BB-Gen: A Packet Crafter for Data Plane Evaluation

**Fabricio E Rodriguez Cesen, P Gyanesh Kumar Patra,
Christian Esteve Rothenberg**

[1]Department of Computer Engineering and Industrial Automation (DCA)
Faculty of Electrical Engineering and Computing (FEEC)
University of Campinas (UNICAMP)
P.O. Box 6101, 13083-970, Campinas, SP, Brazil

{frodri,gyanesh,chesteve}@dca.fee.unicamp.br

Abstract. *With the advent of research on fast path packet processing, traffic generator tools witnessed many entrants with features ranging from supporting list of protocols, analyzing network traffic to measuring throughput and latency of packets. While approaching towards feature completeness, the tools are becoming more complex every time making it difficult to port, manage, and use. BB-Gen with a sole focus on simplicity complements other traffic generators instead of trying to replace them. BB-Gen is a simple CLI-based packet crafter to generate packet flows formatted as PCAP files. The tool supports different standard protocols and creates the necessary traces for network function configuration and testing. It allows creating PCAPs for worst and best case scenarios with all unique flows or following flow distributions published elsewhere. In this demo, we feature BB-Gen as used by the MACSAD development team to test P4-based software switch pipelines.*

1. Introduction

With increasing number of services over the Internet like email, web, video streaming etc., the demand for bandwidth is increasing exponentially. Along with it, the necessity to evaluate and test network capabilities become prevalent. While networks are becoming more (re-)configurable, network testing tools are becoming equally complex adapting to the need of the hour. The network testing and benchmarking tools depend on network workload generation to simulate the network traffic for testing purposes. This trivial task has been the foundation for several research activities like [Botta et al. 2012] focusing towards performance, scalability, and reliability of networks and network devices.

Traffic generator tools are an essential part of network testing with features ranging from supporting list of protocols, analyzing network traffic or measuring throughput to calculating latency of packets. In their strive to achieve feature completeness, the tools are getting more complex each time, and making it hard to port, manage, and use. To address this, we propose BB-Gen which is a python based tool, with a primary focus on simplicity, excelling in the creation of network packet traces.

The remainder of the article is organized as follows: motivations and goals for the development of our tool are introduced in Section 2. Section 3 summarizes related works. Section 4 gives an overview of the BB-Gen tool itself followed by a detailed explanation of its main features. Meanwhile, Section 5 presents an use case depicting the usage of BB-Gen for performance tests. Section 6 presents the documentation and

the demonstration details for the Demo section at SBRC 2018. Finally, future works and conclusion are discussed in Section 7 and Section 8 respectively.

2. Motivation and Goals

The generation of packet traces is commonly required to carry out performance evaluations and hence becomes a relevant task for the development of new network solutions and the evaluation of the existing.

Our analysis found a gap among open source PCAP trace generators with simple interfaces and covering the relevant requirements for rich performance experiments. Some alternatives support a complete set of protocols and allowed to generate different set of PCAP traces but do not provide important traffic flow characteristics (e.g., packet size and address distribution). Other tools support different encapsulation protocols but not both VXLAN and GRE in the same tool. Although there are many tools for packet generation, the complementary creation of table traces remains limited. Table traces are necessary in a programmable network to fill the table flow configuration of the network device.

Our needs on a single packet generator tool to meet the packet trace requirements of our research use cases to evaluate the performance and scalability of programmable data planes [Patra et al. 2016, Patra et al. 2017] and the identified limitations of existing tools motivated our work to develop a packet generator prioritizing two essential characteristics: (*i*) simple to use, (*ii*) wide protocol support and rich customizability. To that end, as our baseline, we opted for the well known Scapy python library [Biondi 2008], which is easy to extend and manipulate.

3. Related Work

Here we summarize a selected set of related PCAP trace generator solutions. Each tool has their advantages and disadvantages depending on their architecture, included features or supported platforms.

RWS [Knutsson 2014] PCAP generator is based on a simple packet descriptor language. The user defines the header fields for the packets required and feed it to **RWS** to generate the PCAP. It can also generate invalid packets which is rather uncommon among PCAP generators. An example of an invalid packet can be a TCP packet tunneled inside a Teredo tunnel and sent over GTP-u.

Ostinato [Ostinato 2010] is one of the most powerful packet crafter, network traffic generator and analyzer with complete GUI support. It implements most of the common standard protocols to facilitate traffic generation and analysis. With a complex user interface and numerous feature combinations, Ostinato presents a steep learning curve to tackle with, which makes it difficult and time-consuming for users to understand and take advantage of the tool.

Scapy[1] is a packet manipulation program with Python interpreter disguised as a Domain Specific Language (DSL). It can create and decode packets of an extended number of protocols. It can send and capture network traffic too. Its extended features also

[1]https://github.com/secdev/scapy/

include some basic network tasks (e.g., scanning, trace routing, probing, arpspoof, arp-sk, arping, tcpdump, tethereal, etc.). Scapy can stand out among competitors with its unique ability to arrange protocol headers in a custom sequence which may not confirm to any protocol logic. This feature allows Scapy to create invalid frames by combine techniques (e.g., VOIP decoding on WEP encrypted channel, etc.), similar to RWS. Among other features, Scapy allows to set values for all header fields, payload, and padding. Moreover, it allows to write a list of packets to a PCAP file.

4. BB-Gen

BB-Gen is a simple CLI based packet crafter written in Python over Scapy library. It can natively craft packets for different standard and custom protocols. It aims to create PCAP files to be used with a wide set of Traffic Generators (e.g., pktgen-dpdk [Olsson 2005], NFPA [Csikor et al. 2015a, Csikor et al. 2015b], TCPDUMP [Tcpdump 2010], etc.) helping network developers to validate the network and execute performance tests over the targets.

Though BB-gen is primarily used to create PCAP trace files, it differs itself by generating the table trace files for the PCAPs which are necessary to fill the table flow configuration of the target device for the network testing. Table traces contains the main information of the generated packets (e.g. source/destination IP/MAC address). BB-Gen allows to create traces files with same/random IP/MAC/L4Port details showing its control over the header fields like source and destination MAC addresses, IP addresses, TCP or UDP ports while creating packets. It also allows the user to create a complete set of PCAPs for performance test by specifying a single flag in the command line. Under this performance setting, PCAPs generated comprises of all the standard packet sizes (64, 128, 256, 512, 1024, 1280, 1518) [Bradner and McQuaid 1999] and also features simple (best-case) and complex (worst-case) scenarios by use of same/random distribution sets of header fields respectively. A single command to generate both PCAP and table trace files, and the command line arguments which counts to only a few and self explanatory agree to the ease of use of BB-Gen. A custom protocol support to BB-Gen can be easily added by first adding the support to Scapy similar to the Contrib[2] and then extending BB-Gen protocol list with minimal code changes.

4.1. Architecture

Figure 1 shows the principal components of the architecture of BB-Gen Packet Crafter.

- ***User:*** The user introduces required parameters such as distribution, protocols, numbers of entries, use case[3], packet sizes, etc., necessary to create the traces files.[4]
- ***Core:*** Being the principal part of BB-Gen, it receives and process information from the *User*, and generates the packet details to be included in the traces files. It comprises of three sub-modules i.e., Parser, Data Generator, and Packet Generator as explained below.

[2]https://github.com/secdev/scapy/tree/master/scapy/contrib

[3]Supported use case: MACSAD

[4]More information on required parameters to generate the traces are described in our GitHub Wiki page, see section 6

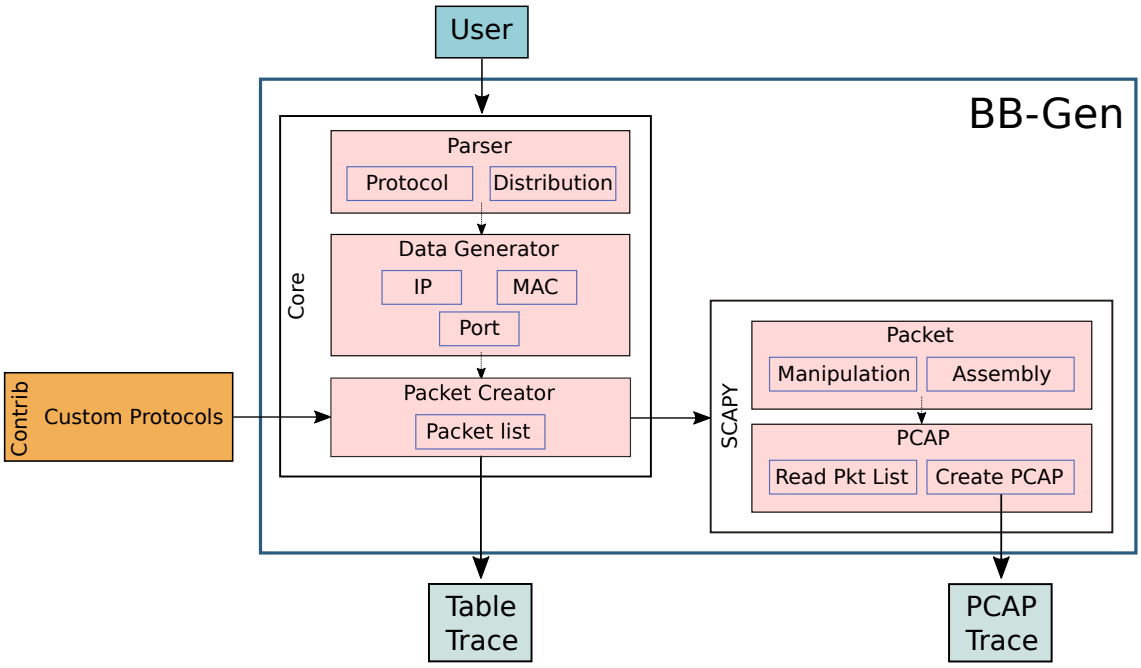

Figure 1. BB-Gen Architecture

  - ***Parser:*** is in charge of selecting the protocols to be used as well as the distribution, using the information introduced by the use or the defaults values in case of missing information.
  - ***Data Generator:*** using the protocols and the distribution details from *Parser*, it generates the list of source and destination IP, MAC and Ports.
  - ***Packet Creator:*** with the information set at *Parser* and the list of IP, MAC and Ports generated at *Data Generator*, the Packet Creator is going to create the list of packets with all the defined fields. With the list of packets prepared, the table trace file is going to be created using the informations about packet contents. And finally, the list of packets are sent to the Scapy block to generate the final PCAP trace.

- ***Scapy:*** is composed of the *Packet* and *PCAP* sub-blocks. The *Packet* is going to assemble the packets included in the list of packets with the correct protocol format. The *PCAP* will read the assembled packets and generate the PCAP file completing the BB-Gen process.

4.2. Main features

The principal features and capabilities of BB-Gen are summarized as below:

- Designed for simplicity, BB-Gen delivers an intuitive CLI based interface. By specifying only a few flags, the user can create a set of traces files.
- Very useful for best-case and worst-case testing. It allows to specify a simple/random distribution of header fields sufficient to address the most complex test cases.
- Being a python based tool, it is easy to build, use and extend to support additional protocols and new features.

- Easily create multiple PCAPs in a single step. The user can define the number of flows, packet sizes, etc. for each PCAP.
- Generates table trace files along with every set of PCAPs utilizing the informations from the PCAP files such as list of IP addresses, MAC address, Port numbers and also the packet encapsulation data for protocols like VXLAN and GRE. Trace file generation is seamless and does not require any additional user input.
- For scalability testing purposes, it can generate traces with more than 1 million unique packet details.
- Supports a list of common standard protocols:
  - Ethernet.
  - IPv4, IPv6.
  - TCP, UDP.
  - Protocol Encapsulations such as GRE and VXLAN.
- Useful for performance tests as it can automatically create packets of different sizes according to the RFC 2544 [Bradner and McQuaid 1999] (64, 128, 256, 512, 1024, 1280, 1518 Bytes) by setting a single performance flag in CLI.
- User defined custom packet sizes are also accepted at the CLI, just being limited by the defined minimum protocol size.
- Accepts user defined payload information. For this scenario, minimum packet size is maintained to be 64B by padding with random strings if necessary. In case payload saturates the 64B, the packet size is determined by the payload.
- The generated PCAP trace files are accepted as *inputs* for different network benchmarking and performance tools.
- Is a cross-platform tool with support for Windows, Linux, BSD and Mac OS X platforms.
- It is an open source project following BSD 3-Clause License.

5. Use Case

In order to demonstrate the usability of BB-Gen, we present a use case featuring a programmable dataplane (MACSAD) and a network performance evaluation tool (NFPA) which accepts a set of PCAP trace files as input for each 'determined setup'/'specified configuration' experiment run.

Multi-Architecture Compiler System for Abstract Dataplanes (MACSAD) [Patra et al. 2016, Patra et al. 2017] tries to converge Programming Protocol-Independent Packet Processors (P4) [Bosshart et al. 2014] and OpenDataPlane (ODP) [OpenDataPlane 2013] through a common compilation process delivering portability of dataplane applications without compromising target performance improvements while translating P4-defined dataplane abstractions into high-level ODP Application Programming Interfaces (APIs).

The Network Function Performance Analyzer (NFPA)[5] was proposed as a benchmarking tool that allows a user to measure important performance metrics of a network function compiled on any hardware and software combination, and to compare and store the results collected. NFPA is built over pktgen and Intel's Data Plane Development Kit (DPDK) [DPDK 2014] surpassing the kernel space limitations towards Network Interface Card (NIC) drivers. NFPA utilizes custom Lua scripts to parametrize, automate and

[5]https://github.com/cslev/nfpa

control the process of measuring the sending and receiving traffic. Moreover, it allows custom PCAP trace files to send traffic and configure network functions respectively.

5.1. Experimental Evaluation

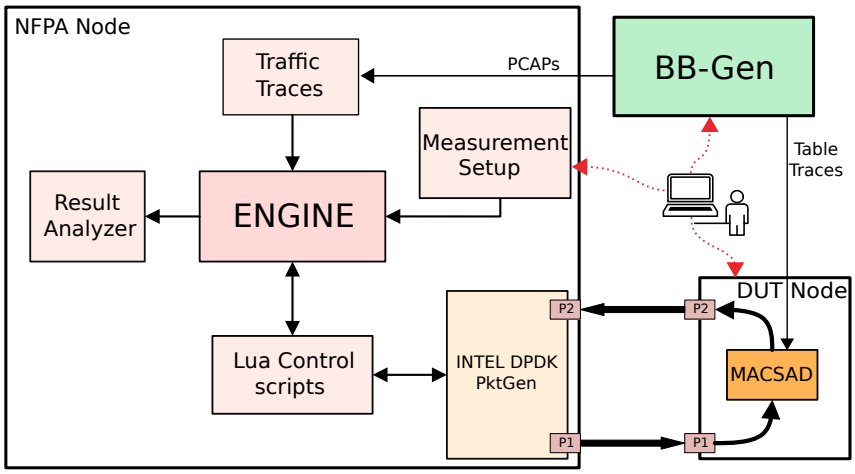

Figure 2. BB-Gen use case with NFPA and MACSAD

While using the NFPA benchmarking tool for performance evaluation of MACSAD, BB-Gen generates the necessary PCAP and table trace files for worst-case scenarios with random header field values (MAC and IP addresses, Port numbers, etc.). Multiple sets of PCAPs are created for different packet sizes according to the RFC 2544, and also with a different number of packet flows (100 to 1 million unique flows). This is repeated for each use-cases supported by MACSAD such as L2-Fwd, L3-Fwd with IPv4 and IPv6, GRE, and VXLAN.

For the use case evaluation, the NFPA standalone node is connected to the Device Under Test (DUT) (MACSAD) as presented in Figure 2. The user specifies the parameters for traces files at BB-Gen and the generated PCAP and table trace files act as input to the NFPA and MACSAD respectively. Then the user configures NFPA with measurement parameters (e.g., Hardware details, version, and path to software components, number of runs to be repeated, duration of each run, etc.), and similarly configures MACSAD with input parameters and the path to the table trace file at the DUT Node. NFPA sends the network traffic using the PCAP files generated by the BB-Gen on port 0, which in turn are processed and forwarded by DUT according to the table entry details which are updated by the table trace files from BB-Gen. And finally, NFPA receives the network traffic back at port 1. The throughput measurements are done for the DUT in terms of packets per second (pps) and bytes per second (bps). At the end of each run, the results are exported and stored in a local database by NFPA.

6. Documentation, Code, and Demonstration

BB-Gen is released as an open source project under BSD 3-Clause license. It can be downloaded from Github at `https://github.com/intrig-unicamp/BB-Gen`. The Github issue site `https://github.com/intrig-unicamp/BB-Gen/issues` is open to report bugs and initiate discussions about the current functionalities of the tool and to propose new features. Contributions to BB-Gen can be made by creating a fork of the project and initiating a pull request in Github.

The documentation is available at the site `https://github.com/intrig-unicamp/BB-Gen/wiki` which shows all the possible commands and flags used by the tool with their complete usage description. In addition, at this link `https://www.youtube.com/watch?v=amoGBOBdwVI` a complete video tutorial of the tool can be found explaining the usage of BB-Gen to generate the traces files with the help of a simple example.

The demonstration will focus on the generation of traces for performance purpose, and their use by different benchmarking tools (e.g., NFPA, pktgen-dpdk, tcpdump, etc.). Different configuration parameters and flags of BB-Gen will also be explained in details. The three important aspects of the demonstration are explained here.

Performance traces. This application adhering to the motto of 'simplicity' demonstrates how to create a full set of traces for different performance evaluations by setting the single *performance* flag at BB-Gen.

Distribution. The demonstration will show the process to generate different flow distribution in terms of the packets in PCAP. By specifying the distribution of MAC, IP and Port values, a full set of packets will be generated for different test scenarios.

Benchmarking tool integration. Finally, the demo will feature a use case scenario to showcase BB-Gen integration with other benchmarking tools. The test will present NFPA integration with BB-Gen, and how NFPA can read and send the network traffic using PCAPs generated by BB-Gen.

7. Future Work

In the near future we are planning to add more protocols (e.g., ICMP, ICMPv6, ARP, VLAN, etc.) while maintaining our fundamental characteristic 'Simplicity'. By leveraging the unique feature of Scapy, we plan to add support of the capability to create packets with a custom sequence of protocols headers. It will enable the user to create packets according to their necessities, even if the packet does not conform any logical sequence of protocols as per standards. That say we will have a feature supporting creation of malformed packets defined by the user. The inclusions of more use cases and the contribution to the NFPA project is also one of the main scopes. Another niche feature to add is the capability to generate a hexdump of the created packet. This will allow the user to verify if the generated packets are as expected or not.

In the longer term, we plan to transform BB-Gen into a modular platform creating different knobs for tool's features, supported protocols etc, which can accept new contribution seamlessly without the need to rewrite codes across the project. Also, all the features will be segregated into different classes and libraries for streamlining the new code development process.

8. Conclusions

BB-Gen is a python based tool, focusing on simplicity, written using Scapy library. It can be a great candidate for generating trace files for performance and evaluation tests. With a simple set of flags, it can create a set of traces from the simplest to the most complex performance test. The integration with a wide set of benchmarking tools reinforces the ease of use of the tool and benefits for the community with the evaluations. MACSAD

project results demonstrate the facility of BB-Gen to create traces covering different complexities and requirements of the project.

Acknowledgments

This work was supported by the Innovation Center, Ericsson Telecomunicações S.A., Brazil under grant agreement UNI.61.

References

[Biondi 2008] Biondi, P. (2008). Welcome to scapy's documentation!

[Bosshart et al. 2014] Bosshart, P., Daly, D., Gibb, G., Izzard, M., McKeown, N., Rexford, J., Schlesinger, C., Talayco, D., Vahdat, A., Varghese, G., and Walker, D. (2014). P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*.

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

[Bradner and McQuaid 1999] Bradner, S. and McQuaid, J. (1999). Benchmarking methodology for network interconnect devices. RFC 2544, RFC Editor. `http://www.rfc-editor.org/rfc/rfc2544.txt`.

[Csikor et al. 2015a] Csikor, L., Szalay, M., Sonkoly, B., and Toka, L. (2015a). Network Function Performance Analyzer. `http://nfpa.tmit.bme.hu`.

[Csikor et al. 2015b] Csikor, L., Szalay, M., Sonkoly, B., and Toka, L. (2015b). Nfpa: Network function performance analyzer. *IEEE Conference on Network Function Virtualization and Software Defined Networks Demo Track*.

[DPDK 2014] DPDK (2014). Data Plane Development Kit. `http://dpdk.org/`.

[Knutsson 2014] Knutsson, K. (2014). RWS Synthetic Pcap Generator. `https://github.com/karknu/rws`.

[Olsson 2005] Olsson, R. (2005). Pktgen the linux packet generator. In *Proceedings of the Linux Symposium, Ottawa, Canada*, volume 2, pages 11–24.

[OpenDataPlane 2013] OpenDataPlane (2013). OpenDataPlane.org. `https://www.opendataplane.org`.

[Ostinato 2010] Ostinato (2010). Ostinato. `https://ostinato.org`.

[Patra et al. 2016] Patra, P. G., Rothenberg, C. E., and Pongracz, G. (2016). Macsad: Multi-architecture compiler system for abstract dataplanes (aka partnering p4 with odp). *ACM SIGCOMM Demo and Poster Session*.

[Patra et al. 2017] Patra, P. G., Rothenberg, C. E., and Pongracz, G. (2017). Macsad: High performance dataplane applications on the move. *IEEE HPSR High Performance Switching and Routing*.

[Tcpdump 2010] Tcpdump (2010). Tcpdump. `https://www.tcpdump.org`.