a self-configurable tool, since it automatically extracts parameters from network traces. It estimates some parameters from original traffic trace: file sizes, inter-connection times, source and destination IP addresses, and the number of active sessions.

As proof of concept [Sommers e Barford 2004], the authors compared statistics from the original, and harpoon's generated traces. The two main types of comparisons: diurnal throughput, and stochastic variable CDF and frequency distributions. Diurnal throughput refers to the average bandwidth variation within a day period. In a usual network, during the day the bandwidth consumption is larger than the night. Also, they compared:

- CDF of bytes transferred per 10 minutes
- CDF of packets transferred per 10 minutes
- CDF of inter-connection time
- CDF of file size
- CDF of flow arrivals per 1 hour
- Destination IP address frequency

In the end, they showed the differences in throughput evaluation of a Cisco 6509 switch/router using Harpoon and a constant rate traffic generator. Harpoon was proven able to give close CDFs, frequency and diurnal throughput plots compared to the original traces. Also, the results demonstrated that Harpoon provides a more variable load on routers, compared to constant rate traffic. It indicates the importance of using realistic traffic traces on the evaluation of equipment and technologies.

C.3.3 D-ITG

D-ITG [Botta *et al.* 2012] is a network traffic generator, with many configurable features. The tool provides a platform that meets many emerging requirements for a realistic traffic generation. For example, multi-platform, support of many protocols, distributed operation, sending/receiving flow scalability, generation models, and analytical model based generation high bit/packet rate. You can see different analytical and models and protocols supported by D-ITG at Table 14.

To the evaluation of realism on analytical model parameterization of D-ITG, the authors used a synthetic replication of a eight players's LAN party of Age of Mythology[7]. They have captured traffic flows during the party, then, they modeled its packet size and inter-packet time distributions. They show that the synthetic traffic and the analytical model have similar curves of packet size and inter-packet time; thus it can approximate the empirical data. Also, the bit rate's mean and the standard deviation of the empirical and synthetic data are similar.

C.3.4 sourcesOnOff

Varet et al. [Varet 2014] create an application implemented in C, called SourcesOnOff. It models the activity interval of packet trains using probabilistic distributions. To choose the best stochastic model, the authors had captured traffic traces using TCPdump. Then the developed tool that could configure out what distribution (Weibull, Pareto, Exponential, Gaussian, etc.) is better to the original traffic traces. They used the Bayesian Information Criterion (BIC) for distance assessment and tested the smaller BIC for each distribution, insuring a good correlation between the original and generated traces and self-similarity.

The validation methods used on sourcesOnOff are:

- A visual comparison between On time and Off time of the original trace and the stochastic fitting;
- *QQplots*, which aim to evaluate the correlation between inter-trains duration of real and generated traffic;
- Measurement of the measured throughput's autocorrelation A of the real and synthetic traffic;
- Hurst exponent computation of the real and the synthetic trace;

The results pointed to an excellent stochastic fitting, better for On-time values. On the other hand, the correlation value of the *QQplot* was more significant on the Off time values

[7] https://www.ageofempires.com/games/aom/

(99.8% versus 97.9%). In the real and synthetic traces, the throughput's autocorrelation remained between an upper limit of 5%. Finally, the ratio between the evaluated Hurst exponent always remained smaller than 12%.

C.3.5 MoonGen

MoonGen [Emmerich *et al.* 2015] is a high-speed scriptable paper capable of saturating 10 GnE link with 64 bytes packets, using a single CPU core. The authors built it over DPDK and LuaJit, enabling the user to have high flexibility on the crafting of each packet, through Lua scripts. It has multi-core support and runs on commodity server hardware. MoonGen also can test latency with sub-microsecond precision and accuracy, using hardware timestamping of modern NICs cards. The Lua scripting API enable the implementation and high customization along with high-speed. This includes the controlling of packet sizes and inter-departure times.

The authors evaluated this traffic generator focused on throughput metrics, rather than others. Also, they have small packet sizes (64 bytes to 256), since the per-packet costs dominate. In their work, they were able to surpass 15 Gbit/s with an XL71040GbENIC. Also, they achieve throughput values close to the line rate with packets of 128 bytes, and 2CPU cores.

C.3.6 LegoTG

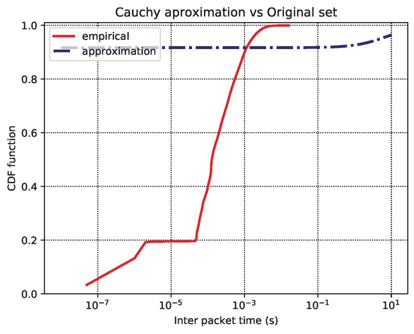
Bartlett et al. [Bartlett e Mirkovic 2015] implemented a modular framework for composing custom traffic generation, called LegoTG. As argued by the authors (and by our present work), automation of many aspects of traffic generation is a desirable feature. The process of how to generate proper background traffic may become research by itself. Traffic generators available today offer a single model and a restricted set of features into a single code base, makes customization difficult.

The primary purpose of their experiment was to show how LegoTG could generate background traffic, only. Also, they showed how realistic background traffic could influence research conclusions. The test chosen is one of the use cases proposed for Swing [Vishwanath e Vahdat 2008], and evaluated the error on bandwidth estimation of different measurement tools. It showed that LegoTG could provide a secure and custom traffic generation.

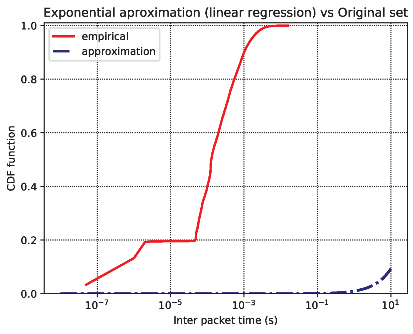
D Chapter 4 Aditional Plots

In this appendix, we include some plots generated by the study presented in Chapter 4, which have been cut off and not included in the final text version. They are:

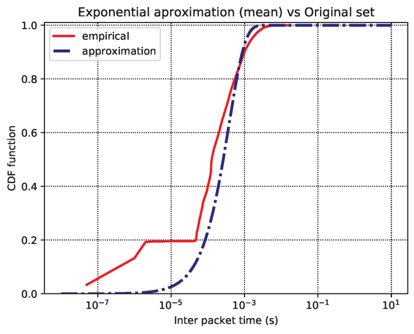
- CDF's distributions;
- QQPlots;
- Cost history (J_∇) and data linearization from linear regression;
- Other plots for AIC, BIC and J_M.



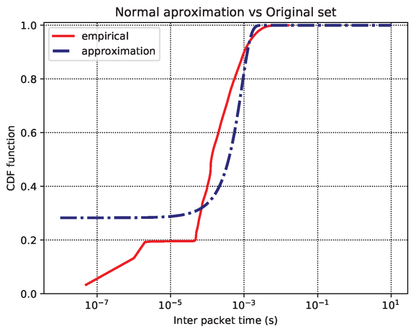
(a) Cauchy



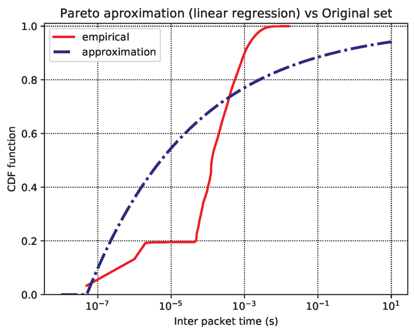
(b) Exponential(LR)



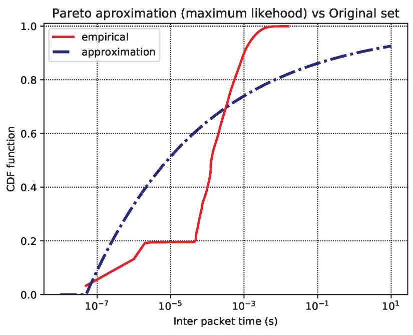
(c) Exponential(Me)



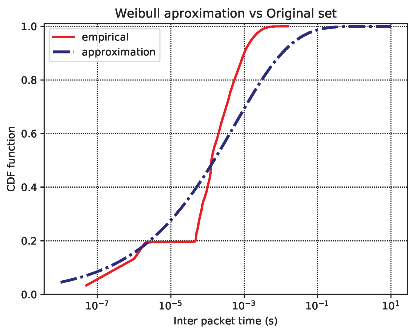
(d) Normal



(e) Pareto(LR)

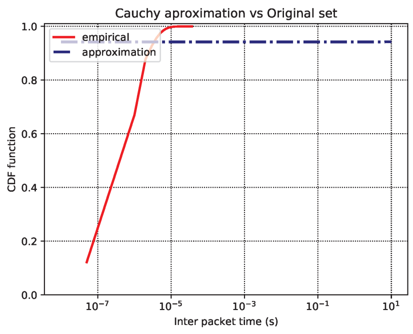


(f) Pareto(MLH)

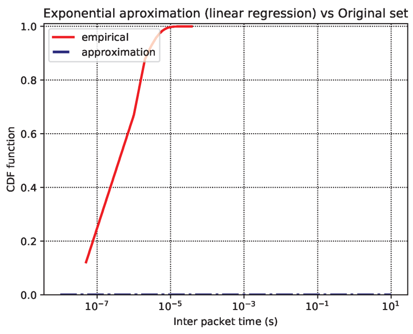


(g) Weibull

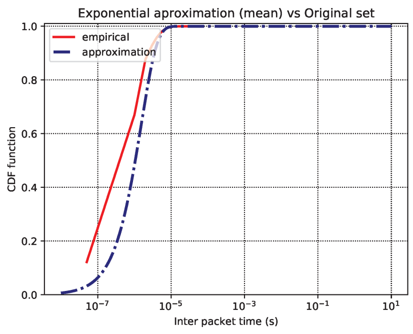
Figure 46 – CDF functions for the approximations of *lan-gateway-pcap* inter packet times, of many stochastic functions.



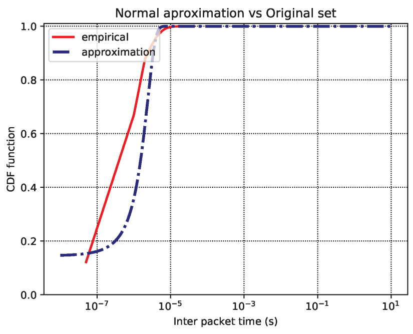
(a) Chauchy



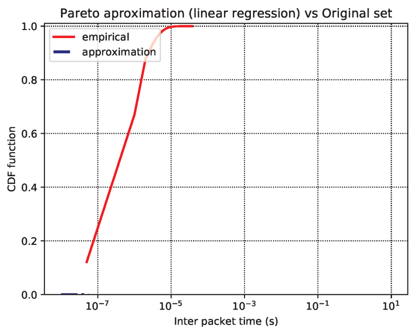
(b) Exponential(LR)



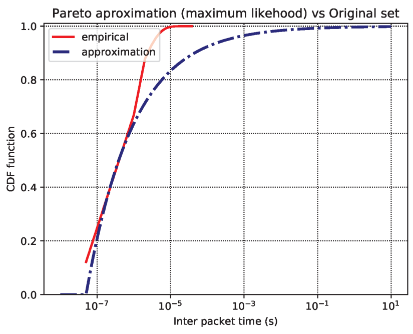
(c) Exponential(Me)



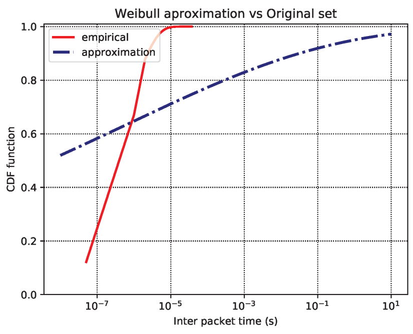
(d) Normal



(e) Pareto(LR)

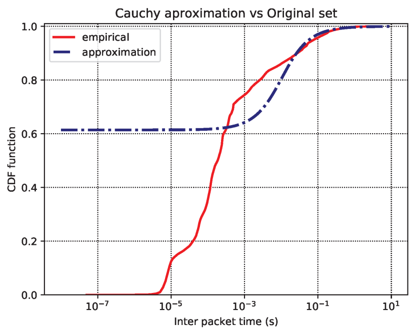


(f) Pareto(MLH)

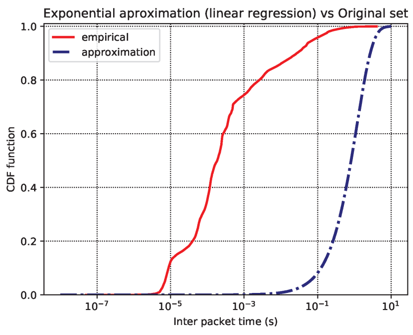


(g) Weibull

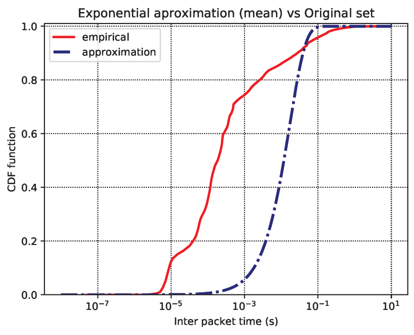
Figure 47 – CDF functions for the approximations of *wan-pcap* inter packet times, of many stochastic functions.



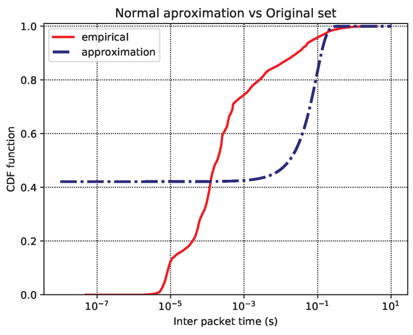
(a) Chauchy



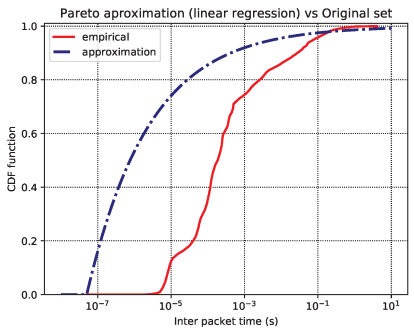
(b) Exponential(LR)



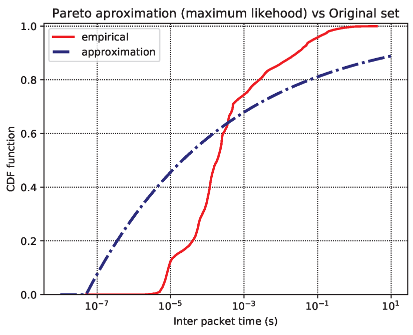
(c) Exponential(Me)



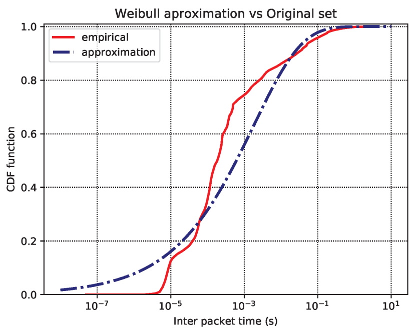
(d) Normal



(e) Pareto(LR)

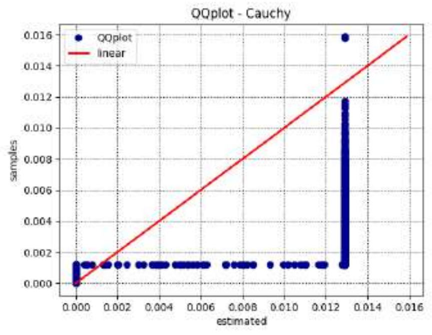


(f) Pareto(MLH)

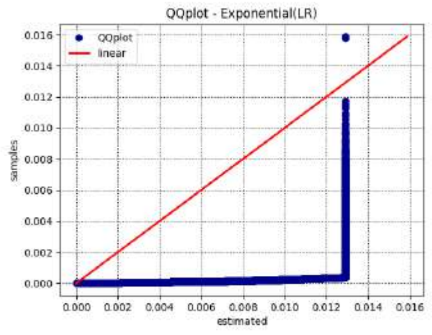


(g) Weibull

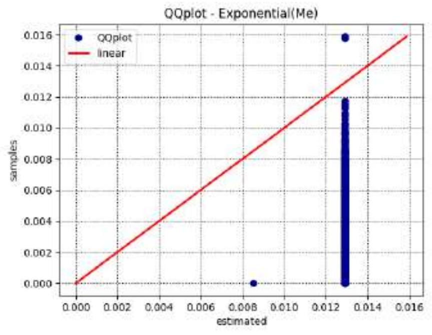
Figure 48 – CDF functions for the approximations of *lan-diurnal-firewall-pcap* inter packet times, of many stochastic functions.



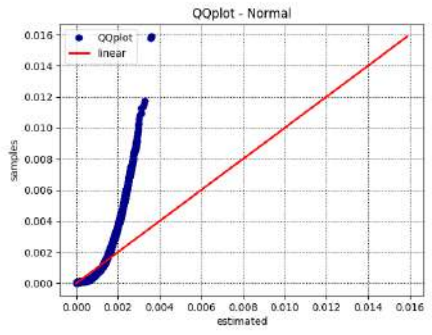
(a) Chauchy



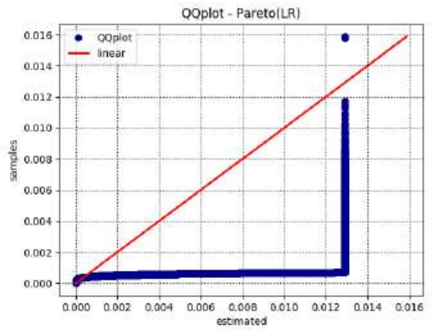
(b) Exponential(LR)



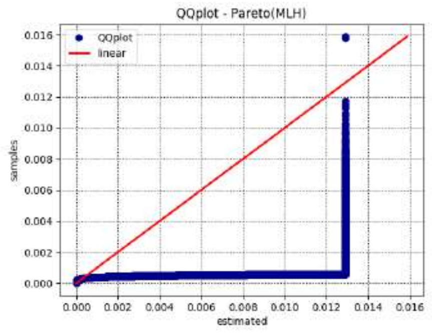
(c) Exponential(Me)



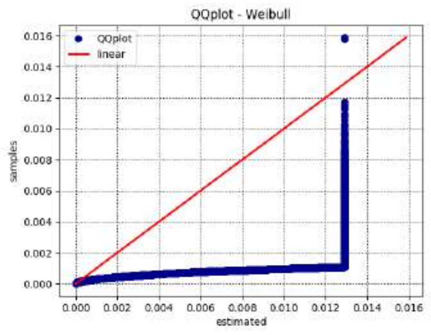
(d) Normal



(e) Pareto(LR)

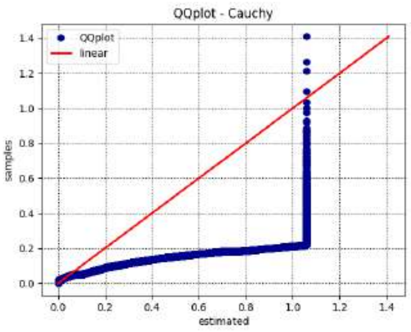


(f) Pareto(MLH)

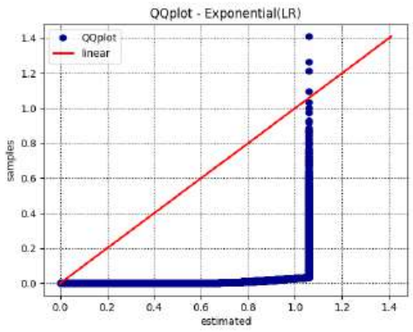


(g) Weibull

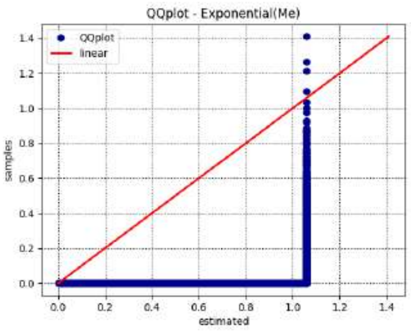
Figure 49 – CDF functions for the approximations of *lan-gateway-pcap* inter packet times, of many stochastic functions.



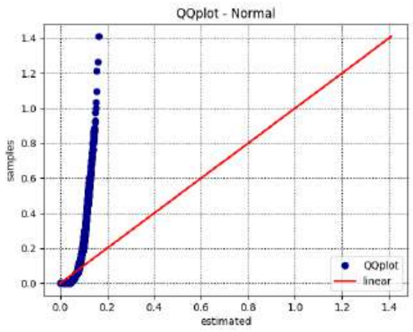
(a) Chauchy



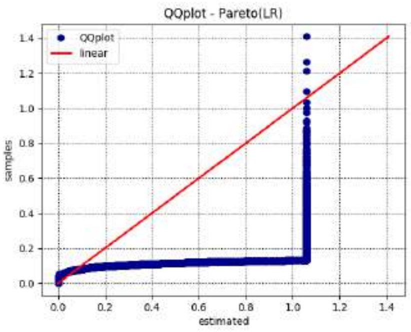
(b) Exponential(LR)



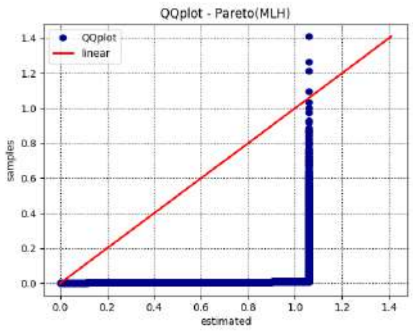
(c) Exponential(Me)



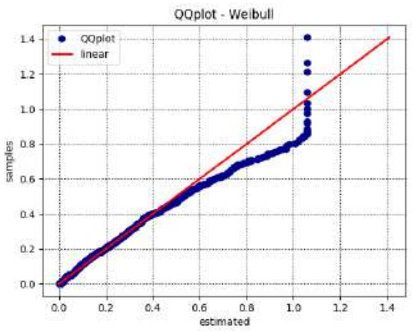
(d) Normal



(e) Pareto(LR)

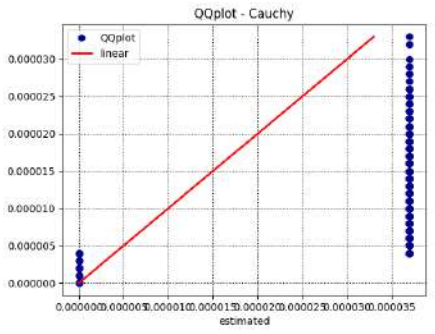


(f) Pareto(MLH)

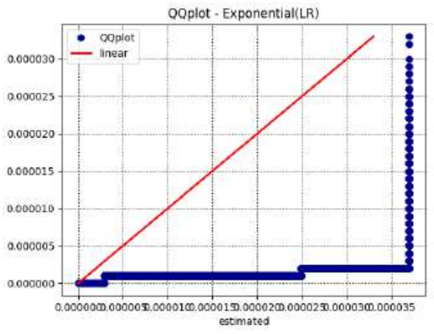


(g) Weibull

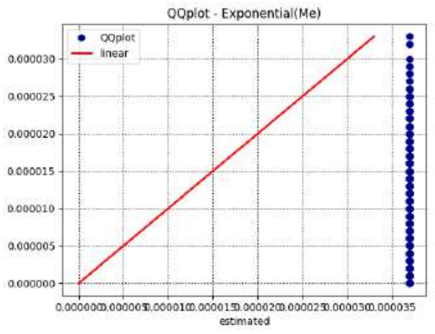
Figure 50 – CDF functions for the approximations of *lan-diurnal-firewall-pcap* inter packet times, of many stochastic functions.



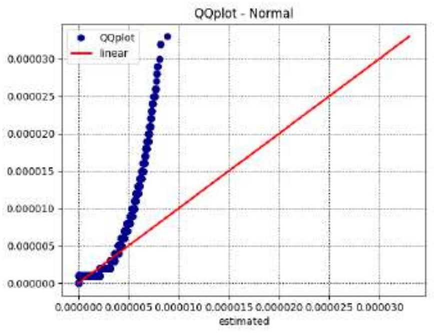
(a) Chauchy



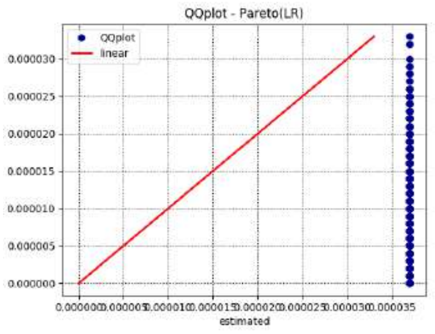
(b) Exponential(LR)



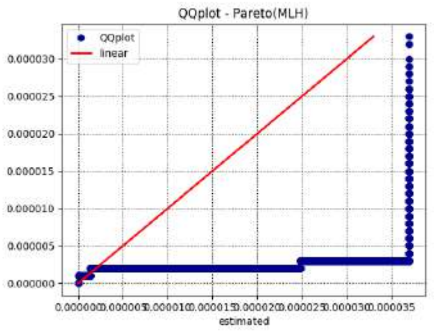
(c) Exponential(Me)



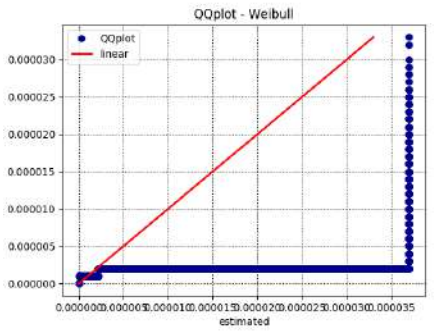
(d) Normal



(e) Pareto(LR)

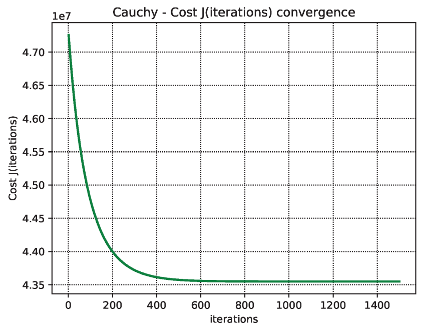


(f) Pareto(MLH)

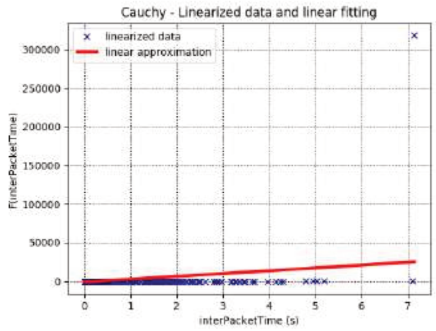


(g) Weibull

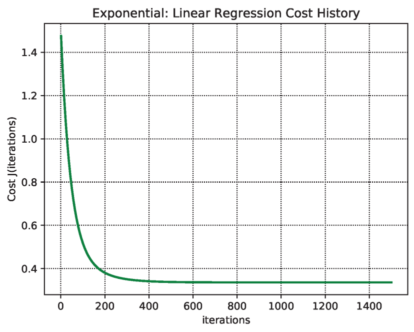
Figure 51 – CDF functions for the approximations of *wan-pcap* inter packet times, of many stochastic functions.



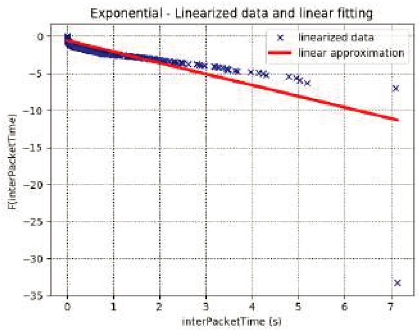
(a) Cauchy



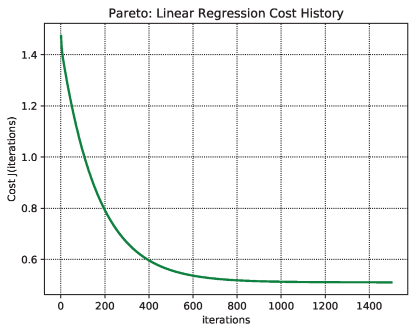
(b) Cauchy



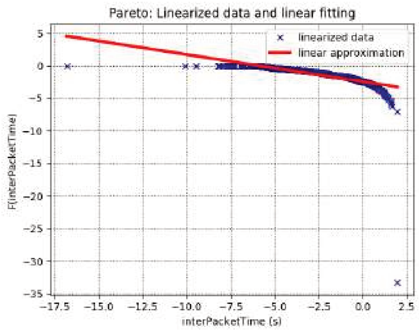
(c) Exponential



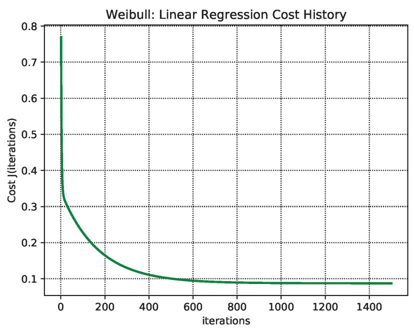
(d) Exponential



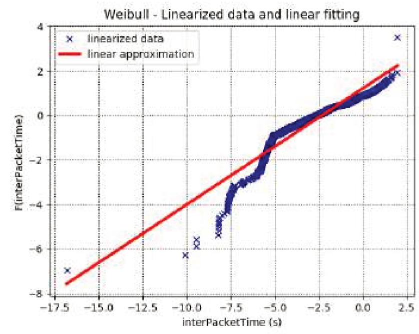
(e) Pareto



(f) Pareto

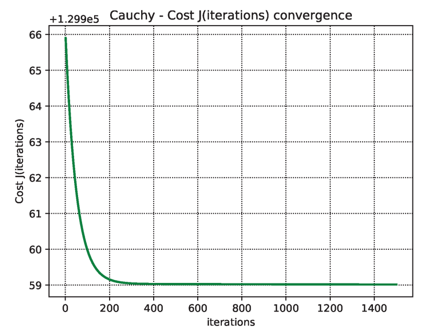


(g) Weibull

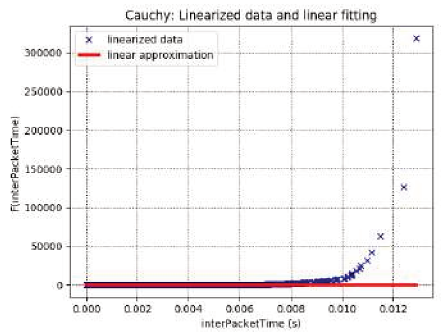


(h) Weibull

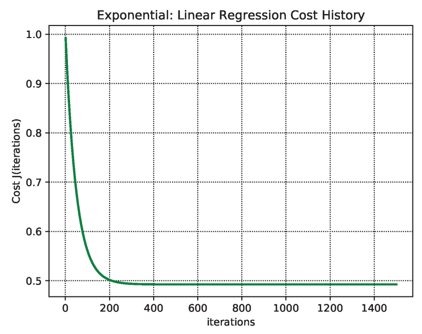
Figure 52 – Data linearization, and linear regression cost history, from gradient descendent for *skype-pcap*.



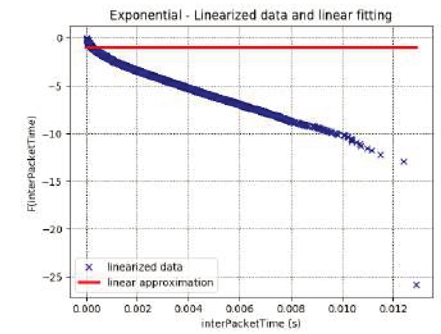
(a) Cauchy



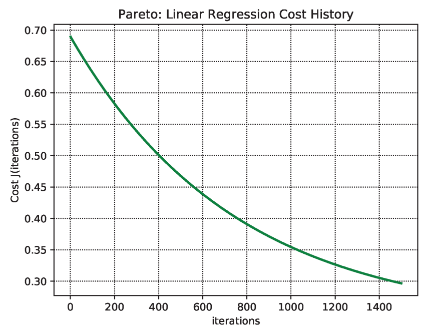
(b) Cauchy



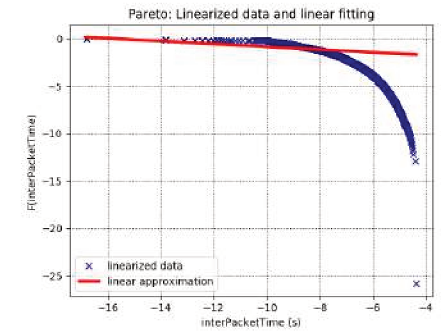
(c) Exponential



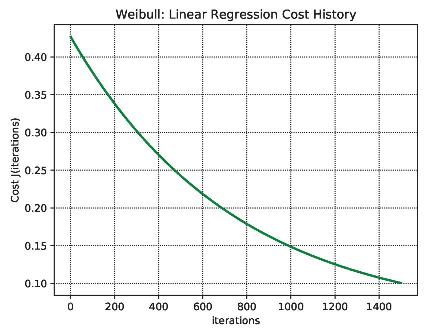
(d) Exponential



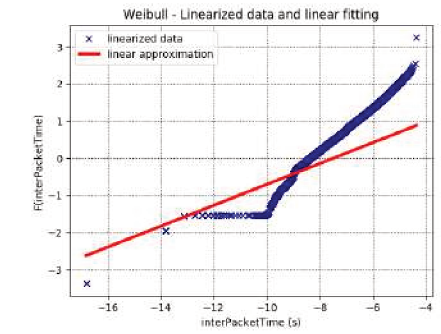
(e) Pareto



(f) Pareto

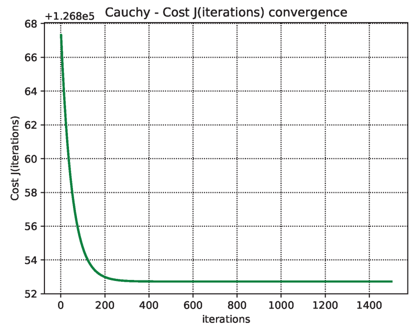


(g) Weibull

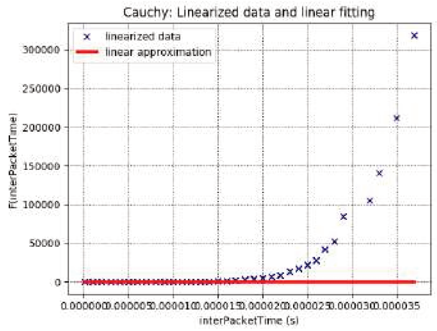


(h) Weibull

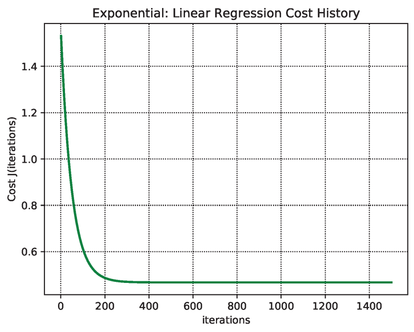
Figure 53 – Data linearization, and linear regression cost history, from gradient descendent for *lan-gateway-pcap*.



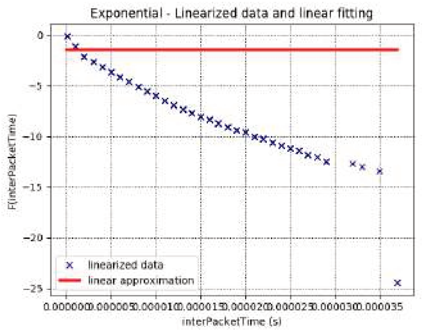
(a) Cauchy



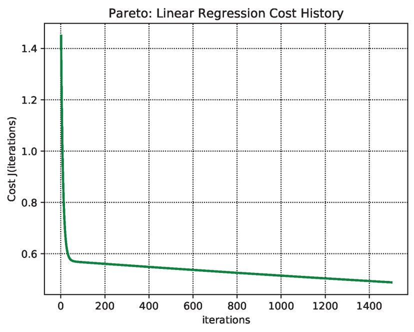
(b) Cauchy



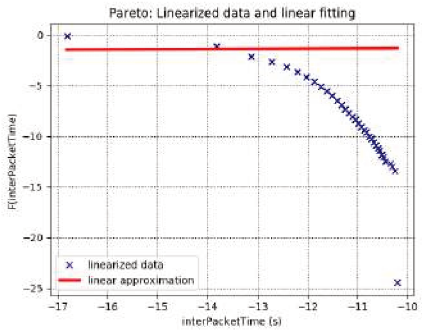
(c) Exponential



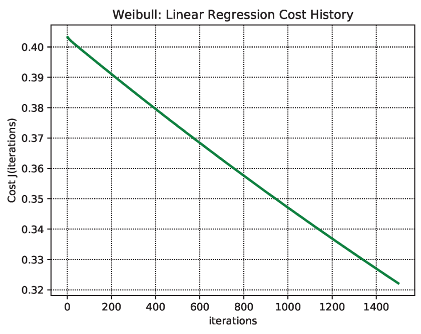
(d) Exponential



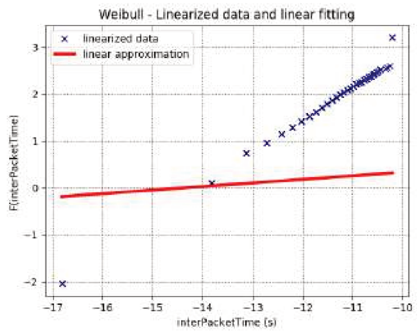
(e) Pareto



(f) Pareto

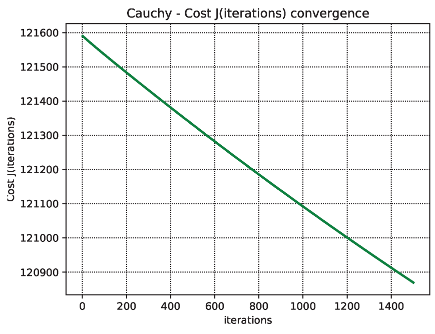


(g) Weibull

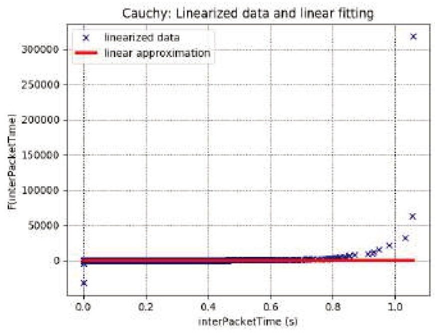


(h) Weibull

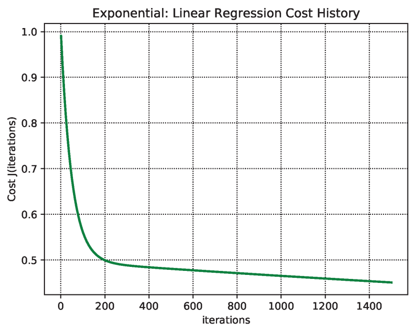
Figure 54 – Data linearization, and linear regression cost history, from gradient descendent for *wan-pcap*.



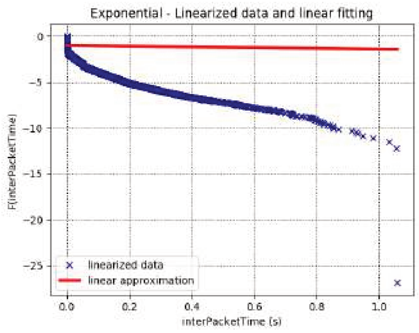
(a) Cauchy



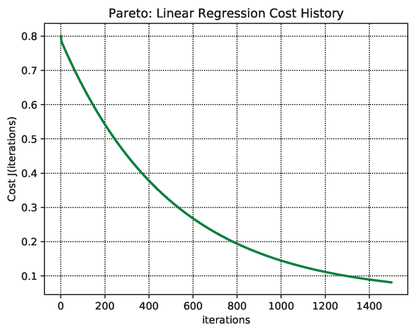
(b) Cauchy



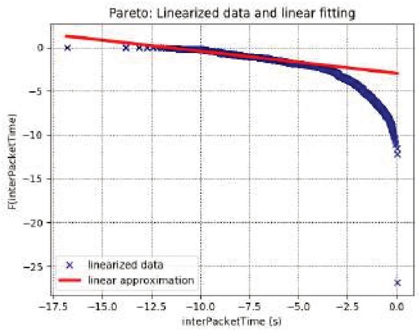
(c) Exponential



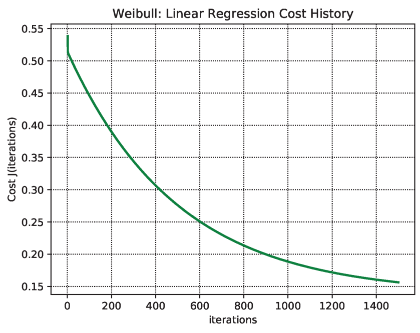
(d) Exponential



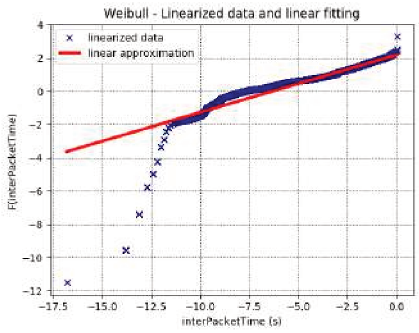
(e) Pareto



(f) Pareto



(g) Weibull



(h) Weibull

Figure 55 – Data linearization, and linear regression cost history, from gradient descendent for *lan-firewall-pcap*.

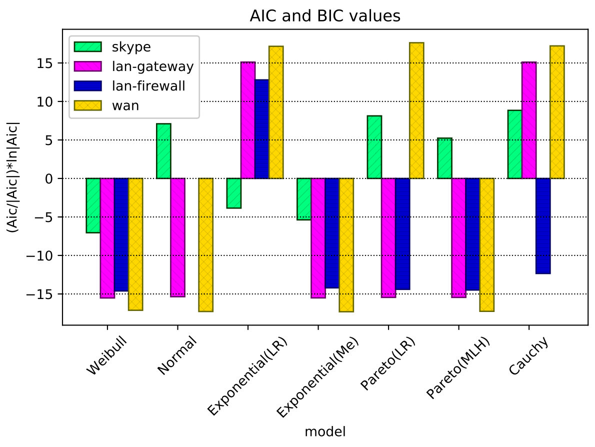


Figure 56 – *AIC* and *BIC* summary for all the traces, presented in log scale.

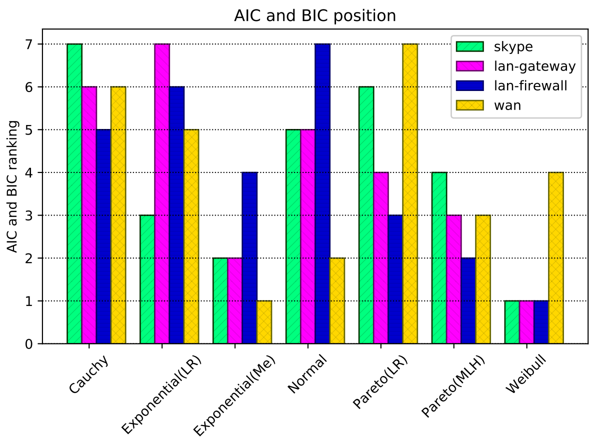


Figure 57 – *AIC* and *BIC* summary for all the traces, presenting the order.

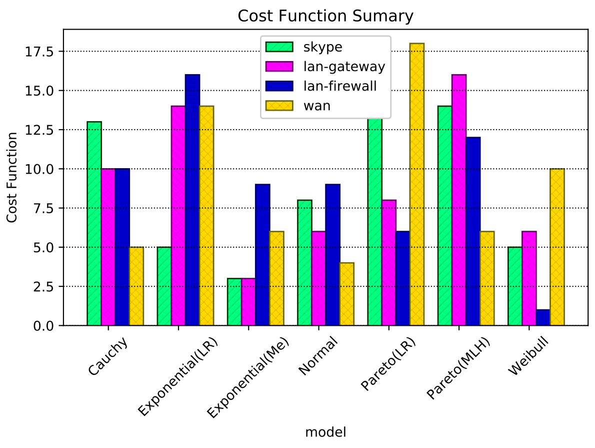


Figure 58 – Cost function J_M summary.

E UML Project Diagrams

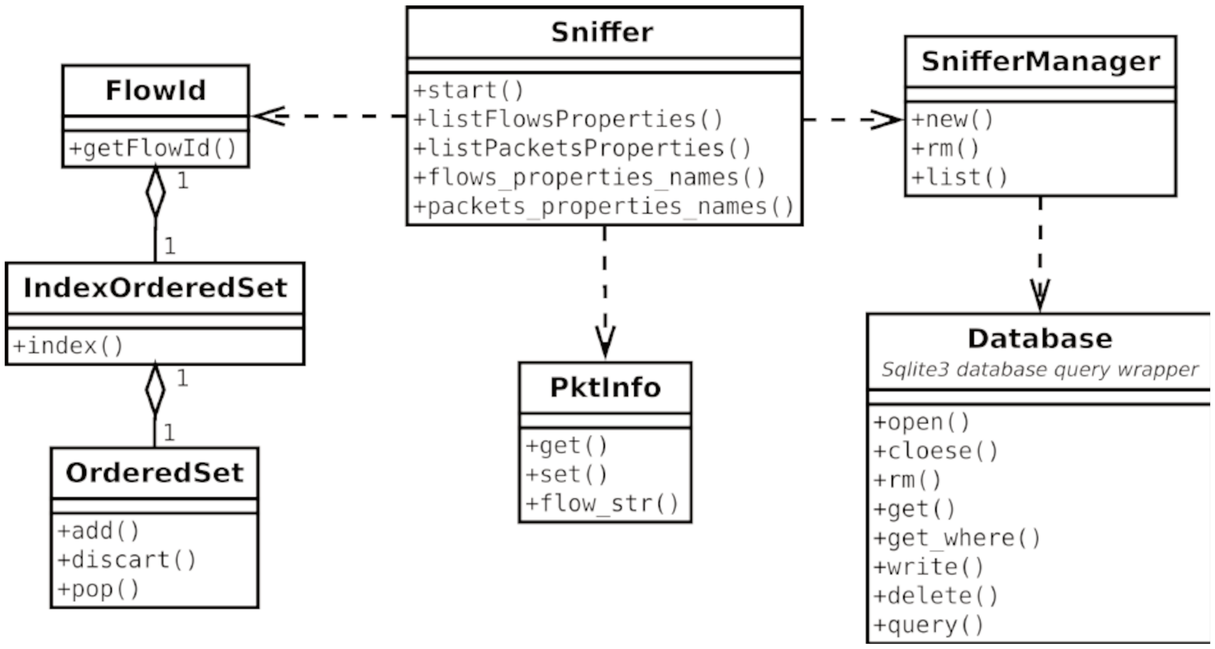


Figure 59 – Sniffer UML Class Diagram

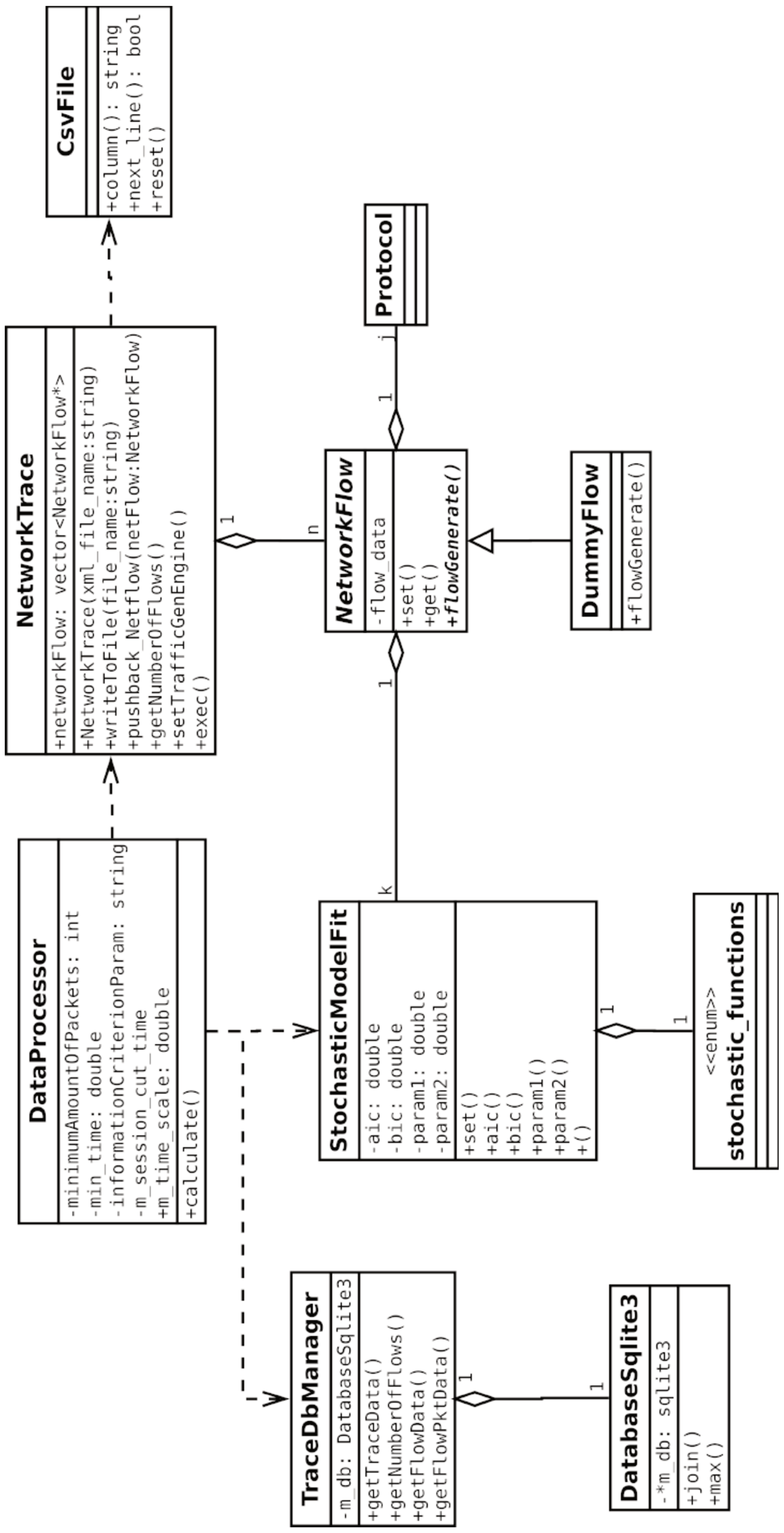


Figure 60 – Trace Analyzer UML Class Diagram

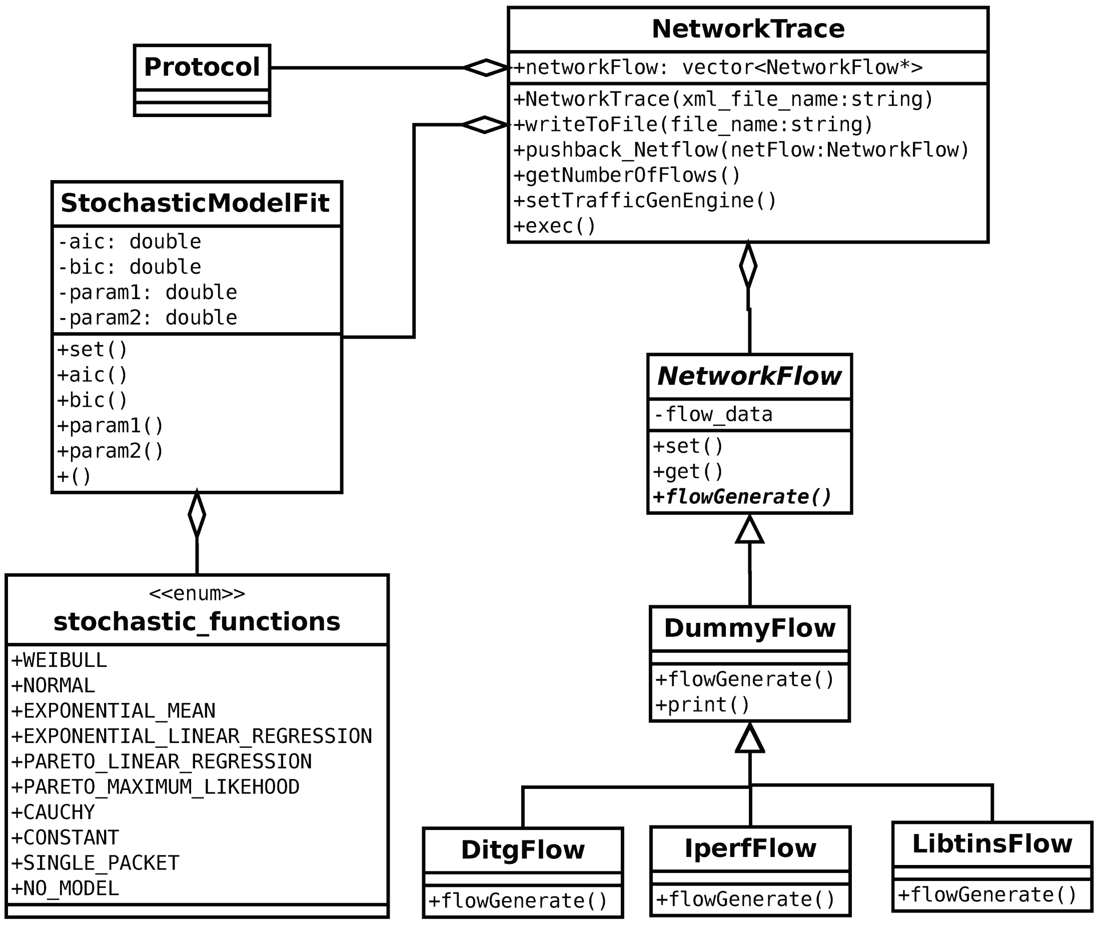


Figure 61 – Flow Generator UML Class Diagram

F Academic contributions

As main academic works, along with this dissertation, we have a set of contributions:

- Two non-indexed articles, presented on the ***EADCA Workshop***:
  - "*Towards a Flexible and Extensible Framework for Realistic Traffic Generation on Emerging Networking Scenarios*" [Paschoalon e Rothenberg 2016]
  - "*Using BIC and AIC for Ethernet traffic model selection. Is it worth?*" [Paschoalon e Rothenberg 2017];
- One Journal Article published 2019 in the ***IEEE Networking Letters***:
  - "*Automated Selection of Inter-Packet Time Models through Information Criteria*" [Paschoalon e Rothenberg 2019]
- One congress submission:
  - "*SIMITAR: Realistic and Autoconfigurable Traffic Generation*";
- One open-source tool:
  - *SIMITAR: SniffIng ModellIng and TrAffic geneRation* [Paschoalon 2019]

IX Encontro de Alunos e Docentes do DCA/FEEC/UNICAMP (EADCA) Campinas, 29 e 30 de setembro de 2016
IX DCA/FEEC/University of Campinas (UNICAMP) Workshop (EADCA) Campinas, Brazil, September 29-30, 2016

Towards a Flexible and Extensible Framework for Realistic Traffic Generation on Emerging Networking Scenarios

Anderson dos Santos Paschoalon , Christian Esteve Rothenberg

Departamento de Engenharia de Computação e Automação Industrial (DCA)
Faculdade de Engenharia Elétrica e de Computação (FEEC)
Universidade Estadual de Campinas (Unicamp)
Caixa Postal 6101, 13083-970 – Campinas, SP, Brasil

{pchoalon ,chesteve}@dca.fee.unicamp.br

Abstract – New emerging technologies have a larger unpredictability, compared to legacy equipment. They require a larger set of meaningful tests on many different scenarios. But, in the open source world is hard to find a single tool able to provide realism, speed, easy usage and flexibility at the same time. Most of the tools are monolithic and devoted to specific purposes. This work presents a flexible and extensible framework which aims to decouple synthetic traffic modelling from its traffic generator engine. Through a new abstraction layer, it would become possible to use modern and throughput optimized tools to create realistic traffic, in an automated way. This enables a platform agnostic configuration and reproduction of complex scenarios via analytical models. Also we use *pcap* files and live-capture to create "Compact Trace Descriptors".

Keywords – realistic, framework, traffic generation, modelling, burstiness, fractal, flow level, packet level, Hurst exponent, wavelet, pcap, emulation, stochastic, inter-departure, packet size, Swing, D-ITG, Harpoon, Swing, SourcesOnOff, LegoTG, DPDK

1. Introduction

Emerging technologies such as SDN and NFV are great promises. If succeeding at large-scale, they should change the development and operation of computer networks. But, enabling technologies such as virtualization still pose challenges on performance, reliability, and security [6]. Thus, guarantee the Service Layer Agreements on emerging scenarios is now a harder question. Applications may have a huge performance degradation processing small packets [12]. As conclude by many investigations, realistic and burstiness traffic impacts on bandwidth measurement accuracy [2]. Also, realistic workload generators are essential security research [4]. Thus, there is a demand for tests able to address realism at high throughput rates.

The open-source community offers a huge variety of workload generators and benchmarking tools [4] [9]. Each tool uses different methods on traffic generation, focusing on a certain aspects. Some traffic generator tools provide support emulation of single application workloads. But this is not enough to describe an actual Service Provider(ISP) load or even a LAN scenario. Other tools work as packet replay engines, such as TCPreplay and TCPivo. Although in that way is possible to produce a realistic workload at high rates, it comes with some issues. First, the storage space required becomes huge for long-term and high-speed traffic capture traces. Also, obtaining good traffic traces sometimes is hard, due privacy issues and fewer good sources. Many tools aim the support of a larger set of protocols and high-performance such Seagull and Ostinato. Many are also able to control inter-departure time and packet size using stochastic models, like D-ITG [4] and MoonGen. They can provide a good control of the traffic, and high rates. But, in this case, selecting a good configuration is by itself a research project, since how to use each parameter to simulate a specific scenario is a hard question [7]. It is a manual process and demands implementation of scripts or programs leveraging human (and scarce) expertise on network traffic patterns and experimental evaluation. Some tools like Swing and Harpoon, try to use the best of both worlds. Both use capture traces to set intern parameters, enabling an easier configuration. Also, Swing uses complex multi-levels which are able to provide a high degree of realism [13]. But they have their issues as well. Harpoon does not configure parameters at packet level [11] and is not supported by newer Linux kernels. Swing [13] aims to generate realistic background traffic, not offering high throughput [13] [2]. As is possible to see, this a result of the fact that its traffic generation engine is coupled to its modeling framework. You can't opt to use a newer/faster packet generator. The only way of replacing the traffic engine is changing and recompiling the original code. And this is a hard task.

This project aims to create a framework able solve many of presented issues. It must be able to "learning" patterns and characteristics of real network traffic traces. Then, using packet generators and accelerators it should reproduce a network traffic with similar characteristics. Based on observation of live captures or PCAP files, the software must choose the bests parametrized stochastic functions (from a list) to fit the data. These parametrized stochastic functions along with collected header features (such as protocols and addresses) will be record in a machine-readable file (such as XML or JSON) we baptise as a Compact Trace Descriptor (CTD). This data must serve as input for an API of a traffic generator. So it will be possible to control packet parameters, and flow's behaviours, through different APIs. Also, and speed-up may be achieved, though the use of DPDk's KNI interfaces[1]. So, the main goal is to offer an easier configuration, realism, at a higher speed than the available platforms today. Also, it will add programmability and abstraction to the traffic generation, since the user may edit or create a custom traffic descriptor in a platform agnostic way. The the intermediate layer of the figure 1 summarize, goal of the project in an illustrative way.

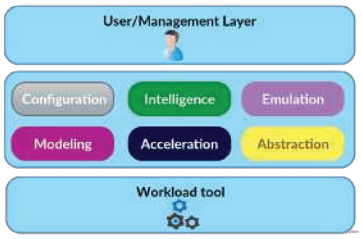


Figure 1. Proposal representation in a layer diagram. It automates features such as configuration, modelling, and parametrization, intelligence, emulation, and abstraction through an additional layer.

2. Literature review and related work

A common taxonomy used on traffic generator tools based on the layer of operation. It divides them into four categories [3]:

Application-level/Special-scenarios traffic generators: they emulate applications behaviours, through stochastic and responsive models. Eg.: Surge, GenSym.

Flow-level traffic generators: they emulate features of the flow level, such as file transference and bursts. But do not model applications or packet behaviour.Eg.: Harpoon.

Packet-level traffic generators: they model inter-departure time and packet size through stochastic distributions. They focus on performance testing. Eg: D-ITG [4], Ostinato, Seagull, TG.

Multi-level traffic generators: These traffic generators models each mentioned layer, describing user and network behaviour. They generate an accurate background traffic, but usually, have bottlenecks on bandwidth. Eg.: Swing [13].

Varet et al. [1] creates an application in C, called SourcesOnOff. It models the activity interval of packet trains using probabilistic distributions. To choose the best stochastic models, the authors captured many traffic traces using TCPdump. Then, they figure out what distribution (Weibull, Pareto, Exponential, Gaussian, etc.) fits better the original traffic traces, using the Bayesian Information Criterion. For this task, they choose the function with the smaller BIC. The smaller this value is, the better the function fits the data.

Bartlett et al. [2] implements a modular framework for composing custom traffic generation. Its goal is making easy the combine of different traffic generators and modulators in different test-beds. It automatizes the process of installation, execution, resource allocation and synchronization using a centralized orchestrator and a software repository. It already has support to many tools, and to add support to new tools is necessary to add and edit two files, called TGblock, and ExFile.

3. System Architecture

We developed and architecture that solves the listed issues. It has five components: a Sniffer, an SQLite database, a Trace Analyzer, a Flow Generator, and a Network Traffic Generator as a subsystem.

The Sniffer is responsible collecting data from the network traffic. It extracts data from the packets and stores them in the database. This information can be protocols, packet size, inter-arrival time, flows, and so on. Also, after finishing a capture, this component is the responsible for providing data visualization. It may work over a PCAP file, or over an Ethernet interface. The prototype version of this component uses tsahrk to capture packets and Shell/Octave scripts. To improve performance and avoid bottlenecks, next implementation

[1] `http://dpdk.org/doc/guides/sample_app_ug/kernel_nic_interface.html`

will use libtins librarie for packet processing. The criteria to classify the traffic into flows is the same of SDN switches: internet protocol, source/destination addresses, transport protocol, and source/destination transport ports. The framework uses an SQLite database.

The Trace Analyzer is the core of the project. It is the tool responsible for characterizing the trace. Using the stored information, breaks the trace into flows, and parametrize each of them. The parameters are header fields and stochastic functions/coefficients for each flow. The component models the behaviour of the trace on flow level and packet level. At the packet level, is possible to model the packet-size and the inter-departure time, during packet bursts (ON times). At the flow level, is possible to control bursts periods, session length, and the number of bytes delivered. We will use likelihood criterions to choose the best probabilistic function and parameters. Options are the smaller error, Akaike information criterion, and Bayesian information criterion [1]. It will sort the parametrized functions in a priority list. After the parametrization, the Trace Analyzer records these features in a machine-readable file (XML, JSON) called "Compact trace descriptor".

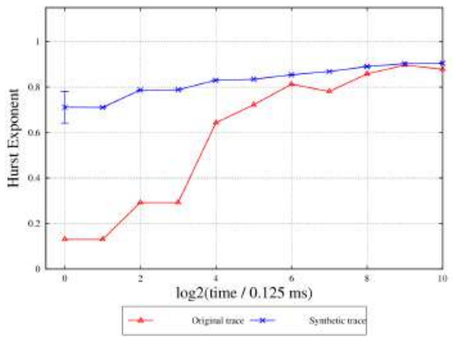


Figure 2. Hurst exponent value of original and synthetic traces

The Flow Generator pick these abstract parameters and feed an Ethernet workload generator tool. It crafts each flow in an independent way, in a different thread. The presented prototype just uses the D-ITG API as workload tool. But it can use any packet-level traffic generator with API or CLI. This component handles the flow level models and parametrizes the packet-level tool underneath. Since each packet-level tool supports a different set of stochastic functions, the Flow Generator should pick the first compatible model from the priority list. But prototype presented here still uses simple models on packet and flow crafting, supporting just constant distributions. But the next release should support at packet-level heavy-tailed [1] and Poison functions for the inter-departure times, and bimodal distributions [5] [10] for the packet size. At the flow level, two different alternatives can be used. Model file transference and session, such as in Harpoon [11]; or use an envelope process, as suggested by Melo et al [8].

4. Partial results

To as proof of concept, we propose a set of tests. We choose them, based on tests used to ensure realism, on related many works [1] [13] [2]. They aim to ensure realism and similarity. Realism tests measure if a synthetic traffic has expected features of an Ethernet capture. Similarity tests measures if the generated traffic represents specific characteristics of the original one. Here, due the limited space, we will present just two results. The first, which test realism, is the Hurst exponent evaluation. It is able to test the self-similarity of the generated traffic. To be self-similar, a process must have a Hurst exponent between 0.5 and 1 [7]. Also, usual values of Ethernet traffic lay between 0.8 and 0.9 [7].

Thus a realistic Ethernet traffic must have a Hurst exponent close to the last interval. The second test is Wavelet Multiresolution Energy Analysis. It is able to capture characteristics of the traffic at different time-scales. For example, it enables visualization of a periodic tendency(decrease) or a self-similar tendency(increase) at a certain time scale. Also, at each point, it represents the mean energy of that signal at that time scale. So, similar Ethernet traffics must have slopes at close time-scales. Also, they must have close energy scales. More close are the curves, more similar are the traces.

The evaluated prototype support just constant functions. It selects the inter-departure time equal to the mean. The packet size is set as the most frequent value. The flow's start time and duration are the same from the observed traffic. We capture the original traffic trace on the laboratory LAN. The results are at figures 2 and 3. On both analysis, the generation of the synthetic trace was repeated 30 times. The keys which serves as input to D-ITG were randomly selected. At is possible to see that the on both cases the Hurst exponent converge to the same value, close to 0.9. But, on the wavelet multiresolution analysis, both curves still different behaviours.

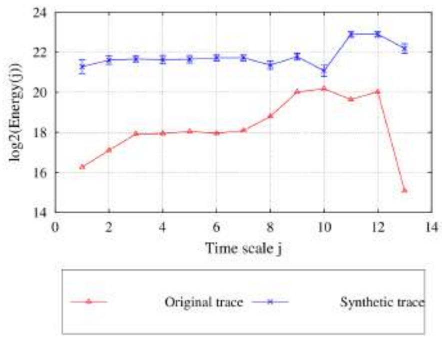


Figure 3. Wavelet Multiresolution Energy Analysis of the original and synthetic traces

5. Conclusion and future work

The framework prototype was already able to generate a realistic (self-similar) Ethernet traffic. But, as expected, is still unable to represent well the particular features of the original traffic trace. This is a result of the, still, poor stochastic modelling. The next job will be implementing a significative modelling of the original traffic trace, through the specified methodology. We expect more significant results, them. Also, we will expand the framework to others workload platforms and compare the results. Packet acceleration could speed-up the performance, which may enable the reproduction of high-throughput traces. This can be implemented using DPDK. Finally, the results should be compared to Swing, on realism, similarity, and performance. This will give a measurement of how good is the framework, compared with others alternatives, and its strong and weak points.

References

[1] Nicolas Larrieu Antoine Varet. Realistic network traffic profile generation: Theory and practice. *Computer and Information Science*, 7(2), 2014.

[2] G. Bartlett and J. Mirkovic. Expressing different traffic models using the legotg framework. In *2015 IEEE 35th International Conference on Distributed Computing Systems Workshops*, pages 56–63, June 2015.

[3] A. Botta, A. Dainotti, and A. Pescape. Do you trust your software-based traffic generator? *IEEE Communications Magazine*, 48(9):158–165, Sept 2010.

[4] Alessio Botta, Alberto Dainotti, and Antonio Pescapé. A tool for the generation of realistic network workload for emerging networking scenarios. *Computer Networks*, 56(15):3531 – 3547, 2012.

[5] Ewerton Castro, Ajey Kumar, Marcelo S. Alencar, and Iguatemi E.Fonseca. A packet distribution traffic model for computer networks. In *Proceedings of the International Telecommunications Symposium – ITS2010*, September 2010.

[6] Bo Han, V. Gopalakrishnan, Lusheng Ji, and Seungjoon Lee. Network function virtualization: Challenges and opportunities for innovations. *Communications Magazine, IEEE*, 53(2):90–97, Feb 2015.

[7] W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson. On the self-similar nature of ethernet traffic (extended version). *IEEE/ACM Transactions on Networking*, 2(1):1–15, Feb 1994.

[8] Cesar A.V. Melo and Nelson L.S. da Fonseca. Envelope process and computation of the equivalent bandwidth of multifractal flows. *Computer Networks*, 48(3):351 – 375, 2005. Long Range Dependent Traffic.

[9] S. Molnár, P. Megyesi, and G. Szabó. How to validate traffic generators? In *2013 IEEE International Conference on Communications Workshops (ICC)*, pages 1340–1344, June 2013.

[10] L. O. Ostrowsky, N. L. S. da Fonseca, and C. A. V. Melo. A traffic model for udp flows. In *2007 IEEE International Conference on Communications*, pages 217–222, June 2007.

[11] Joel Sommers, Hyungsuk Kim, and Paul Barford. Harpoon: A flow-level traffic generator for router and network tests. *SIGMETRICS Perform. Eval. Rev.*, 32(1):392–392, June 2004.

[12] S. Srivastava, S. Anmulwar, A. M. Sapkal, T. Batra, A. K. Gupta, and V. Kumar. Comparative study of various traffic generator tools. In *Engineering and Computational Sciences (RAECS), 2014 Recent Advances in*, pages 1–6, March 2014.

[13] K. V. Vishwanath and A. Vahdat. Swing: Realistic and responsive network traffic generation. *IEEE/ACM Transactions on Networking*, 17(3):712–725, June 2009.

X Encontro de Alunos e Docentes do DCA/FEEC/UNICAMP (EADCA) Campinas, 26 e 27 de outubro de 2017
X DCA/FEEC/University of Campinas (UNICAMP) Workshop (EADCA) Campinas, Brazil, October 26-27, 2017

Using BIC and AIC for Ethernet traffic model selection. Is it worth?

Anderson dos Santos Paschoalon , Christian Esteve Rothenberg

Departamento de Engenharia de Computação e Automação Industrial (DCA)
Faculdade de Engenharia Elétrica e de Computação (FEEC)
Universidade Estadual de Campinas (Unicamp)
Caixa Postal 6101, 13083-970 – Campinas, SP, Brasil

{apaschoalon,chesteve}@dca.fee.unicamp.br

Abstract – In this work, we aim to evaluate how good are the information criteria AIC and BIC inferring which is the best stochastic process to describe Ethernet inter-packet times. Also, we check if there is a practical difference between using AIC or BIC. We use a set of stochastic distributions to represent inter-packet of a traffic trace and calculate AIC and BIC. To test the quality of BIC and AIC guesses, we define a cost function based on the comparison of significant stochastic properties for internet traffic modeling, such as correlation, fractal-level and mean. Then, we compare both results. In this short paper, we present just the results of a public free Skype-application packet capture, but we provide as reference further analyzes on different traffic traces. We conclude that for most cases AIC and BIC can guess right the best fitting according to the standards of Ethernet traffic modeling.

Keywords – BIC, AIC, stochastic function, inter-packet times, correlation, Hurst exponent, heavy-tailed distribution, fractal-level, burstiness, linear-regression, weibull, pareto, exponential, normal, poison, maximum likelihood, Ethernet traffic, traffic modeling, fractal-level, pcap file, Skype traffic

1. Introduction

There are many works devoted to studying the nature of the Ethernet traffic [1]. Classic Ethernet models use Poisson related processes. Initially, it makes sense since a Poisson-related process represents the probability of events occur in many independent sources with a known average rate, and independently of the last occurrence [1] [2]. However studies made by Leland et al. [1] showed that the Ethernet traffic has a self-similar and fractal nature. Even if they can represent the randomness of an Ethernet traffic, simple Poisson processes can't express traffic "burstiness" in a long-term time scale, such as traffic "spikes" on long-range ripples. These characteristics are an indication of the fractal and self-similar nature of the traffic that usually we express by distributions with infinite variance, called heavy-tailed. Heavy-tail means its distribution is not exponentially bounded[3], such as Weibull, Pareto and Cauchy distribution. Heavy-tailed processes may guarantee self-similarity, but not necessarily will ensure other important features like high correlation between data and same mean packet rate.

Many investigations were made on the literature about the nature of the Internet traffic [1][4][5][6][7], and many others on the modeling of stochastic functions for specific scenarios [8][9][10][11][12][9]. However, there are some limitations on this idea of finding a single model. Usually, not the same stochastic distribution will present a proper fitting for all possible kinds of traces [3]. Depending on some variables, such as the capture time, the number of packets or type of traffic, different functions may fit better the available data. On most works the best model representation for an Ethernet traffic is not chosen analytically but based on the researcher own data analyses and purposes [13][11][12]. Also, some methods like linear regression may diverge sometimes. Furthermore, it has already been proven that a single model cannot represent arbitrary traffic traces [3].

In this work we test the use of information criteria BIC (Bayesian information criterion) and AIC (Akaike information criterion) as tool for choosing the best fitting for inter-packet times of a traffic trace. It is an analytical method which spares and avoid human analyzes, is easy to be implemented by software, and don't relies on simulations and generation of random data. We fit a set of stochastic models through different methods and applying BIC and AIC to choose the best. On this article, we analyze the results of inter-packet time fitting for one public available trace we call as *skype-pcap*[1].

First, we explain the mathematical meaning of BIC and AIC and state the methods we are going to use to create a set of candidate models for our dataset. Then we define our cross-validation method based on a cost function J, attributing weights from the best to the worst representation for each properties using randomly generated data with our stochastic fittings, we can choose the best possible

[1]It is a lightweight Skype capture, available at https://wiki.wireshark.org/SampleCaptures, named *SkypeIRC.cap*

traffic model among these fittings. Thus we compare the results achieved by AIC/BIC and our cost function. Showing that BIC and AIC are good at guessing the model with smaller J values. Also, we found that for traffic inter-packet times, that the difference between BIC and AIC values is minimal. So choosing one over the other do not seem to be a key question.

2. AIC and BIC

Suppose that we have an statistical model M of some dataset $\boldsymbol{x} = \{x_1, ..., x_n\}$, with n independent and identically distributed observations of a random variable X. This model can be expressed by a probability density function (PDF) $f(x|\boldsymbol{\theta})$, where $\boldsymbol{\theta}$ is a vector of parameter of the PDF, $\boldsymbol{\theta} \in \mathbb{R}^k$ (k is the number of parameters). The likelihood function of this model M is given by:

$$L(\boldsymbol{\theta}|\boldsymbol{x}) = f(x_1|\boldsymbol{\theta}) \cdot ... \cdot f(x_n|\boldsymbol{\theta}) = \prod_{i=1}^{n} f(x_i|\boldsymbol{\theta}) \quad (1)$$

Now, suppose we are trying to estimate the best statistical model, from a set $M_1, ..., M_n$, each one with an estimated vector of parameters $\hat{\boldsymbol{\theta_1}}, ..., \hat{\boldsymbol{\theta_n}}$. AIC and BIC are defined by:

$$AIC = 2k - \ln(L(\hat{\boldsymbol{\theta}}|\boldsymbol{x})) \quad (2)$$

$$BIC = k \ln(n) - \ln(L(\hat{\boldsymbol{\theta}}|\boldsymbol{x})) \quad (3)$$

In both cases, the preferred model M_i, is the one with the smaller value of AIC_i or BIC_i.

3. Methodology

We collect inter-packet times from the traffic capture we call *skype-pcap*. Then, we estimate a set of parameters for stochastic processes, using a set of different methodologies, including linear-regression, maximum likelihood, and direct estimation. We are modeling:

- Weibull, exponential, Pareto and Cauchy distributions, using linear regression, through the Gradient descendent algorithm. We refer to these exponential and Pareto approximations as Exponential(LR) and Pareto(LR);
- Normal and exponential distribution, using direct estimation the mean and the standard deviation of the dataset for the normal, and the mean for the exponential. We refer for to this exponential approximation as Exponential(Me) ;
- Pareto distribution, using the maximum likelihood method. We refer to this distribution as Pareto(MLH);

Then, from these parametrized models, we estimate which one best represent our dataset, using AIC and BIC criteria. These results were obtained using Octave language[2], and the scripts are available at [14] for reproduction purposes. Thus, to see if our criterion of parameter selection can find which is the best model according to traffic modeling standards on realism and benchmarking[15], we define a validation methodology. We randomly generated a dataset using our parameterized stochastic processes. Then we compare it with the original and synthetic sample, trough three different metrics, all with a confidence interval of 95%:

- Correlation between the sample data and the estimated model (Pearson's product-moment coefficient);
- Difference between the original and the synthetic Hurst exponent;
- Difference between the original and the synthetic mean inter-packet time;

The Pearson's product-moment coefficient, or simply correlation coefficient, is an expression of the linear dependence or association between two datasets. To estimate it, we use the Octave's function `corr()`. The Hurst exponent is meter self-similarity and indicates the fractal level of the inter-packet times. To estimate this value we use the function `hurst()` from Octave, which uses rescaled range method. Finally, the mean is also relevant, since it will meters if the packet rate of the approximation and the original trace are close to each other.

To measure if AIC and BIC are suitable criteria for model selection for inter-packet times, we define a cost function based on the correlation, Hurst exponent and mean. We define Cr as the vector of correlations of the models ordered from the greater to the smaller. Also, let the vectors Me and Hr be the absolute difference (modulus of the difference) between the estimated models and the original datasets of the mean and the Hust exponent respectively. We order both from the smaller

[2] https://www.gnu.org/software/octave/

Table 1. Results of our modeling and simulation methodology for the traffic trace *skype-pcap*. The stochastic processes are ordered form the worst to the best fitting, according to AIC and BIC.

Function	AIC	BIC	Parameters	
Weibull	-2293.8	-2283.8	$\alpha : 0.522$	$\beta : 0.097$
Exponential (Me)	-426.13	-421.1	$\lambda : 3.319$	
Exponential (LR)	96.9	101.8	$\lambda : 1.505$	
Pareto (MLH)	361.9	371.8	$\alpha : 0.0747$	$x_m : 5e-8$
Normal	2423.8	2433.8	$\mu : 0.301$	$\sigma : 0.749$
Pareto (LR)	6411.0	6421.08	$\alpha : 0.413$	$x_m : 5e-8$
Cauchy	13464.6	13474.5	$\gamma : 0.000275$	$x_0 : 0.219$

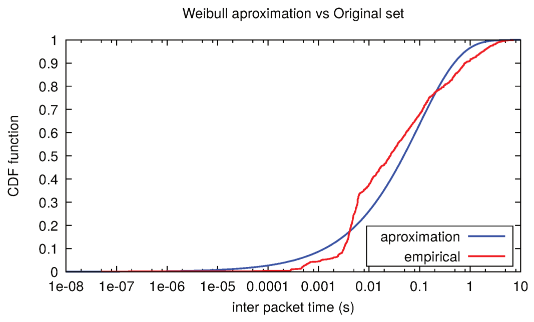


Figure 1. Cumulative distribution function(CDF) for Weibull fitting and empirical data for inter-packet times of *skype-pcap*.

to the greatest values. Letting $\phi(V, M)$ be an operator who gives the position of a model M in a vector V, we define the cost function J as:

$$J(M) = \phi(Cr, M) + \phi(Me, M) + \phi(Hr, M) \quad (4)$$

The smaller is the cost J, the best is the model. Then we compare the results achieved by AIC and BIC, and J.

4. Results

In table 1 we summarize our estimations for AIC, BIC, and the stochastic process estimated parameters. Then we organize the values in crescent order of quality, according to the selection criteria (the smaller, the better). In figure 1 we present the best fitting chosen both by BIC and AIC criteria. It is on log-scale, which provides a better visualization for small time values. Visually we can see that linear regression with Weibull distribution was able to provide a good approximation for this dataset.

The difference between BIC and AIC values in all simulations are much smaller than the difference between the distributions. This result indicates that for inter-packet times, using AIC or BIC to pick a model, do not influence the results significantly. According to BIC and AIC previsions, Weibull and Exponential (Me and LR) are the best options, and the cost functions gave exact this same order 2. In fact, Weibull pointed as the best stochastic function by BIC and AIC, has half of the penalty imposed by the cost function J. The following models however are not in the same order since some results are flipped. But still, no opposite correspondence can be found. No result found by AIC and BIC were far from the one pointed by J. An important observation to be made here is about the fact that each stochastic function may guarantee different porperties. For example, the Exponential(Me) guarantee the same mean packet rate from the original. Heavy-tailed distributions are more likely to guarantee

5. Conclusion

In this work, we analyze how BIC and AIC perform being used as analytical selection criteria form stochastic models for Ethernet inter-packet times. Using a cross-validation methodology based on the generation of random data using these models, and pointing a cost function. We saw that both AIC or

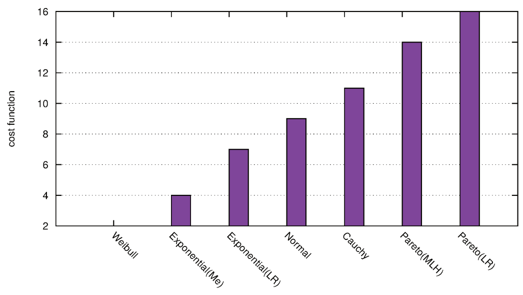


Figure 2. Cost function J of each stochastic process for *skype-pcap*.

BIC and the cost function were able to pick the first models in the same order. Therefore, analytically with BIC and AIC, we were able to achieve the same results as pointed by our simulations. Even if AIC and BIC mathematical definitions are unaware of the specific requirements of Ethernet traffic modeling, such as same fractal-level and close packet per second rate, they still can point the best choices according to these constraints. In this work, we analyze just inter-packet times of a single trace. However at [14] we perform the same methodology on different types of traffic captures, finding similar results. Therefore, we can conclude that BIC and AIC are healthy alternatives for model selection of Ethernet inter-packet times models and we can safely use them. Finally, we must point some advantages of BIC and AIC instead of simulations. Since it is an analytical model, no generation of random data is necessary, being computationally cheaper and easy to code. Also, since we do not use a single stochastic function and parameterization strategy, it is resilient to the fact that some methods like linear-regression over Weibull may diverge sometimes. If it happens, BIC or AIC will discard this guesses, and choose another one automatically. Last but not least, to the best of our knowledge, this is the most comprehensive investigation of the actual quality of BIC and AIC as model selection criteria of for inter-packet times.

References

[1] W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson. On the self-similar nature of ethernet traffic (extended version). *IEEE/ACM Transactions on Networking*, 2(1):1–15, Feb 1994.

[2] Frank A. Haight. *Handbook of the Poisson Distribution*. John Wiley & Son, New York, 1967.

[3] Nicolas Larrieu Antoine Varet. Realistic network traffic profile generation: Theory and practice. *Computer and Information Science*, 7(2), 2014.

[4] F. Ju, J. Yang, and H. Liu. Analysis of self-similar traffic based on the on/off model. In *2009 International Workshop on Chaos-Fractals Theories and Applications*, pages 301–304, Nov 2009.

[5] Zhao Rongcai and Zhang Shuo. Network traffic generation: A combination of stochastic and self-similar. In *Advanced Computer Control (ICACC), 2010 2nd International Conference on*, volume 2, pages 171–175, March 2010.

[6] W. Willinger, M. S. Taqqu, R. Sherman, and D. V. Wilson. Self-similarity through high-variability: statistical analysis of ethernet lan traffic at the source level. *IEEE/ACM Transactions on Networking*, 5(1):71–86, Feb 1997.

[7] Ibrahim Cevizci, Melike Erol, and Sema F. Oktug. Analysis of multi-player online game traffic based on self-similarity. In *Proceedings of 5th ACM SIGCOMM Workshop on Network and System Support for Games*, NetGames '06, New York, NY, USA, 2006. ACM.

[8] N. M. Markovitch and U. R. Krieger. Estimation of the renewal function by empirical data-a bayesian approach. In *Universal Multiservice Networks, 2000. ECUMN 2000. 1st European Conference on*, pages 293–300, 2000.

[9] A. J. Field, U. Harder, and P. G. Harrison. Measurement and modelling of self-similar traffic in computer networks. *IEE Proceedings - Communications*, 151(4):355–363, Aug 2004.

[10] T. Kushida and Y. Shibata. Empirical study of inter-arrival packet times and packet losses. In *Proceedings 22nd International Conference on Distributed Computing Systems Workshops*, pages 233–238, 2002.

[11] P. M. Fiorini. On modeling concurrent heavy-tailed network traffic sources and its impact upon qos. In *1999 IEEE International Conference on Communications (Cat. No. 99CH36311)*, volume 2, pages 716–720 vol.2, 1999.

[12] F. D. Kronewitter. Optimal scheduling of heavy tailed traffic via shape parameter estimation. In *MILCOM 2006 - 2006 IEEE Military Communications conference*, pages 1–6, Oct 2006.

[13] S. D. Kleban and S. H. Clearwater. Hierarchical dynamics, interarrival times, and performance. In *Supercomputing, 2003 ACM/IEEE Conference*, pages 28–28, Nov 2003.

[14] Projeto mestrado. https://github.com/AndersonPaschoalon/ProjetoMestrado, 1234. [Online; accessed May 30th, 2017].

[15] Sandor Molnar, Peter Megyesi, and Geza Szabo. How to validate traffic generators? In *2013 IEEE International Conference on Communications Workshops (ICC)*, pages 1340–1344, June 2013.

Automated Selection of Inter-Packet Time Models through Information Criteria

Anderson dos Santos Paschoalon and Christian Esteve Rothenberg
School of Electrical and Computer Engineering (FEEC)
University of Campinas (UNICAMP), Campinas, Sao Paulo, Brazil
Email: anderson.paschoalon,chesteve@dca.fee.unicamp.br

***Abstract*—A well-known problem of network traffic representation over time is that there is no "one-fits-all" model. The selection of the "best" model is traditionally made in a time-consuming and ad-hoc manner by human experts. In this work, we evaluate the feasibility of using Bayesian Information Criterion (BIC) and Akaike Information Criterion (AIC) as tools for automated selection of the best-fit stochastic process for inter-packet times. We propose and validate a methodology based on Information Criteria, resulting in an automated and accurate approach for such traffic modelling tasks.**

***Index Terms*—BIC, AIC, stochastic function, inter-packet times, Hurst exponent.**

I. Introduction

Traffic identification [1] and generator tools [2] [3] rely on a set of pre-defined stochastic models to set the packet classification/generation rules by configuring packet bursts and inter-packet times. Studies show that realistic network traffic provides different and more variable load characteristics on routers [4], even for the same average bandwidth. Bursty traffic can cause more packet buffer overflows on a given network [5], resulting in higher network performance degradation than under constant-rate traffic [4].

Many efforts have been devoted to understanding the traffic nature, which has been proved to be self-similar and fractal [6] [7]. Classical network traffic models based on Poisson related processes cannot express well this type of scenarios. Therefore, research has been devoted to processes with high-variability [8]. For example, the use of heavy-tailed stochastic processes, such as Weibull, Pareto, and Cauchy, have non-exponentially bounded distributions [3] and can guarantee self-similarity via Joseph and Noah effects [8]. However, they do not necessarily ensure correlation on other quality measures between the model and the actual traffic, such as the average packet rate [9]. There are works that advocate for the use of Cauchy [5], Weibull [10], Bivariate gamma [11], and Moravian-related process [12], just to cite some.

While there is an extensive amount of study-cases on network traffic modeling, there is a gap of suitable generic methods for automating the choice of the "best" model. Specific models valid for some research studies do not guarantee that the same model will apply for new cases. Investigations point to the opposite direction: a change in the scenario can change the best model as well [5] [10]. Since no "one-fits-all" model is viable, the *status quo* of traffic modeling is to be done on an *ad-hoc* manner by human specialists [13]. Another option would be to simulate all outputs a given set of random processes and choose the model that best fits the data. However, this task turns into a research project itself, involving definition of metrics, random-data generation, cross-validation methods, repetitions to guarantee high confidence intervals, and so on. Therefore, such an approach is not practical if that is not the primary research target.

In this work, we propose and evaluate the use of the Information Criteria (IC), more specifically BIC (Bayesian Information Criterion) and AIC (Akaike Information Criterion) [14], as suitable methods for automated model selection for network traffic inter-packet times. Being analytic and deterministic methods which spare model designer humans in the loop, they are also simple to implement and do not rely on hypothesis testing. In addition, We define a cross-validation method based on a cost function J, which acts as an aggregator of traditional and key metrics used for validation of stochastic models and traffic samples. J assigns weights from the best to the worst representation for each property of each trace model by using randomly generated data with our stochastic fittings. Through this process, we choose the best-fitted traffic model under evaluation. Afterward, we compare the results achieved by AIC/BIC and our cost function. Given the aforementioned approach, we show that AIC/BIC methods provide an accurate stochastic process selection strategy for inter-packet times models. Some marginal caveats include limiting our work to independent, and identical distributed random variables, since they are commonly used to describe network traffic [5] and are widely supported in traffic generators [2]. Information criteria on more complex models such as Markov-chain and envelope processes [15] have been left for future work.

II. A Primer on BIC and AIC

Let M represent a statistical model of some dataset $\boldsymbol{x} = \{x_1, ..., x_n\}$, with n independent and identically distributed observations of a random variable X. This model can be expressed by a probability density function (PDF) $f(x|\boldsymbol{\theta})$, where $\boldsymbol{\theta}$ is a vector of the PDF's parameters, $\boldsymbol{\theta} \in \mathbb{R}^k$ (k is the $\boldsymbol{\theta}$'s dimension). The likelihood function of this model M is given by [14]:

$$L(\boldsymbol{\theta}|\boldsymbol{x}) = f(x_1|\boldsymbol{\theta}) \cdot ... \cdot f(x_n|\boldsymbol{\theta}) = \prod_{i=1}^{n} f(x_i|\boldsymbol{\theta}) \qquad (1)$$

The goal is to estimate the best statistical model, from the set $\{M_1, ..., M_n\}$, where each one has an estimated vector of parameters $\hat{\boldsymbol{\theta}_1}, ..., \hat{\boldsymbol{\theta}_n}$. AIC and BIC are defined by:

$$AIC = 2k - 2\ln(L(\hat{\boldsymbol{\theta}}|\boldsymbol{x})) \tag{2}$$

$$BIC = k\ln(n) - 2\ln(L(\hat{\boldsymbol{\theta}}|\boldsymbol{x})) \tag{3}$$

In both cases, the preferred model M_i, is the one with the smaller value of AIC_i or BIC_i.

III. Methodology

We used four packet captures (*pcaps*) to extract inter-packet times we used in this work, where three of them are publicly available. The first one is a Skype packet capture[1], which we name *skype-pcap*. The second one is a CAIDA capture[2], from which we use its first capture second[3], referred to as *wan-pcap*. The third one is a capture of a busy private network Internet access point[4], which is referred as *lan-gateway-pcap*. Finally, we capture the last traffic trace at our INTRIG/UNICAMP laboratory LAN through a period of one hour on a firewall gateway. We call it *lan-firewall-pcap*. All the developed scripts and data sets are publically available [16], for reproducibility purposes. The results were obtained using the Octave tool.[5] We retrieved inter-packet times from the traffic traces and divided them into two equally sized datasets. to avoid data *over-fitting*, we use odd-indexed elements as *training dataset*, and even-indexed as *cross-validation dataset*. We then apply the *training dataset* on several techniques for model estimation:

- Weibull, exponential, Pareto and Cauchy distributions: We use linear regression through the Gradient descendent algorithm. Also, we refer to these exponential and Pareto approximations as Exponential (LR) and Pareto (LR);
- Normal and exponential distributions: We approximate the mean and variance by the average and standard deviation on the normal, and the rate by the inverse of the average on the exponential, we refer as Exponential (Me);
- Pareto distribution: We use the maximum likelihood method, which we refer to as Pareto (MLH);

Given the seven models (*hypothesis*) above, we then compute a quality ranking to evaluate AIC and BIC using the *cross-validation* dataset. To validate the information criteria effectiveness, we develop a weight system based on traditional methodologies for model quality verification and synthetic traffic validation [9] [6]. First, we randomly generate datasets following each stochastic processes hypothesis resulting in the synthetic inter-packet times, which are then compared with the *cross-validation* dataset based on the following metrics:

[1]Available at https://wiki.wireshark.org/SampleCaptures, named *SkypeIRC.cap*

[2]http://www.caida.org/home/

[3]Available at https://data.caida.org/datasets/passive-2016/equinix-chicago/20160121-130000.UTC, named as *equinix-chicago.dirB.20160121-135641.UTC.anon.pcap.gz*

[4]Available at http://tcpreplay.appneta.com/wiki/captures.html named *bigFlows.pcap*

[5]https://www.gnu.org/software/octave/

- The Pearson's product-moment coefficient between the sample data and the estimated model. The closer to one, the better;
- Hurst exponent estimation, via range re-scaling. The closer to the *cross-validation* Hurst value, the better;
- Average inter-packet time. The closer to the cross-validation dataset average, the better.

We choose these metrics according to traffic standards on realism and benchmarking [9]. The "Pearsons product-moment coefficient" is a measure of the correlation[6] between datasets. The Hurst exponent is a measure of self-similarity [6][7] and indicates the fractal level of the distribution of inter-packet times within a trace. Finally, a trace's average inter-packet time is inversely proportional to its packet rate. The closer the model's average inter-packet is to the original, the closer will also be its packet rate and throughput [9]. We consolidate all these metrics in a best-effort weight system, we call cost function J. Let Cr be the array of correlations between the randomly generated data and the *cross-validation dataset*, sorted from the better (greater) to the worst (smaller). Let Me and Hr be defined as vectors of absolute difference of the mean and Hurst exponent between the synthetic and the *cross-validation dataset*. These vectors are sorted: the lower the differences, the better the model hypothesis represents the same cross-validation measured metric (throughput and fractal-level). Letting $\phi(V, M)$ be an operator giving the position (starting from 0) of a model M in a vector V, we define the cost function J as:

$$J(M) = \phi(Cr, M) + \phi(Me, M) + \phi(Hr, M) \tag{4}$$

To illustrate an example application, suppose a model m_1 with the best correlation, second and third smaller values of Hr and Me, respectively, would result in: $J(m_1) = 0+1+2 = 3$. Therefore, the smaller J, the better the model to represent a wide range of different metrics, since it consolidates many widely adopted metrics [9] in a single value or *ranking*. The estimation of these values was repeated 30 times, with a confidence interval of 95%, small enough to not interfere with the results. If the information criteria and J returns related results, this is interpreted as a strong indication of the reliability and robustness of AIC and BIC.

IV. Results

Table I summarizes the estimates obtained for AIC, BIC, and the stochastic process estimated parameters for all *pcap* traces. Each model order is graphically presented in Figure 1. For all *pcap* experiments, we verify that the difference between BIC and AIC for a given function is always smaller than its value among different distributions. As shown in the table I, AIC and BIC criteria always pointed to the same model ordering. Table II presents the percentage difference between the obtained values. We verify that their values tend to converge when the dataset increases.

[6]Octave's function `corr()`

[7]Octave's function `hurst()`, which uses the re-scaled range method.

TABLE I: Experimental results, including the estimated parameters and the BIC and AIC values of the four pcap traces.

	Trace							
Function	AIC		Parameters		AIC	BIC	Parameters	
	skype-pcap				lan-firewall-pcap			
Cauchy	$6.94E+03$	$6.95E+03$	$\gamma: 1.71E-04$	$x_0: 1.88E-01$	$-2.29E+05$	$-2.29E+05$	$\gamma: 1.93E-02$	$x_0: -4.97E-02$
Exponential(LR)	$-4.70E+01$	$-4.28E+01$	$\lambda: 1.79E+00$		$-2.22E+06$	$-2.22E+06$	$\lambda: 4.05E-01$	
Exponential(Me)	$-2.16E+02$	$-2.12E+02$	$\lambda: 3.45E+00$		$3.63E+05$	$3.63E+05$	$\lambda: 1.13E+02$	
Normal	$1.21E+03$	$1.22E+03$	$\mu: 2.90E-01$	$\sigma: 6.95E-01$	$-1.48E+06$	$-1.48E+06$	$\mu: 8.85E-03$	$\sigma: 3.49E-02$
Pareto(LR)	$3.38E+03$	$3.39E+03$	$\alpha: 4.28E-01$	$x_m: 5.00E-08$	Inf[1]	Inf[1]	$\alpha: 2.51E-01$	$x_m: 5.00E-08$
Pareto(MLH)	$1.88E+02$	$1.97E+02$	$\alpha: 7.48E-02$	$x_m: 5.00E-08$	$-1.80E+06$	$-1.80E+06$	$\alpha: 1.15E-01$	$x_m: 5.00E-08$
Weibull	$-1.15E+03$	$-1.14E+03$		$\beta: 9.68E-02$	$-1.97E+06$	$-1.97E+06$	$\alpha: 3.46E-01$	$\beta: 1.79E-03$
	lan-gateway-pcap				wan-pcap			
Cauchy	$3.65E+06$	$3.65E+06$	$\gamma: 1.95$	$x_0: -4.45E+03$	$2.99E+07$	$2.99E+07$	$\gamma: 8.17E+02$	$x_0: -4.45E+03$
Exponential(LR)	$3.67E+06$	$3.67E+06$	$\lambda: 9.75E-03$		$2.84E+07$	$2.84E+07$	$\lambda: 2.20E-05$	
Exponential(Me)	$-5.44E+06$	$-5.44E+06$	$\lambda: 2.64E+03$		$-3.29E+07$	$-3.29E+07$	$\lambda: 6.58E+05$	
Normal	$-4.67E+06$	$-4.67E+06$	$\mu: 3.79E-04$	$\sigma: 1.00E-06$	$-3.19E+07$	$-3.19E+07$	$\mu: 2.00E-06$	$\sigma: 1.00E-06$
Pareto(LR)	$-5.13E+06$	$-5.13E+06$	$\alpha: 1.49E-01$	$x_m: 5.00E-08$	$4.51E+07$	$4.51E+07$	$\alpha: 4.00E-14$[2]	$x_m: 5.00E-08$
Pareto(MLH)	$-5.13E+06$	$-5.13E+06$	$\alpha: 1.36E-01$	$x_m: 5.00E-08$	$-3.13E+07$	$-3.13E+07$	$\alpha: 3.39E-01$	$x_m: 5.00E-08$
Weibull	$-5.50E+06$	$-5.50E+06$	$\alpha: 2.81E-01$	$\beta: 1.00E-06$	$-2.73E+07$	$-2.73E+07$	$\alpha: 7.64E-02$	$\beta: 1.00E-06$

[1] The computation of the likelihood function has exceeded the computational precision used, so it was the highest AIC and BIC for this trace.
[2] The linear regression did not converge to a valid value, so we used a small value instead to perform the computations.

TABLE II: Relative difference(%) between AIC and BIC.

	skype-pcap	lan-gateway -pcap	wan-pcap	lan-firewall -pcap
Weibull	$7.47E-01$	$3.96E-04$	$8.86E-05$	$9.21E-04$
Normal	$7.04E-01$	$4.66E-04$	$7.58E-05$	NaN
Exponential(LR)	$9.54E+00$	$2.97E-04$	$4.26E-05$	$2.81E-03$
Exponential(Me)	$2.00E+00$	$2.00E-04$	$3.68E-05$	$6.90E-04$
Pareto(LR)	$2.53E-01$	$4.25E-04$	$5.36E-05$	$1.13E-03$
Pareto(MLH)	$4.45E+00$	$4.25E-04$	$7.74E-05$	$1.04E-03$
Cauchy	$1.23E-01$	$5.97E-04$	$8.08E-05$	$8.90E-03$

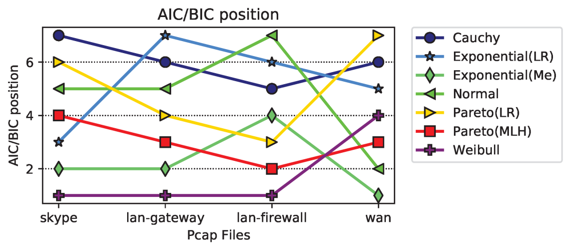


Fig. 1: Comparison of the quality order of each model given by AIC and BIC

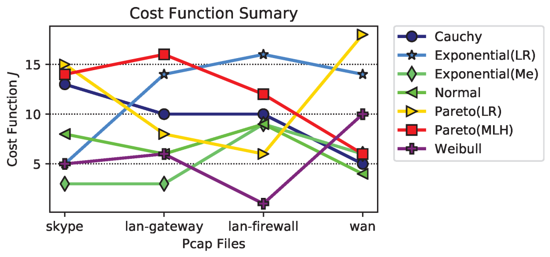


Fig. 2: Cost function for each one of the datasets used in this validation process.

Figure 2 illustrates the cost function values for all the models on each *pcap* file. For example, for *skype-pcap*, BIC and AIC point that Weibull and Exponential (Me) are the best representation for the traffic trace. The cost function used for cross-validation points both as best options, along with Exponential (LR). To simplify the visualization and comparison of the differences between the rankings given by both methodologies, Figure 3 presents a chart with the relative differences from the order of each model. Taking as a reference the position of each model given by J, we sorted them from the better to the worst (0 to 6, on the x-axis), and measured the position distance with the ones given by the information criteria. Since the worst case for this value is 6 (opposite correspondence), we draw a line on the average: the expected value in the case no positive or negative correspondence existed between both information criteria and J. Using the ϕ operator, as defined before, we can calculate the ranking delta, as explained, for the *i-th* model by:

$$\delta(m_i) = \phi(Jv, m_i) - \phi(IC, m_i) \tag{5}$$

where Jv and IC are the ordered pairs vectors on models and cost functions/information criteria, from the best to the worst, respectively. We can observe that in most cases, the information criteria and the cost function choose the best models in a similar order. A hypothesis ranked as good by one tends to be ranked also as good by the other. For the 28 possible study cases, 19 (68%) resulted in the same ranking or at most one position difference. In addition, AIC/BIC tend to prioritize most of the heavy-tailed processes, such as Weibull and Pareto (except of Cauchy). This is a useful feature when the scaling and long-range characteristics of the traffic have to be prioritized by the selected model.

Finally, we observe AIC and BIC presenting a bias in favor of Pareto (MLH). Even though it was never ranked as the best model, it was always better positioned by AIC and BIC than by J. We explain this result by the fact that AIC and BIC calculation uses the model likelihood, which Pareto (MLH) maximizes. This effect is clear on the *lan-firewall-pcap*. Figure 4 presents results from the cross-validation dataset, where we can observe the best fitting pointed by both methods (Weibull), and the second-best indicated by J (Pareto (LR))

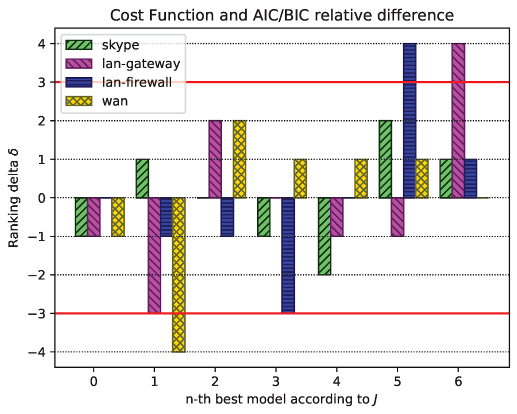


Fig. 3: Comparison of the model selection order for BIC/AIC and the cost function J for each *pcap* traffic trace.

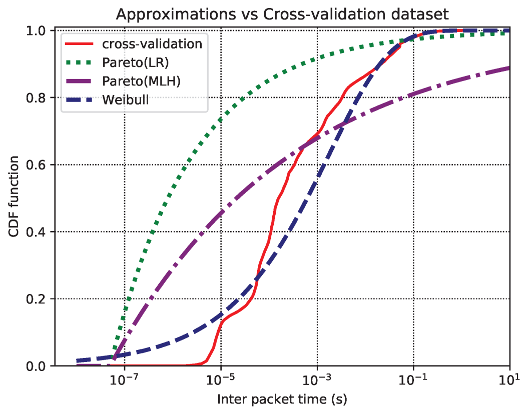


Fig. 4: Inter-packet times CDF function and stochastic models for *firewall-pacap*.

and by AIC/BIC (Pareto (MLH)). Even though Pareto (MLH) presents a good performance representing small values, about 10% of the inter-packet times are higher than 10 seconds, a prohibitive high value that overall turns Pareto (LR) into a better performing option.

V. Conclusion

This work presents and evaluates a method based on BIC and AIC for automated selection criteria of the best stochastic process to model network traffic in terms of inter-packet times. Through a cross-validation methodology based on random data generation following the selected models and cost function measurements, we observe that the proposed methodology is able to accurately pick the first models in the same order, in support of the feasibility and automation benefits of using Information Criteria as reliable model selectors for network network. We conclude that BIC and AIC are suitable alternatives to derive realistic network traffic models that could be used for diverse scenarios to add useful and efficiently add realism to experiments based on synthetic traffic generation or network traffic identification. One identified caveat is the use of the Maximum Likelihood method, which can over-prioritized some models over more performing ones. As future work, we will investigate the use of different and more complex stochastic processes such as Markovian-related and Envelope processes, beyond the scope of this article.

References

[1] M. Jaber, R. G. Cascella, and C. Barakat, "Can we trust the inter-packet time for traffic classification?" in *2011 IEEE International Conference on Communications (ICC)*, June 2011, pp. 1–5.

[2] A. Botta, A. Dainotti, and A. Pescape, "A tool for the generation of realistic network workload for emerging networking scenarios," *Computer Networks*, vol. 56, no. 15, pp. 3531 – 3547, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128612000928

[3] A. Varet and N. Larrieu, "Realistic network traffic profile generation: Theory and practice," *Computer and Information Science*, vol. 7, no. 2, 2014.

[4] J. Sommers and P. Barford, "Self-configuring network traffic generation," in *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '04. New York, NY, USA: ACM, 2004, pp. 68–81. [Online]. Available: http://doi.acm.org/10.1145/1028788.1028798

[5] A. J. Field, U. Harder, and P. G. Harrison, "Measurement and modelling of self-similar traffic in computer networks," *IEE Proceedings - Communications*, vol. 151, no. 4, pp. 355–363, Aug 2004.

[6] W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson, "On the self-similar nature of ethernet traffic (extended version)," *IEEE/ACM Transactions on Networking*, vol. 2, no. 1, pp. 1–15, Feb 1994.

[7] F. Ju, J. Yang, and H. Liu, "Analysis of self-similar traffic based on the on/off model," in *2009 International Workshop on Chaos-Fractals Theories and Applications*, Nov 2009, pp. 301–304.

[8] W. Willinger, M. S. Taqqu, R. Sherman, and D. V. Wilson, "Self-similarity through high-variability: statistical analysis of ethernet lan traffic at the source level," *IEEE/ACM Transactions on Networking*, vol. 5, no. 1, pp. 71–86, Feb 1997.

[9] S. Molnar, P. Megyesi, and G. Szabo, "How to validate traffic generators?" in *2013 IEEE International Conference on Communications Workshops (ICC)*, June 2013, pp. 1340–1344.

[10] S. D. Kleban and S. H. Clearwater, "Hierarchical dynamics, interarrival times, and performance," in *Supercomputing, 2003 ACM/IEEE Conference*, Nov 2003, pp. 28–28.

[11] A. Bhattacharjee and S. Nandi, "Bivariate gamma distribution: A plausible solution for joint distribution of packet arrival and their sizes," in *2010 13th International Conference on Computer and Information Technology (ICCIT)*, Dec 2010, pp. 125–130.

[12] P. M. Fiorini, "On modeling concurrent heavy-tailed network traffic sources and its impact upon qos," in *1999 IEEE International Conference on Communications (Cat. No. 99CH36311)*, vol. 2, 1999, pp. 716–720 vol.2.

[13] P. Tune, M. Roughan, and K. Cho, "A comparison of information criteria for traffic model selection," in *2016 10th International Conference on Signal Processing and Communication Systems (ICSPCS)*, Dec 2016, pp. 1–10.

[14] A. Chakrabarti and J. K. Ghosh, "Aic, bic and recent advances in model selection," in *Philosophy of Statistics*, ser. Handbook of the Philosophy of Science, P. S. Bandyopadhyay and M. R. Forster, Eds. Amsterdam: North-Holland, 2011, vol. 7, pp. 583 – 605. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780444518620500186

[15] C. A. Melo and N. L. da Fonseca, "Envelope process and computation of the equivalent bandwidth of multifractal flows," *Computer Networks*, vol. 48, no. 3, pp. 351 – 375, 2005, long Range Dependent Traffic. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128604003305

[16] "Github code and data set repository," https://github.com/AndersonPaschoalon/aic-bic-paper, 2018, [Online].

SIMITAR: Realistic and Auto-configurable Traffic Generation

Anderson dos Santos Paschoalon
School of Electrical and Computer Engineering (FEEC)
University of Campinas (UNICAMP)
Campinas, Brazil
anderson.paschoalon@dca.fee.unicamp.br

Christian Esteve Rothenberg
School of Electrical and Computer Engineering (FEEC)
University of Campinas (UNICAMP)
Campinas, Brazil
chesteve@dca.fee.unicamp.br

***Abstract*—Evolving network technologies keep imposing challenges on the validation and performance evaluation processes. For instance, the amount and characteristics of flows impact the performance of SDN switches and controllers. Virtualization technologies, such as NFV, are harder to predict and to guarantee performance compared to hardware-based devices. IoT introduces new traffic patterns, such as machine-to-machine traffic and protocols adapted to resource-constrained scenarios, which will coexist with traditional traffic. Along with all these changes, in practice, real traffic behaves differently compared to constant-rate traffic, commonly used by typical benchmarking and experimental validations. The type of traffic matters. Bursty traffic may cause buffer overflows while constant traffic does not. Most of the tools for traffic generation offer a large set of options to be configured but are not auto-configurable, leaving the user in charge of the creation, parameterization, validation, and implementation of the traffic model. The production of actual realistic traffic is a challenging project by itself. In addition, the majority of open-source tools have the modeling layer coupled to the traffic generator. This paper presents our proposed tool called SIMITAR, an auto-configurable, realistic, and extensible traffic generator motivated by the needs of evolving networking environments such as NFV, SDN, and IoT**

***Index Terms*—Sniffing, traffic modelling, BIC, AIC, inter-packet times, Wavelet scalling, traffic generator, Burstiness, pcap file, linear regression, Iperf**

I. Introduction

The type of traffic used for performing evaluation matters; this is a fact. Studies show that realistic Ethernet traffic provides different and variable load characteristics on routers [1], even with the same average bandwidth consumption, showing that constant traffic is not sufficient for complete technology validation. This conclusion indicates that tests which employ traffic generators with constant rates are not enough for complete validation of new technologies. Bursty traffic can cause packet losses and buffer overflows, impacting network performance and measurement accuracy [2]. Small packets tend to degrade application performance [3]. Furthermore, realistic traffic is essential on security research, such as for the evaluation of firewall middleboxes, studies on intrusion, and malicious workloads [4].

New networking scenarios such as SDN and virtualized networks (NFV and VNFs) become harder to predict in terms of performance compared to hardware-based technologies, due to the multiple layers of software and platform parameters demanding validation in a broadening range of use cases [5]. In addition, new types of traffic patterns introduced by IoT and Machine-to-Machine communication [6] increase the complexity of the network traffic characterization, turning pre-defined models used by traffic generators obsolete.

Aiming at addressing these gaps, this paper introduces SIMITAR, an auto-configurable network traffic generator. SIMITAR stands for *SnIffing, ModellIng, and TrAffic geneRation*, which correspond to the main operation processes of the proposed framework. SIMITAR has an application independent traffic model, that can represent a wide variety of scenarios. It also decouples the traffic modeling and packet-generation layer, using a factory design pattern, enabling its application on different scenarios, and technology update, via technology abstraction. SIMITAR code and all scripts used in this paper are available at GitHub [7] for validation, experiment reproducibility, and re-use purposes.

II. Related Work

Traffic generators are tools to transfer or inject network packets in a controlled manner, aiming not at the actual data transfer data but at the functional validation and performance benchmarking of devices under test (DUT) for varying technologies or scenarios. The open-source community offers a vast variety of traffic generators. Since most have been built for specific goals, each uses different methods for traffic generation, and offer control over different traffic features, such as throughput, packet-sizes, protocols, and so on [4].

Traffic generators can be classified into two main groups: replay engines [10] and model-based tools. Replay engines, such as Tcpreplay and TCPivo [11], work replicating in a given network interface a given packet capture file. These tools can generate realistic traffic but have their constraints. They are deterministic since will always reproduce the same traffic from the packet capture. Replay engines require storage of packet capture, what can be a problem for traffics of high bandwidth traffic. Also, they assume the user has access to packet captures appropriate for his testing purposes, which is not always true, due to a limited number of public sources. Model-based tools rely on software models to replicate one or more characteristics of the traffic. Following the taxonomy presented by Botta et al. [12]:

TABLE I: Comparison of existing traffic generation tools.

Solution	Auto-configurable	Realistic Traffic	Traffic Custumization	Extensibility
Harpoon	yes	yes	yes	**no**
D-ITG	**no**	yes	yes	**no**
Swing [8]	yes	yes	**no**	**no**
Ostinato	**no**	**no**	yes	yes
LegoTG [9]	**no**	**no**	yes	yes
sourcesOnOff	**no**	yes	yes	**no**
Iperf	**no**	**no**	yes	**no**
SIMITAR	yes	yes	yes	yes

- **Application-level traffic generators**: they try to emulate network applications simulating real workloads stochastically and(or) responsively. Eg.: Surge, D-ITG[1].
- **Flow-level traffic generators**: they can reproduce features of flows, such as flow duration, a diurnal behavior. Eg.: Harpoon [1].
- **Packet-level traffic generators**: most traffic generators available fall in this class. They aim to reproduce physical features such as inter-packet times, packet size, bandwidth and packets per second. Eg.: Iperf[2], Ostinato[3], D-ITG [4], sourcesOnOff [10].
- **Multi-level traffic generators**: it proposes to take into account existing interaction among each layer of the network stack. Eg.: Swing [8].

Model-based tools have their limitations as well. Application-level traffic generators are designed to represent only specific scenarios on computer networking. Many of the packet-level traffic generator tools only offer constant-rate and Poisson models, which does not represent well the complexity of internet traffic [13]. Other tools such as D-ITG offer dozens of parameters and models to be configured, but delegate to the user the task of creating, validate and script his traffic model. To the best of our knowledge, we found only two open-source auto-configurable tools: Swing and Harpoon. However, none of them has an extensible architecture, which turns supporting modern and fast I/O APIs (such as DPDK[4]) a hard task. Table I presents a summary of the above-mentioned features mentioned above.

III. Architecture and Modelling

The SIMITAR architecture show in Figure 1 is composed of four components: (*i*) *Sniffer*, (*ii*) *SQLite database*, (*iii*) *Trace Analyzer*, and (*iv*) *Flow Generator*, which we describe next.

A. Sniffer and SQLite database

This component collects traffic information and classifies into flows through header field matching. It uses the same criteria used by SDN switches [14]: Link, Network and Transport protocols; Network Source and Destination Address; and Transport Source and Destination Ports. Our sniffer has a data structure called *OrderedSet*; which is a set that keeps the insertion order. These listed header fields are input for a hash

[1]D-ITG works mainly on packet-level but can emulate some applications
[2]Iperf homepage: https://iperf.fr/
[3]Ostinato homepage: https://ostinato.org/
[4]DPDK homepage: https://www.dpdk.org

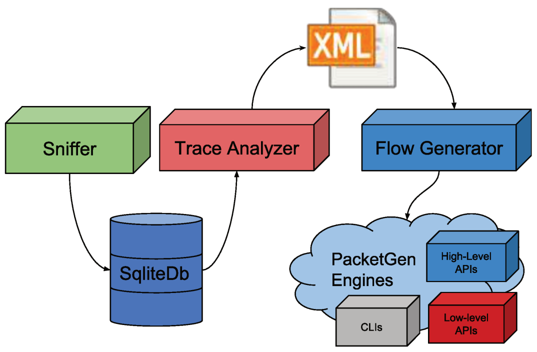


Fig. 1: Architecture of SIMITAR

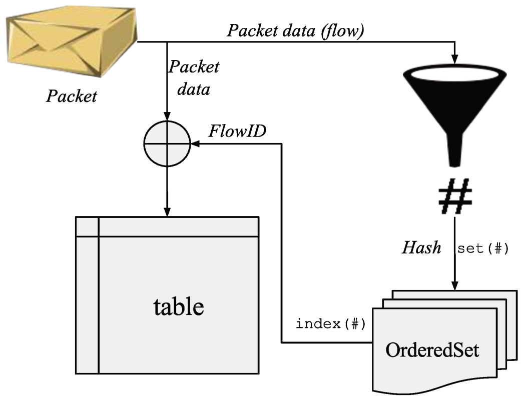


Fig. 2: SIMITAR's Sniffer hash-based flow classification

function of 64 bits from the family FNV. The hash result is inserted into the *OrderedSet*, which returns the hash position that acts as the packet flow ID. Finally, the Sniffer records the packet data and flow ID on the database for further analysis.

B. Trace Analyzer

This module is the heart of SIMITAR. It creates a trace model through the analysis of the collected data stored on the database. The Trace Analyzer saves this model in an XML file named Compact Trace Descriptor (CTD). Each traffic flow represents an entry on the CTD file. Each entry stores information about the flow, a model for inter-packet times and a model for packet-sizes.

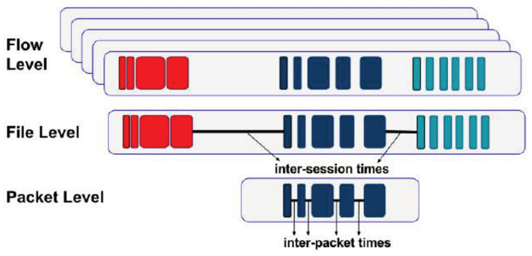


Fig. 3: Inter-packet time model

1) Flow features: Directly extracted data from flows, including flow duration, start delay, throughput, Protocols, IP and MAC addresses, ports and average TTL.

2) Inter-packet times: The inter-packet times model has been inspired by the Harpoons traffic model, but applies to any traffic, not just TCP, since it does not use header fields. The model represents the time between packets on three abstraction layers: (*i*) flow, (*ii*) file, and (*iii*) packet. The Trace Analyzer calculates the inter-packet times only in the context of a flow; in other words, the Trace Analyzer threats each flow as independent traffic. The intermediate layer is called file-level. For SIMITAR, a file is a sequence of consecutive packets transmitted with the inter-packet times smaller than a threshold `session_cut_time`. We use the same value of the *Request and Response timeout* from Swing: 30 seconds. The times within a file represents when the flow is active and called ON times. The intervals between files are called inter-session times and indicate that the flow is inactive (OFF times). In other words, this a packet-train (or ON/OFF) model for the flow traffic.

Inter-packet times are considered the times between packets inside files and are represented as random variables by stochastic processes. Currently, we are using eight different stochastic processes: Weibull parameterized with linear regression, Normal estimated with mean and standard deviation calculation, Exponential via average rate, Exponential via linear regression, Pareto through linear regression, Pareto through maximum likelihood, Cauchy using linear regression and Constant using mean calculation. For small flows (less than 30 packets), we estimate the constant model, since they have a limited impact on the final result. To rank these processes, from the best to the worst, per default, we use the Akaike Information Criterion (AIC). The usage is simple: the smaller, the better the process represents a dataset. The user may also choose to use the Bayesian Information Criteria (BIC). Further details on the effectiveness of these processes to (automatically) rank inter-packet times are described in [15].

3) Packet Sizes: For packet-sizes we adopted a bimodal, with two constant modes, with the small average packet sizes(smaller than 750 bytes), the average of the larger packets (greater than 750 bytes).

C. Flow Generator

The Flow Generator the component responsible for generating the traffic. It consumes the Compact Trace Descriptor files. It has an abstract C++ class called NetworkFlow that abstracts the layer of traffic generation. To replace the traffic generation API, the developer has to create a derivate class from NetworkFlow. Currently, we have two implementations: TinsFlow, which uses Libtins[5], a C++ API for packet generation; and IperfFlow, that uses the Iperf command line interface as API. To generate the traffic, the FlowGenerator instantiate a thread for each flow and make the sleep until the first ON time arrives. Then, the thread is weakened and starts to generate traffic through this time. To configure the traffic, it consumes the flow model created by the Trace Analyzer, described in the last session. When the ON time ends, the thread sleeps again for the next OFF time from the list. The procedure continues until the last ON time. Then, the thread ends.

IV. Experimental Validation

TABLE II: Description of the experimental setup.

Processor	Intel(R) Core(TM) i7-4770, 8 cores, CPU @ 3.40GHz
RAM	15.5 GB
HD	1000 GB
Linux	4.8.0-59-generic
Ubuntu	Ubuntu 16.10 (yakkety)
SIMITAR	v0.4.2 (Eulemur rubriventer)
Mininet	2.3.0d1
Iperf	iperf version 2.0.9 (1 June 2016) pthreads
Libtins	3.4-2
OpenDayLight	0.4.0-Beryllium
Octave	4.0.3
Pyshark	0.3.6.2
Wireshark	2.2.6+g32dac6a-2ubuntu0.16.10
Tcudump	4.9.0
libpcap	1.7.4

A. Methodology

As the experimental platform for validation we use Mininet-based emulated scenarios. For reproducibility purposes, Table II presents all relevant experimental details. All the required scripts are available on the SIMITAR code repository. We explore a tree topology (Fig. 4b, and a one-hop connection (Fig. 4a). Both scenarios as SDN networks with an OpenDayLight (Beryllium) controller. We use two pcap files. The first is a Skype capture (*skype-pcap*), and the second (*lgw10s-pcap*) corresponds to the first ten seconds of a gateway capture[6].

To compare the degree of realism of the generated traffic, we use the flows' cumulative distribution function (CDF) [1], and the Wavelet multi-resolution analysis [8]. On both analysis, the closer the plots are, the more realistic is the traffic generated by SIMITAR.

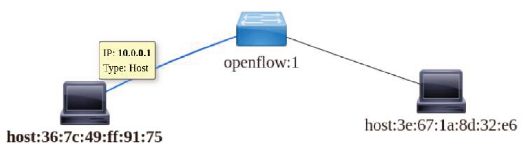
The flows cumulative distribution measures the ingress of new flows over time, and it is a measure similarity and

[5]Libtins homepage: http://libtins.github.io

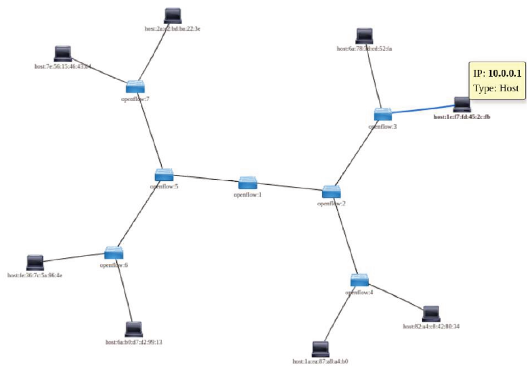
[6]*skype-pcap*: available at https://wiki.wireshark.org/SampleCaptures, named *SkypeIRC.cap*; lgw10s-pcap avaiable at http://tcpreplay.appneta.com/wiki/captures.html named *bigFlows.pcap*

TABLE III: Summary of results comparing the original traces (italic) and the traffic generated by SIMITAR.

	skype-pcap	skype, one-hop, iperf	skype, tree, iperf	skype, one-hop, libtins	*lgw10s-pcap*	lgw10s, one-hop, libtins
Hurst Exponent	0.601	0.618	0.598	0.691	0.723	0.738
Data bit rate (kbps)	7	19	19	12	7252	6790
Average packet rate (packets/s)	3	4	5	6	2483	2440
Average packet size (bytes)	260,89	549,05	481,14	224,68	365,00	347,85
Number of packets	1071	1428	1604	2127	24 k	24 k
Number of flows	167	350	325	162	3350	3264



(a) Single hop SDN topology



(b) Tree SDN topology

Fig. 4: Mininet SDN topologies. Controller OpenDayLight Beryllium.)

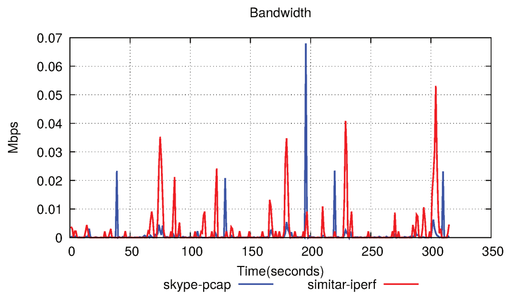


Fig. 5: Bandwidth plot (Iperf, single-hop, skype-pcap)

evolution of the traffic at the flow-level. On the other hand, the wavelet multi-resolution analysis can extract traffic scaling characteristics and is a measure of similarity at the packet-level. If the curve decreases, this indicates a periodicity on that time scale exists. If the curve remains approximately constant, it indicates similarity to white-noise. Finally, if the traffic has self-similar characteristics around a particular time scale, its curve increases linearly. Table III presents a compendium of metrics extracted from the traffic, including the Hurst exponent, which is a metric of the traffic fractal level. Self-similar processes, such as the network traffic, have its value between 0.5 and 1.0 ($0.5 < H < 1.0$) [13].

B. Results

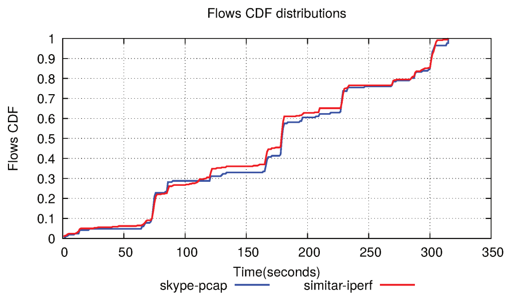
Figures 6, 7 and Table III show the obtained results when comparing the original and the synthetic traffic generated by SIMITAR. We also illustrate the bandwidth traffic for one of the use cases in Figure 5. As we can see the generated traffics are not identical regarding bandwidth. However, both present similar fractal-like shape. The Hurst exponent of inter-packet times in every case has an error smaller than 10% compared to the original in every case, i.e., the fractal-level of each synthetic traffic is indeed similar to the original trace.

Even though the generated traffic is not identical to the original, the cumulative flow distribution obtained for every study-case is almost identical on every plot. The small differences on the curves result from threads and process concurrence for resources, in addition to noise from the sleep/wake processes on the thread signals. Since the operating system made the packet capture and timing, the packet capture buffer queue may have contributed as well. This result was our most significant achievement in our implementation. This result shows that our method of flow scheduling and independent traffic generation was effective and efficient in replicating the original traffic at the flow-level. However, the actual number of flows was more significant when SIMITAR used Iperf as the traffic generator API and slightly smaller when using libtins. This discrepancy can be explained because Iperf establishes additional connections to control the signaling and traffic statistics for every connection. On the other hand, with libtins, the number of flows is small, since a flow generation is aborted if the NetworkFlow flow class fails to create a new traffic flow.

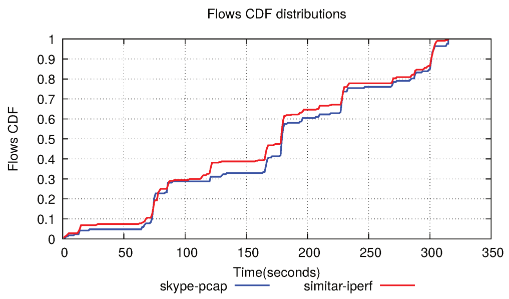
The results from the Wavelet multi-resolution analysis of inter-packet times vary in each case. The time resolution chosen was ten milliseconds, and it is represented in log 2 scale. The time of each time-scale j can be calculated by the equation:

$$t = \frac{2^j}{100}[s] \qquad (1)$$

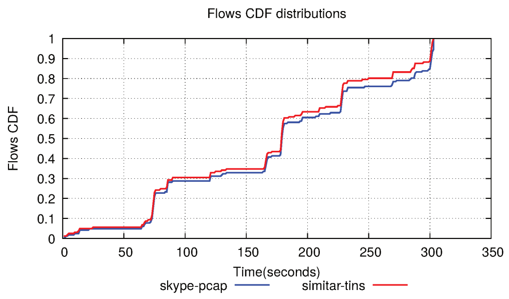
In the first case (figure 6a), SIMITAR reproduced Skype traffic, using Iperf in a single-hop scenario. On small time



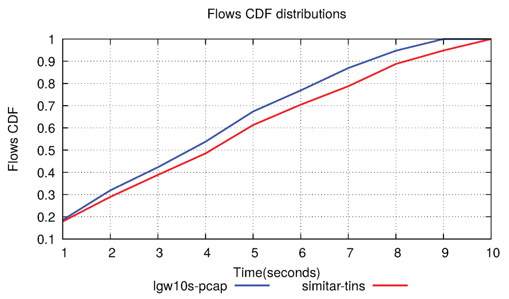
(a) *Iperf, single-hop, skype-pcap*



(b) *Iperf, tree, skype-pcap*

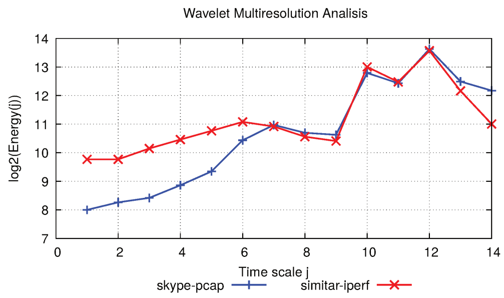


(c) *libtins, single-hop, skype-pcap*

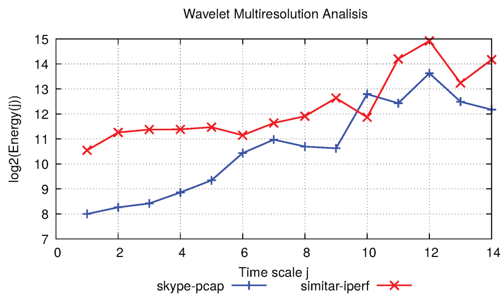


(d) *libtins, single-hop, langw10s-pcap*

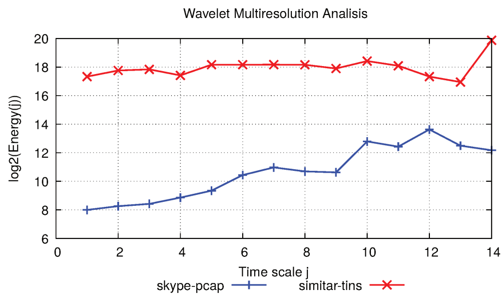
Fig. 6: Flows cumulative distributions.



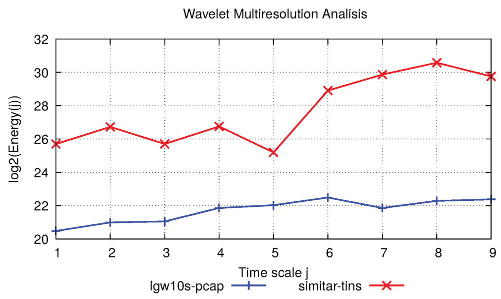
(a) *Iperf, single-hop, skype-pcap*



(b) *Iperf, tree, skype-pcap*



(c) *libtins, single-hop, skype-pcap*



(d) *libtins, single-hop, langw10s-pcap*

Fig. 7: Wavelet multiresolution energy analysis.

scales, both curves increased linearly, which indicates a fractal shape. However, at this point, they exhibit different slopes with the synthetic traces behaving closer to a white-noise shape. After the time scales 5 and 6 (300-600 milliseconds) scale, the error between the curves becomes almost negligible. We also observe a periodicity pattern at the time-scale of 9 seconds. Vishwanath and Vahdat [8] measured the same periodicity pattern; which appears to be an intrinsic characteristic of TCP traffic. We observe some periodicity at 11 and 13 time-scales (20 and 80 seconds).

In the second case (Fig. 6b), on a tree topology on small time scales, we identify behavior closer to white-noise on small scales, and similar results, but with more substantial energy levels on greater time scales. The diversity introduced by the topology and the concurrent signaling traffic caused by the other hosts and switches do explain the observed behavior since node signaling tends to be more randomized than user-generated traffic. Indeed, as we can see in Table II, there are two hundred more packets captured on the client interface in the tree topology compared to the one-hop scenario.

In the last two plots (Figures 6c and 6d), where we use libtins as the packet crafter, the energy level is higher, and the curves are less correlated. SIMITAR, in the current implementation, is not modeling inter-packet with libtins and sends packets as fast as possible, which explains this discrepancy. However, in the last scenario, due to the higher average throughput, the observed performance was better.

V. Future Work

There is a large room for new features and improvements to the SIMITAR project, which we are using on a number of active research projects. In the current model of traffic generation, we use a set of constants. Calibrating these constants may improve the realism and quality of the synthetic traffic outcomes. Also, a smarter flow scheduler that instantiates flow threads on demand would likely reduce overheads, yielding more realism under higher throughput rates. Our implementation for libtins was minimalistic and did not use many of the features we model. With a more complete implementation, we should achieve better results. Another extension we plan to develop is a metering component, to measure QoS features counting delivered and received packets, and thereby perform passive bandwidths measurements. Finally, we plan to leverage the extensibility of our tool to generate traffic of additional scenarios such as WANs (using DPDK), ZigBee, and WiFi [16].

VI. Conclusions

We present SIMITAR as a tool and methodology to attend the evolving needs of rich and realistic network traffic experiments working at both flow- and packet-level. At the flow-level, our methodology already achieves high fidelity results. The cumulative distribution of flows is almost identical in each case. From the perspective of benchmarking of a middle-boxes or SDN switches, this is a valuable result, since their performance, especially in SW implementations, largely depend on the number and characteristics of the stimuli flows. However, because of packets exchanged by background signaling connections, the traffic generated by Iperf, even following the same cumulative flow distribution, ended up creating more streams then expected. At the packet level, the current results with Iperf replicate with high accuracy the scaling characteristics of the first traffic, and the number of generated packets are not far than the expected. Despite all identified optimizations, the results are more than satisfactory and prove the potential of the proposed methodology. At the flow-level, our results are at least as good as those achieved by best-of-breed related work like Harpoon and Swing.

References

[1] J. Sommers and P. Barford, "Self-configuring network traffic generation," in *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '04. New York, NY, USA: ACM, 2004, pp. 68–81. [Online]. Available: http://doi.acm.org/10.1145/1028788.1028798

[2] Y. Cai, Y. Liu, W. Gong, and T. Wolf, "Impact of arrival burstiness on queue length: An infinitesimal perturbation analysis," in *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*, Dec 2009, pp. 7068–7073.

[3] S. Srivastava, S. Anmulwar, A. M. Sapkal, T. Batra, A. K. Gupta, and V. Kumar, "Comparative study of various traffic generator tools," in *Engineering and Computational Sciences (RAECS), 2014 Recent Advances in*, March 2014, pp. 1–6.

[4] A. Botta, A. Dainotti, and A. Pescap, "A tool for the generation of realistic network workload for emerging networking scenarios," *Computer Networks*, vol. 56, no. 15, pp. 3531 – 3547, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128612000928

[5] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network function virtualization: Challenges and opportunities for innovations," *Communications Magazine, IEEE*, vol. 53, no. 2, pp. 90–97, Feb 2015.

[6] E. Soltanmohammadi, K. Ghavami, and M. Naraghi-Pour, "A survey of traffic issues in machine-to-machine communications over lte," *IEEE Internet of Things Journal*, vol. 3, no. 6, pp. 865–884, Dec 2016.

[7] "Projeto mestrado," https://github.com/AndersonPaschoalon/ProjetoMestrado, 1234, [Online; accessed May 30th, 2017].

[8] K. V. Vishwanath and A. Vahdat, "Swing: Realistic and responsive network traffic generation," *IEEE/ACM Transactions on Networking*, vol. 17, no. 3, pp. 712–725, June 2009.

[9] G. Bartlett and J. Mirkovic, "Expressing different traffic models using the legotg framework," in *2015 IEEE 35th International Conference on Distributed Computing Systems Workshops*, June 2015, pp. 56–63.

[10] N. L. Antoine Varet, "Realistic network traffic profile generation: Theory and practice," *Computer and Information Science*, vol. 7, no. 2, 2014.

[11] W.-c. Feng, A. Goel, A. Bezzaz, W.-c. Feng, and J. Walpole, "Tcpivo: A high-performance packet replay engine," in *Proceedings of the ACM SIGCOMM Workshop on Models, Methods and Tools for Reproducible Network Research*, ser. MoMeTools '03. New York, NY, USA: ACM, 2003, pp. 57–64. [Online]. Available: http://doi.acm.org/10.1145/944773.944783

[12] A. Botta, A. Dainotti, and A. Pescape, "Do you trust your software-based traffic generator?" *IEEE Communications Magazine*, vol. 48, no. 9, pp. 158–165, Sept 2010.

[13] W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson, "On the self-similar nature of ethernet traffic (extended version)," *IEEE/ACM Transactions on Networking*, vol. 2, no. 1, pp. 1–15, Feb 1994.

[14] D. Kreutz, F. Ramos, P. Esteves Verissimo, C. Esteve Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, Jan 2015.

[15] A. d. S. Paschoalon and C. E. Rothenberg, "Automated selection of inter-packet time models through information criteria," *IEEE Networking Letters*, pp. 1–1, 2019.

[16] R. d. R. Fontes and C. E. Rothenberg, "Mininet-wifi: A platform for hybrid physical-virtual software-defined wireless networking research," in *Proceedings of the 2016 ACM SIGCOMM Conference*, ser. SIGCOMM '16. New York, NY, USA: ACM, 2016, pp. 607–608. [Online]. Available: http://doi.acm.org/10.1145/2934872.2959070