SIGCOMM: U: High-Performance Flexible Packet Generator Using Programmable Switching ASIC

Zhaowei Xi, Mingwei Xu (Advisor)
Tsinghua University

1 PROBLEM AND MOTIVATION

Packet generator is widely used to generate traffic with customized properties (e.g., rate, packet type) and plays a vital role in network researches and network operations. Network researchers use packet generators to examine the performance of purposed prototypes [1]. For network operators, packet generators are required in latency measurement [2][3] and failure troubleshooting [4]. The development of current network gives rise to new demands on packet generators in two ways. Firstly, packet generators need to be high-performance to meet expanding network bandwidth (from 10Gbps to 100Gbps). Secondly, packet generators should be capable of customizing packets flexibly to satisfy constantly emerging network functions and protocols.

Existing packet generators can be categorized into hardware approaches and software approaches. Commodity packet generators based on proprietary hardware [5][6] can generate high rate traffic with certain pre-defined properties, yet typically they are not flexible enough for testing new protocols and new functions unless asking for extra customization from the product provider. Besides, the proprietary hardware can be expensive. For instance, a two-port (10GbE) packet test module can cost $25,000 [7]. Open-source packet generators based on the programmable chip like NetFPGA are reconfigurable [7][8] but of limited performance. Software approaches [9][10][11] based on general platforms are flexible enough for users to customize generation logic, but users have to consider the trade-off between performance and cost. For instance, an 8-core commodity server can hardly generate 100Gbps traffic [11][12]. Therefore, when operators need high rate traffic (e.g., pressure test of a large network), many servers are needed, and the cost increases rapidly.

Motivated by limitations of current approaches, we present HyperGen, a high-performance, flexible packet generator with reasonable cost using programmable switching ASIC [13][14][15]. HyperGen can produce above 1Tbps accurate traffic and allows users to customize packets flexibly by reconfiguring the packet processing logic, which makes HyperGen competent for plenty of tasks, such as throughput testing, latency and loss measurement, and denial-of-service attack emulation. With our further development, HyperGen now serves as an essential part of a mature network tester [16] and is capable of many complicated testing tasks.

However, It is not trivial to design programmable switching ASIC as a packet generator due to its limitations on programmability and resources. We deal with the challenge from two perspectives. Firstly, *we co-design switch CPU and switching ASIC by proposing template-based packet generation* that leverages switch CPU to generate template packets to enhance the flexibility of packet generation. Secondly, *we propose a new pipeline design in switching ASIC* for high-performance packet generation. The pipeline conducts acceleration, replication, and edition on template packets to generate high-performance testing traffic for various tasks.

2 BACKGROUND AND RELATED WORK

2.1 Background

Programmable switching ASIC. Based on reconfigurable match-action table (RMT) model [17], the reconfigurable components in programmable switching ASIC can be loosely divided into two parts, the parser and the pipeline. Parser decodes and encodes packets as the user-defined packet header formats. Pipeline contains multiple stages to implement RMTs that define the packet processing logic. Users can use network processing language like P4 [13] to access the switching ASIC and modify the packet processing logic.

Switching ASIC is designed as high-performance packet forwarding data plane, accounting for switching ASIC prohibiting operations that cannot guarantee the performance (e.g., nonlinear operations such as for/while). Therefore, switching ASIC, though titled programmable, has limited programmability. Meanwhile, the resources of ASIC are not without ground, especially RAM for data plane storage.

Programmable switch CPU. Programmable switches such as Tofino [15] also have a switch CPU connecting to switching ASIC by PCIe. Switch CPU serves as the control plane and is responsible for managing table rules of the data plane. Switch CPU can also acquire data (e.g., counter value, packet digest) from switching ASIC by control programs.

2.2 Related Work

In this section, we talk about three types of related works: commodity packet generators based on proprietary hardware, open-source packet generators based on NetFPGA, and software packet generators based on the general server. The comparison of related works is summarized in table 1.

Table 1: Comparison of current approaches [1].

Metrics (1Tbps)	Device(s) Required	Flexibility	Equipment Cost
Proprietary hardware	1	limited	≈ $100,000
NetFPGA	≈ 8	high, with heavy workload designing circuit	≈ $50,000
Software	≈10	high, with little workload using API and high-level programming language	≈ $30,000
HyperGen	**1**	**high, with moderate workload using NTAPI [16]**	**≈ $3,600**

Commodity packet generator. Commodity packet generators based on proprietary hardware [5][6] can provide high rate customized traffic with high precision. Commodity packet generators are user-friendly and support rich network functions. However, proprietary hardware typically costs a lot [7]. Meanwhile, though supporting many network functions and protocols, commodity packet generators can hardly be extended to support new functions and protocols. HyperGen, compared with commodity packet generators, has improvements on flexibility as switching ASIC can be reconfigured to customize new generation logic for users.

Open-source packet generator. Based on programmable hardware like NetFPGA [18], open-source packet generators are reconfigurable, thus providing flexible customization. Unfortunately, NetFPGA achieves limited throughput (e.g., a NetFPGA board equipped with four 10Gbps ports costs approximately $7000 [19]). HyperGen outperforms open-source packet generators for generating above 1Tbps traffic in a single programmable switch whose cost is acceptable (e.g., a Barefoot Tofino [15] programmable switch with 32 100Gbps ports costs approximately $3600 [20]). Furthermore, designing FPGA is notoriously complex [21][22], while HyperGen allows users to control generation logic with network testing API as our recent work [16], accordingly easing the workload of reconfiguration.

Software packet generator. Software packet generators based on general servers make huge progress in recent years. Original software packet generators leverage socket programming (e.g., scapy [23]) or kernel functions (e.g., pktgen in Linux [9]). Consequently, original software generators have quite limited performance [12]. With the development of fast network I/O technologies like DPDK [24] and netmap [25], the throughput has raised over one order of magnitude [10][11]. However, throughput and accuracy of software generators are still constrained by capability of the CPU.

[1]We estimate the cost of NetFPGA using NetFPGA-SUME Virtex-7 FPGA Development Board (four 10Gbps port with one extra empty interface for 100Gbps, about $7,000), so number is 1000 / 140 ≈ 8, and cost is 1000 * 7000 / 140 = $50,000. We estimate cost of server with the fact MoonGen can produce less than 100Gbps traffic in 8-core server (about $3,000).

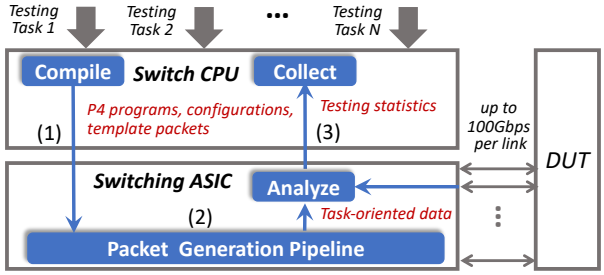


Figure 1: Architecture of HyperGen.

Therefore, higher demands on performance ask for more CPU cores, accounting for the undesirable trade-off between performance and cost. Software packet generators are in no doubt the most flexible packet generators, while HyperGen outperforms software packet generators with a high performance-to-price ratio.

3 APPROACH AND UNIQUENESS

3.1 Workflow

Figure 1 illustrates the architecture of HyperGen. The workflow of HyperGen can be summarized into three steps:

Compile tasks to generate template packets and packet processing logic of switching ASIC. Generation tasks are complied into switch configurations, template packets, and a P4 program controlling the packet processing logic of switching ASIC. Besides, in most tasks, packet generators are also responsible for analyzing generated packets or received packets from devices under test (DUT) to make the generation helpful for operators.

Forward template packets into packet generation pipeline to generate testing traffic. Template packets have not satified all demands of testing tasks (e.g., high rate). Therefore, after switch configurations are set and the P4 program is downloaded into switching ASIC, switch CPU forwards template packets into packet generation pipeline, and switching ASIC generates testing traffic from template packets.

Collect and analyze statistics from generated packets and received packets. HyperGen can record a particular per-packet field value, or statistical data such as packet count,

$T_1^t = trigger()$
$\quad .set([dip, dport, proto, flag, seq\_no], [X, 80, tcp, SYN, 1])$
$\quad .set(sip, range(Y, Z, 1))$
$\quad .set(sport, range(A, B, 1))$
$\quad .set([length, interval], [64, 0.1us])$
$Q_1^t = query(T_1^t).map(p->(pkt\_len)).reduce(func = sum)$
$Q_2^t = query().map(p->(pkt\_len)).reduce(func = sum)$

Table 2: Example of Dos emulation.

throughput, and delay. The aforementioned data can be attained by switch CPU via pulling from data plane counters or receiving packet digests pushed by switching ASIC. As a result, network operators get statistics they intend to acquire.

3.2 Template-based packet generation

It is hard for switching ASIC to customize properties like packet size or payload. Therefore, we decouple users demands as two types: demands that should be configured as template packets generated in switch CPU and demands that should be realized in switching ASIC. Our original design compiles users' demands by python and P4 programs. We further develop network testing API (NTAPI) [16] to represent testing intents using a similar programming model with the stream processing frameworks (e.g., Flink [26]). NTAPI [16] uses *network stream trigger* to describe generation tasks and *network stream query* to describe analysis tasks. NTAPI uses the *set* primitive to define packet header, payload, packet size, injection interval, injection port, and loop (i.e., how many times the template packet stream should be generated).

Table 2 provides an example of representing Dos emulation. The trigger T_1^t describes the generation task and corresponds to a template packet stream about the generation. First, HyperGen initializes the template TCP SYN packets with certain header fields and payload. Then, HyperGen generates the code and configurations for mcast engine and periodic timer (Figure 3) to realize the rate of 10Mpps and forward generated packets to the destination port. At last, HyperGen generates the code for modifying testing traffic. In this case, HyperGen sequentially sets the source IP address and source port as values in the range. The query describes the analysis tasks. Q_1^t records the number of generated packets, and Q_2^t records the number of received packets.

3.3 Packet generation pipeline

As Figure 2 illustrates, packet generation pipeline consists of three components that perform acceleration, replication, and edition sequentially on template packets. Besides, Figure 3 shows the component layout inside switching ASIC, and the meaning of different line types are identical to Figure 2.

Accelerator. By using accelerator, HyperGen accelerates template packets to 100GbE full line rate in negligible time.

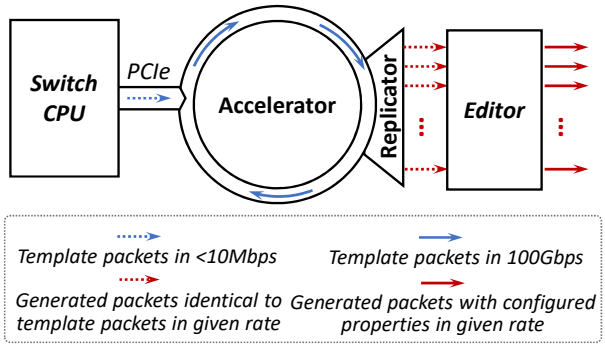


Figure 2: Design of Packet Generation Pipeline.

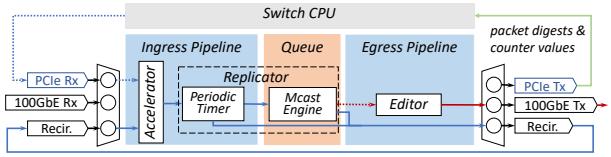


Figure 3: Component layout of packet generation pipeline in RMT.

Accelerator forwards template packets into a loop via *recirculation* (a general primitive supported by P4-programmable hardware). Switching ASIC realizes *recirculation* by setting the port as loop-back mode and forwarding packets back to ingress directly after leaving egress Tx queue and serialization. Our experiments testify that Tofino [15] can recirculate packets at a speed of no less than 100Gbps. Thus, the template packets never leave the switch and serve as a stable high rate packet source for the replicator through *recirculation*.

Replicator. Replicator executes conditional packet replication on the looping template packets. Replicator designs a nanosecond-level periodic timer to adjust rate via adjusting inter-departure time (i.e., the gap time between two packets). Once the timer expires, replicator conducts replication via *multicast*, a general primitive supported by many switches that replicates packets to multiple ports simultaneously. The timer threshold can be set the same way as header modification in editor, which means replicator manages to generate packets as constant rate, regularly changing rate or random distribution. Evaluations (§4.2) prove replicator achieves strong similarity and generality.

Editor. After replication, generated packets are identical to template packets. Editor conducts per-packet modification on packets to form testing traffic. Editor supports four types (constant value, given value list, arithmetic progression, random values according to a certain distribution) of header field modification on parsed header fields.

The first one is the constant value that can be set in template packets. The second one is the given value list. Editor implements a periodic counter to record packet ID and provide table matching from packet ID to given value list. For instance, we can assume users intend to set the TCP source port to 80, 81, 82 sequentially. If the packet ID is 2,

the editor sets the TCP source port to 81 and increases the packet ID to 3. The third one is the arithmetic progression (i.e., adding or subtracting over a certain value every time), similar to the implementation of the given value list. The fourth one is the random values according to a certain distribution. P4 only supports a uniform random generator (i.e., modify_field_rng_uniform). Editor implements the inverse transformation method [27] with two tables. Through the inverse transformation method, editor can generate values based on arbitrary distributions as long as the cumulative distribution function is provided.

3.4 Limitation

Although HyperGen reconciles the goal of high-performance and flexible with limitations on programability and resources through the aforementioned designs, it is responsible for us to claim the limitations of our designs.

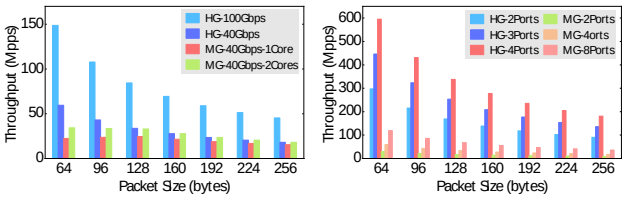
Limitation of template-based packet generation. Since we use the recirculation operation to accelerate template packets, there are conflicts if the task needs too many template packets and strict order. Our experiments prove that recirculation RTT of a 64-byte template packet is about 600ns, which means the maximum number of template packets that one loop-back port can accelerate sequentially is nearly 100. Therefore, HyperGen is not competent if the task can only be represented with a large number of template packets with high relevance (e.g., packet trace replaying according to captured traffic). Using more ports for loop-back can ease the problem of too many template packets if we can loosely assume irrelevant among template packets.

Limitation of programmable switching ASIC. Tofino only supports modifying the first 1500 bytes of packets. Therefore, HyperGen is not competent for tasks that need involved modification on payload of the generated traffic (e.g., IPSec testing that requires decryption or encryption over the payload). Besides, we use stateless connections [16] to avoid a huge amount of connection states, which requires connection state transitions in the task can be explicitly triggered by packets (e.g., SYN and SYN+ACK for TCP handshaking). Therefore, HyperGen cannot support tasks whose connection state transitions are not triggered by packets (e.g., emulate duplicated ACK behaviors). Users can modify HyperGen to support specific stateful tasks, but scalability may be a challenge due to the limited storage.

3.5 Uniqueness

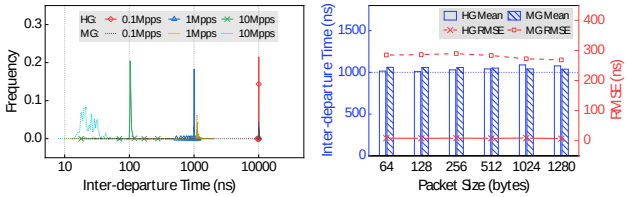
The novelty of HyperGen can be summarized as follows:

- Propose the idea of leveraging programmable switching ASIC to build a packet generator.
- Present template-based packet generation to co-design switch CPU and switching ASIC.
- Design packet generation pipeline in switching ASIC for high-performance packet generation with limited programmability and resources.



(a) Single-port rate (b) Multi-port rate

Figure 4: Packet generation rate in Mpps.



(a) Inter-departure time distribution (b) Mean and RMSE

Figure 5: Accuracy of constant rate generation.

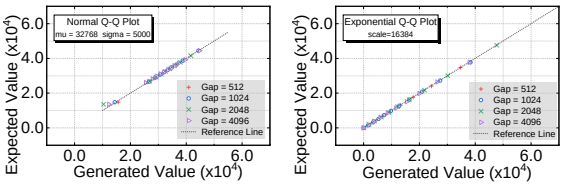
4 RESULT AND CONTRIBUTION

We compare HyperGen with a widely used DPDK-based packet generator, MoonGen [11]. We hope to call attention that MoonGen is a software-based approach, while HyperGen leverages the capability of programmable switching ASIC. In our defense, we do not have access to commercial hardware packet generators, and we have described the capability of hardware approaches in related work with no partiality. Besides, we have claimed the limitations on flexibility in detail in §3.4 to clarify what HyperGen cannot do.

4.1 Experiment setup

We implement a prototype of HyperGen (HG) on Wedge 100BF-32X equipped with Tofino and 32 100GbE ports. Taking the consideration of reserving ports for normal functions in a practical network, we use one port for recirculation and four ports for multicast in our testbed. Experiments prove users can use more ports for *recirculation* to support more template packets and use more ports for *multicast* to achieve a higher rate(i.e., over 1Tbps)[1]. We use a server with 64GB RAM and 8 2.10GHz CPU cores to run MoonGen (MG). We employ another programmable switch with precise timestamps to evaluate traffic generated by *HyperGen* and MoonGen to avoid involving errors of different timestamps.

[1]The capability of *recirculation* and *multicast* can be observed by implementing the packet counter at egress pipelines of the same switching ASIC.



(a) Normal distribution (b) Exponential distribution

Figure 6: Accuracy of random distribution generation.

4.2 Results

Packet generation throughput. Figure 4 shows the evaluations of throughput that HyperGen and MoonGen can achieve under different packet sizes and configurations. The evaluations using one single port and multiple ports demonstrate HyperGen can achieve full line-rate of 400Gbps in our testbed, which outperforms MoonGen with 8 cores.

Constant rate generation. We evaluate the quality of generated traffic by measuring the inter-departure time between two generated packets, which can be achieved by recording the ingress timestamps when the switch receives the generated packets in the idle network. Figure 5(a) illustrates the distribution of inter-departure time under different rate configurations. HyperGen preserves accurate towards the configured rates, while MoonGen performs well at 0.1Mpps, moderate at 1Mpps, inaccurate at 10Mpps. One possible reason is that DPDK-based packet generation has to generate a batch of packets at one time to reach the high rate, accounting for the mainly inter-departure time of MoonGen is between 20ns and 70ns. Meanwhile, MoonGen adjusts the gap between batches of packets to make the average inter-departure time 100ns for 10Mpps. Figure 5(b) shows the statistical metrics of different packet sizes at 1Mpps. We calculate the average inter-departure time as well as Root Mean Squared Error (RMSE). Both HyperGen and MoonGen have good accuracy at average inter-departure time. HyperGen has much lower RSME, which means the generated traffic can be more precise to constant rate.

Random distribution generation. We also test the capability of generating packets in random distribution. Figure 6 draws the Q-Q plot of normal distribution and exponential distribution. The results claim that HyperGen is capable of generating random numbers obeying a specific distribution with extreme similarity and generality.

4.3 Contribution

We present HyperGen, a high-performance, flexible packet generator with competitive cost using programmable switching ASIC. We propose template-based packet generation to decouple tasks and dispatch demands between switch CPU and switching ASIC to improve flexibility. We propose a new pipeline design inside switching ASIC to support high-performance packet generation, which is creative and can be enlightening for other pipeline-based packet processing hardware. Our experiments prove HyperGen supports line-rate packet generation with high quality. HyperGen now serves as a significant component in a mature network tester and is competent for many complicated testing tasks.

REFERENCES

[1] P.G. Kannan et al. Precise time-synchronization in the data-plane using programmable switching asics. In *SOSR*, page 8–20, 2019.

[2] C. Guo et al. Pingmesh: A large-scale system for data center network latency measurement and analysis. In *SIGCOMM*, pages 139–152, 2015.

[3] Y. Geng et al. Simon: A simple and scalable method for sensing, inference and measurement in data center networks. In *NSDI*, page 549–564, 2019.

[4] Y. Zhao et al. Pronto: Efficient test packet generation for dynamic network data planes. pages 13–22, 2017.

[5] IXIA. Test hardware. Website, 2020. https://ixia.keysight.com/products/test-hardware.

[6] SPIRENT. Test hardware. Website, 2020. https://https://www.spirent.com/products-for/high-speed-network-testing.

[7] Gianni Antichi et al. OSNT: Open Source Network Tester. *IEEE Network Magazine*, 2014.

[8] G. A. Covington et al. A packet generator on the netfpga platform. In *FCCM*, pages 235–238, 2009.

[9] pktgen. Website, 2020. https://www.kernel.org/doc/Documentation/networking/pktgen.txt.

[10] pktgen dpdk. Website, 2020. http://git.dpdk.org/apps/pktgen-dpdk.

[11] P. Emmerich and et al. Moongen: A scriptable high-speed packet generator. In *IMC*, pages 275–287, 2015.

[12] P. Emmerich et al. Mind the gap - a comparison of software packet generators. In *ANCS*, 2017.

[13] P. Bosshart et al. P4: Programming protocol-independent packet processors. *SIGCOMM CCR*, 44(3):87–95, July 2014.

[14] Barefoot Networks. The world's fastest & most programmable networks. Website. https://goo.gl/1mtjpf.

[15] Barefoot Networks. Barefoot tofino switch. Website. https://barefootnetworks.com/technology/.

[16] Y. Zhou et al. Hypertester: High-performance network testing driven by programmable switches. In *CoNEXT*, page 30–43, 2019.

[17] P. Bosshart et al. Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn. In *SIGCOMM*, 2013.

[18] N. Zilberman et al. Netfpga: Rapid prototyping of networking devices in open source. In *SIGCOMM*, page 363–364, 2015.

[19] Dijilent. Netfpga sume virtex-7 fpga development board. Website, 2020. https://store.digilentinc.com/netfpga-sume-virtex-7-fpga-development-board/.

[20] J. Sonchack et al. Turboflow: Information rich flow record generation on commodity switches. In *EuroSys*, 2018.

[21] H. Wang et al. P4fpga: A rapid prototyping framework for p4. In *SOSR*, page 122–135, 2017.

[22] B. Li et al. Clicknp: Highly flexible and high performance network processing with reconfigurable hardware. In *SIGCOMM*, 2016.

[23] Scapy. Website. https://scapy.net/.

[24] Intel. Data plane development kit. Website. https://www.dpdk.org.

[25] L. Rizzo. Netmap: A novel framework for fast packet i/o. In *ATC*, 2012.

[26] The Apache Software Foundation. Flink: Stateful computations over data streams. Website, 2020. https://flink.apache.org.

[27] wikipedia. Inverse transform sampling. Website, 2020. https://en.wikipedia.org/wiki/Inverse_transform_sampling.