Many-Core Processor based Pseudo-Stateful Traffic Multiplier

Zhitao Wan
School of Electronics Engineering and Computer Science
Peking University
Beijing, China
zhitao.wan@pku.edu.cn

Ping Wang
National Engineering Research Center for Software Engineering
Peking University
Beijing, China
pwang@pku.edu.cn

***Abstract* — Benchmark of high performance network devices relies on high throughput traffic generator. The traffic generators are software based or dedicated hardware based. The software based traffic generators are usually running on relatively cheap commercial computer in a flexible, low cost way but suffering from poor performance. The hardware based generators can provide more precisely measured heavy traffic load but for complex traffic generation the replay mode is usually adopted. The replay mode is stateless without real interactive procedures and status verification. In this paper, a many-core processor based Traffic Multiplier working with complex stateful traffic source(s) is presented to generate pseudo-stateful heavy traffic load with normal interactive procedures. The simulated traffic pattern can be changed flexibly by adopting corresponding traffic source(s) to fit the workload diversification for the device under test. The benchmark results of many-core processor based Traffic Multiplier proofed that it can multiply complex traffic with high fidelity.**

Keywords - Traffic generator, Traffic multipiler, Traffic Cloud, Many-core, Peseudo-stateful

I. INTRODUCTION

The more bandwidth for every telecommunication subscriber drives the core network evolution to 10 Gbps, 40 Gbps, 100 Gbps bandwidth and more. The performance measurement of the network devices needs high throughput traffic generators for different abstract layers. The traffic generators can be classified into four types: closed loop and multilevel, packet level, flow level and application level [1]. Application level test is the most complex and critical one for telecommunication networks. There are lots of application layer protocols with complex states and constraints such as Quality of Service (QoS), Quality of Experience (QoE) and so on. Usually the datasheet of telecommunication network equipment should indicate the concurrent session for give application/protocol and user number with real test data supporting. The reliable high throughput traffic generator is necessary for this purpose.

The MIPS architecture many-core processors are widely adopted in today's core networks. The capacity and power efficiency of them are competitive for packet processing. Moreover, most of them are System on Chip (SoC) design with integrated network interfaces and dedicated hardware for packet preprocessing. It is easy to implement high performance networking applications on them. This paper presents a many-core based Traffic Multiplier that can work with any traffic generator or network application to generate high throughput synthetic traffic.

II. BACKGROUND

Generally there are two types of traffic generators. One is software based which commonly means a traffic generation application runs on a general Operation System (OS) and general purpose hardware such as Personal Computer (PC), Laptop or powerful workstation. The other is hardware based which commonly adopts one or more dedicated hardware such as Field Programmable Gate Array (FPGA), Network Processor (NP) [2,3], Application Specific Integrated Circuit (ASIC) and their integration with General Purpose Processor (GPP).

A. Software Based Traffic Generator

The advantages of software based generators mainly include:"

- Easy deploy ability of multiple nodes to reproduce distributed scenarios.
- Ability to rapidly modify and extend the code for a specific research purpose, adding new features, statistical models, and support of new operating systems and hardware platforms.
- Possibility to perform more realistic experiments and test actual implementations by running on top of real operating systems and network protocol stacks" [1].

Moreover, they are much cheaper comparing with dedicated hardware. But software based generators cannot provide detailed datasheets containing certified information such as confidence intervals of the imposed values [1].

The software based traffic generators include TG, MGEN, RUDE/CRUDE, D-ITG, Brute, KUTE, PktGen, IXPKTHGEN, Harpoon, Surge and etc [1].

The software based traffic generators can simulate real interactive procedures and also replay captured traffic.

B. Dedicated Hardware Based Traffic Generator

The dedicated hardware based implementations are usually delivered by special design boxes. Some vendors provide common host boxes and different line cards to provide more flexible choices. This type of commercial traffic generator provides more precise statistical features,

more user friendly control interfaces and higher performance. But, even for this expensive hardware only for a pure application level protocol or simple hybrid protocols test is stateful. High throughput test are mainly replay or packets sending sequentially without real interactive between corresponding parts [4]. Usually a dedicated hardware based traffic generator only supports protocols in vendors' list and hard to extend by end users. And, the price of this type of traffic generator is relative high.

The dedicated hardware based traffic generator vendors include Agilent, IXIA, Spirent, Omnicor, Candelatech and etc [1].

III. PSEUDO-STATEFUL TRAFFIC MULTIPLIER

A traffic generator usually fit for some test scenarios, e.g., only for some abstract layers and/or for some given protocols. Figure 1 depicts a typical test bed with a traffic generator, traffic convergence hardware and a Device Under Test (DUT). A traffic generator usually constructs, stores packets and sends them as fast as it can. The packet construct is knowledge based, i.e., the traffic generator should conform its behavior to all or part of protocol specification. Each protocol supported by a traffic generator should also be verified and updated frequently to follow the latest version. The maintenance cost much effort especially for those application layer protocols.

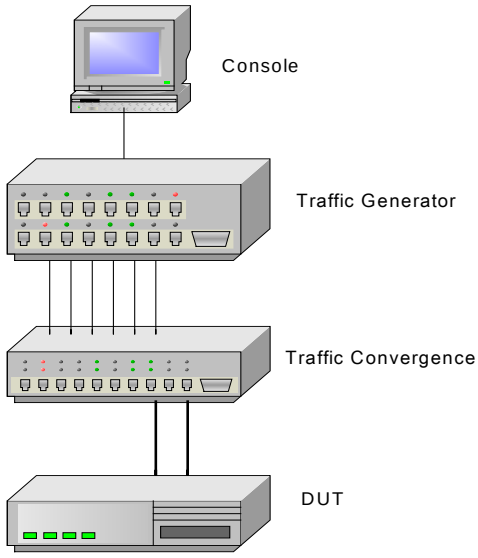

Figure 1. A typical test bed

A traffic generator should also keep the state machines and timers for some protocols. The overall throughput of a traffic generator is also limited by the complexity of the state machines. The state machines and timers consume much resource and block the system performance enhancement.

In those high throughput traffic test scenarios the synthetic traffic is generated without state machine support in most of current commercial of the shelf products. A traffic generator sends packets in high speed following predefined mode. For complex test scenarios such as multi service trace and replay is an overwhelming method. The software based traffic generator is flexibility and easy to be modified to fit for complex test scenarios but the fatal limitation is the poor performance. Dedicated hardware based traffic generator can reach very high throughput but cannot support complex application layer protocols in interactive mode.

To harmonize the advantage of software based and hardware based implementation, in this paper a new hardware based Traffic Multiplier is presented. The Traffic Multiplier intercepts the traffic generated by software based traffic generator or real applications and duplicates the traffic for times to produce high throughput traffic. The interactive procedures of all duplicated traffic follow the original traffic generator(s) and/or application(s). It just likes using a group of mirrors to reflect more copies of original packets to boost the output throughput.

Figure 2 depicts the interactive procedure that supports Client Server model applications. It includes the following steps:

- Client sends out an original request packet to Traffic Multiplier.
- Traffic Multiplier duplicates packets for arbitrary times and sends out duplicated packets with original request packet to DUT.
- DUT responses the requests and sends response packets to Traffic Multiplier.
- Traffic Multiplier only sends original request to the Server if the packet passes through the DUT successfully.
- Server responses the only original request as normal way and sends response packet to Traffic Multiplier.
- Traffic Multiplier duplicates packets for the same times as request packets and sends out duplicated packets with original response packet to DUT.
- DUT sends the packets back to Traffic Multiplier.
- Traffic Multiplier filters the original response and sends it to Client.

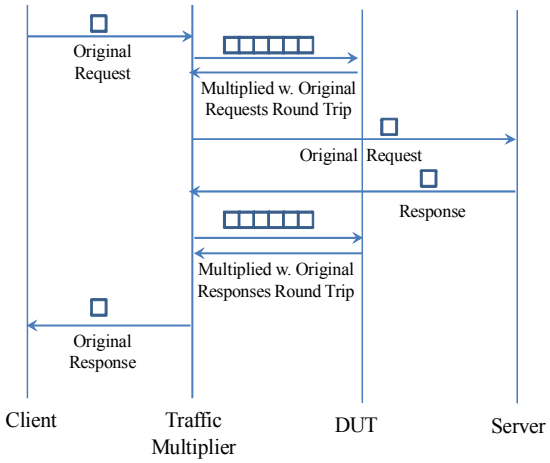

Figure 2. Interactive procedure for Client Server model

There are two ways to handle the original request and response. One is the Traffic Multiplier sends an original request or response to counterpart directly. The Traffic Multiplier just likes isolation transformer between alternating current power source and the powered device. The other way is the Traffic Multiplier tags and waits for a round trip request or response from DUT before sends the request or response back to the counterpart. Obviously, the former only pours traffic to the DUT. It is effective but not well observed. It is useful for some test scenarios that needs some background traffic. The later is more reasonable for most other test scenarios.

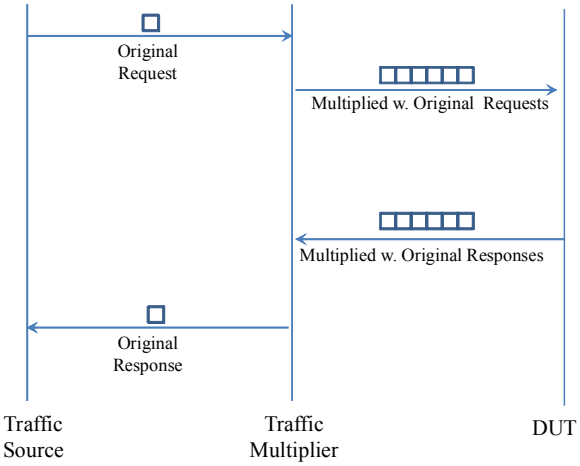

Figure 3. Interactive procedure for termination model

Figure 3 depicts the test scenario that DUT terminates connections. For example, the DUT is an HTTP server. The Traffic Multiplier does not check requests anymore and it only send the response for the original request to the traffic source rather than all responses.

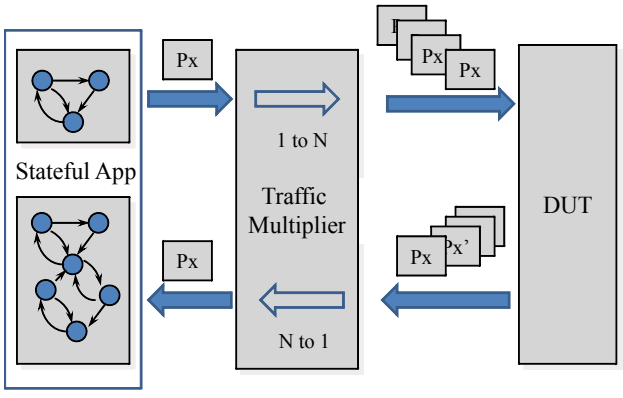

Figure 4. Loose coupling stateful Traffic Multiplier

Most of protocols are stateful. The counterparts keep state machines on each side and state transitions are according to the messages from counterparts. Figure 4 depicts the loose coupling stateful Traffic Multiplier. The Stateful App is outside the Traffic Multiplier. The main advantage of this configuration is any application can act as traffic source.

Figure 5 depicts tight coupling Traffic Multiplier the stateful application is integrated with Multiplier. Because the Traffic Multiplier is all in one box and it is simpler to mark out the original traffic. This configuration is easier to use and more efficient. The configuration of Traffic Multiplier can be easily changed and the functions of traffic source and multiplier are logically independent.

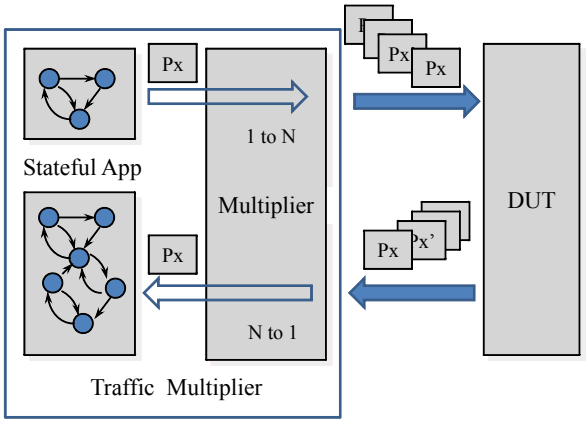

Figure 5. Tight coupling stateful Traffic Multiplier

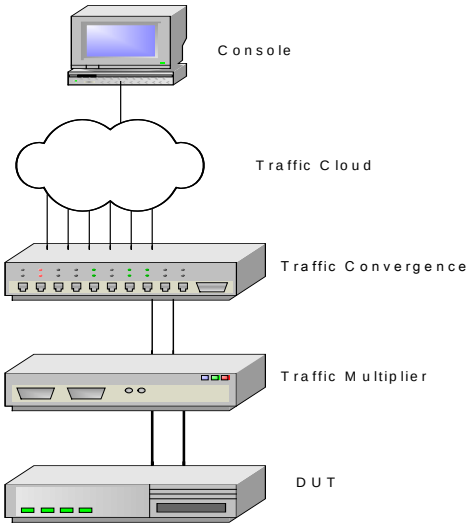

Figure 6. The test bed with Traffic Multiplier

Figure 6 depicts a test bed in loose coupling configuration adopts a variety of traffic sources named as Traffic Cloud and a Traffic Multiplier to perform high throughput test on the DUT. The Traffic Multiplier replaces the Traffic Generator in Figure 1. The Traffic Cloud including any possible traffic source entity is introduced as the seed for all traffic. The traffic convergence occurs before source traffic goes into the Traffic Multiplier. The traffic

convergence device is to merge relative complex traffic and classify traffic into two directions as the input of the Traffic Multiplier. It also performs part of traffic shaping and grouping on this device.

A test bed for many protocols is depicted in Figure 6. The Traffic Cloud enables all necessary protocols with relative lower session rate. The output of Traffic Multiplier includes traffic generated and original input traffic. For those stateful streams the state machines are kept in the Traffic Cloud. An original stream and all its synthetic streams are in a group. All streams share the same state machine(s) of the original stream. The original session failure means all the duplicated sessions failed as well. It means only part of the output traffic is real stateful, i.e., the output traffic is pseudo-stateful.

IV. IMPLEMENTATION

With the technology advance, many-core processors with dedicate packet preprocessing hardware and acceleration engines emerged in the market. They are also known as Network Services Processor (NSP) [5]. They hold the traditional NP's advantages and provides high flexibility and easy to program software environment includes Linux and Bare Metal. The OCTEON Plus CN3850 is one of such chips with twelve cnMIPS64 cores at up to 800 MHz speed on a single chip that integrate high-density, high-bandwidth standard interfaces along with other accelerators. The Evaluation Board (EVB) for this implementation has one XUAI interface and five 1G RJ45 interfaces with 4G DDR2 memory at 600MHz.

A. Typical Architecure Of MIPS Many-core Processors

Many-core processor usually refers more than eight interconnected processing cores on a single chip. MIPS(Microprocessor without Interlocked Pipeline Stages) is a RISC(reduced instruction set computer) instruction set architecture. The advantages of this type of chip are mainly including better power efficiency, higher computing capacity density and more flexibility.

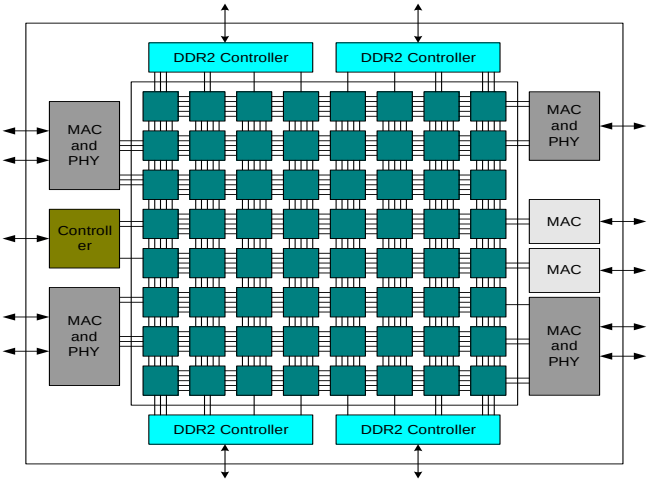

Figure 7. Diagram of Tilera Tilepro64

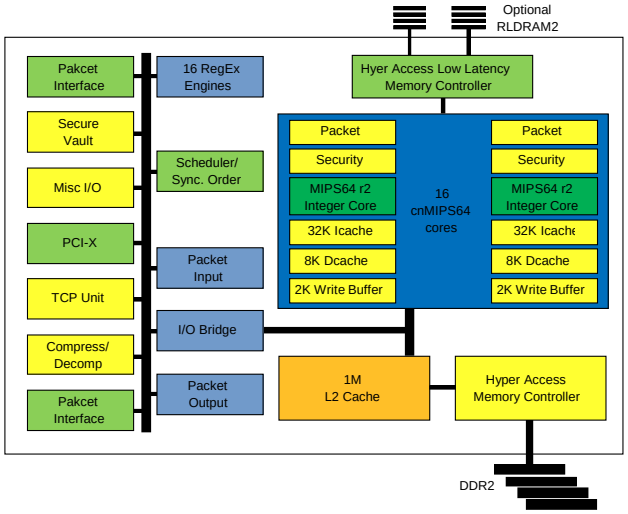

Figure 8. Diagram of Cavium CN3850

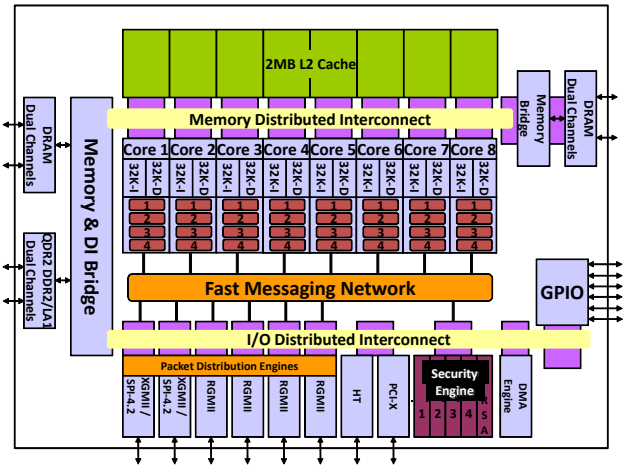

Figure 9. Diagram of Broadcom XLR 732

The different MIPS based many-core processor venders provide chips with different instruction set extensions and different interconnection architectures. Usually a Mesh, Bus or Ring is adopted for on chip inter-core connection. As shown in Figure 7 Tilera Tilepro64 uses two dimension meshes to connect 64 cores, memory banks and I/O devices [6]. Figure 8 shows Cavium CN3850 use Coherent Low Latency Interconnection to connect 16 cores with L2 Cache and I/O Bridge [7]. Figure 9 shows Broadcom XLR 732 uses a ring named Fast Messaging Network (FMN) to connect components in the processor [8].

Current mainstream vendors adopt SoC design with rich integrated network interfaces. It is very convenient for network processing and the total cost is relative low. The different architectures have different way to handle packets. But the ingress packets are stored in main memory before they were processed by CPU cores. This paper presents a Cavium CN3850 based implementation which can be also ported to other Many-core processor platforms.

B. Software Architecture

The purpose of this implementation is to construct a Traffic Multiplier. Each input packet will be duplicated for times before it was sent out. To simply the traffic reproduce the IP addresses, ports number and URLs to be injected and so on should be predefined as input for each planned test running. The Traffic Multiplier will replace the corresponding content in a packet and send it out. Logically there are many shadow flows with an original flow and the output traffic is multiples of original input traffic. There are mainly four modules in this implementation:

1) The packet receiving module: It receives packets from 1 Gbps ports and performs packet processing based on Layer 2 to Layer 4 headers. Then it schedules the packets to reproducing module. In this implementation an input packet will be scheduled to a core base on the hash value of Layer 4 headers. The hash value is generated by the dedicated packet preprocessor. This module also acts as load balancer.

2) The packet reproducing module: This module is the kernel of this implementation. Each packet will be reproduced for times. The main task is to replace the content of a packet and send it out as the new packets belongs to other flows. If each input packet has been reproduced for *T* times the output traffic bandwidth is *T+1* times of the input traffic.

3) The packet transmitting module: This module puts packets in the output queues and sends them to the 10G SFP+ interface.

4) The configuration module: The shadow IP planning, ports planning and URLs to be injected can be configured by Command Line Interface (CLI). This module accepts input from console and converts commands into rules for packet reproducing module.

Figure 10 depicts the software modules and the relationship among the modules.

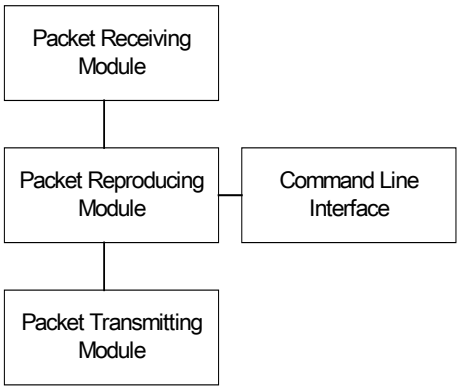

Figure 10. Software architecture of Traffic Multiplier

Figure 11 depicts the mapping from software to hardware. The packet receiving module is mapped to on chip hardware Packet Input Processing/Input Packet Data Unit (PIP /IPD) [9]. The packet reproducing module runs on 11 cores out of the total 12 cores with identical code for each. The Packet transmitting module maps to on chip Packet Output Processing Unit (PKO) [9]. The configuration module runs on core 0. The on chip Packet Order/Work Unit (POW) [9] acts as fast packet transfer channel.

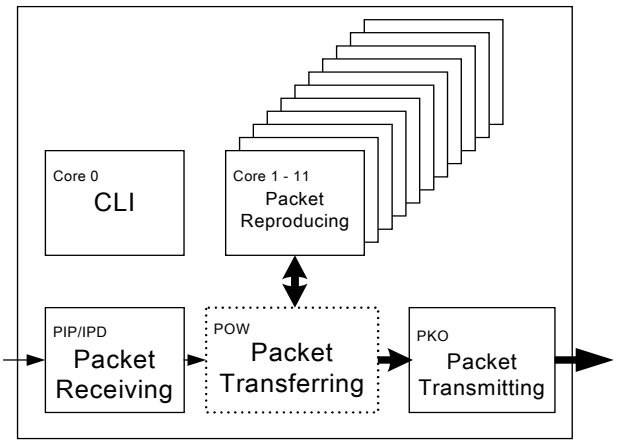

Figure 11. Software architecture mapping to hardware

C. Parallelization of Packet Processing

For many-core processors the parallelization of processing is critical to system performance. The Traffic Multiplier is packet driven system and the fast simple packet processing is a fine granularity task can be easily dispatched to cores parallel. But there are four main challenges and the solutions for them are:

1) Workload distribution: An input packet will be scheduled to a core by the hash value of Layer 4 headers. To duplicate traffic the headers and/or payload of a packet should be replace to simulate new session. The instruction and data cache for each core is relatively small. To accelerate the packet duplication each core keeps a subset of packet generation configuration. A reschedule function call puts the packet back to the head of input queue and the packet can be catch by next core immediately for inter core packet transferring.

2) Packet Ordering: As a transparent device in network system, the Traffic Multiplier should keep the packet order to avoid possible chaos and performance drop. For example the packet disorder in a TCP connection causes packet dropping and retransmission. The Packet Order/Work Unit (POW) [9], a dedicated hardware unit on chip can keep the input packet order by setting packet tags. The ATOMIC tag [9] means the packet order should be kept in each session. For the packet order keeping during duplication operation in this implementation a policy named Synthetic Packet First Out (SPFO) is adopted, i.e., all synthetic packets should be sent before the ingress packet. The ingress packets act as sentinel to guarantee all duplicated packets has been sent out. As depicted in Figure 12 the ingress packet 1 is transferred among all cores and is sent out after all reproduced packets are sent. The cost of inter core packet transferring is not much because there is only one pointer passed for each ingress packet.

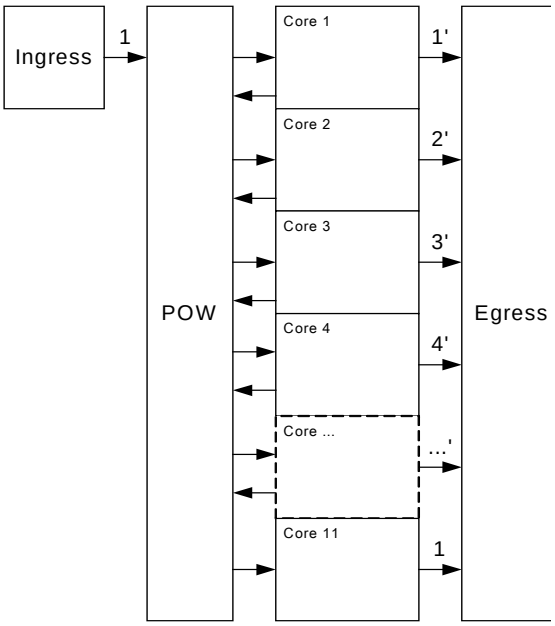

Figure 12. Packet flow in Traffic Multiplier

3) Multiplicative Factor: Theoretically the multiplier can duplicate packets arbitrarily. But the constraints including jitter, delay and throughput make the Traffic Multiplier duplicate packets up to tens of times. An observation on traffic generators showed that the performance decreased while the traffic complexity increased. This multiplier does not suffer from this issue because it only duplicates packets with minor change of packet heads and payloads if necessary. But the top speed is limited by the computing capacity and interface bandwidth in this implementation. The main advantage of this Traffic Multiplier is for those complex or hybrid application layer traffic test scenarios. The number of packet copies reproduced on each core is configurable. The output traffic is T times of input traffic. It equals the sum of copies C_i generated by core i ($i=0,\ldots,n$) plus one, i.e., the original input.

$$T = \sum_{i=0}^{n} C_i + 1 \qquad (1)$$

4) Original Session Fitering: It is important to keep the fidlity of the original traffic for a perfect test. As discussed in Section III there are two different round trip paths for original traffic. One is there is a direct loop through Traffic Multiplier. It is clear that there is to need to check the packets. Traffic Multiplier passes the packets to counterparts directly. The other is the original traffic goes into DUT and out. The original traffic must be filtered out of the synthetic traffic. One solution is keep the original traffic in a special IP range and all synthetic traffic in other IP ranges. The predefined rules keep original traffic pass throught assigned ports.

V. Performance Evaluation

The performance benchmark of the Traffic Multiplier covers latency, throughput in different scenarios. The test equipments also include Spirent avalanche/reflector 2700 [10], IXIA 400T [11] and Juniper EX4200 switch [12].

The Spirent avalanche/reflector sends out original HTTP sessions with predefined packet length and IXIA 400T inject UDP packets for latency test. These test equipments act as the traffic source in traffic cloud as depicted in Figure 6. The switch acts as the traffic convergence device and it also separates the original traffic from synthetic traffic.

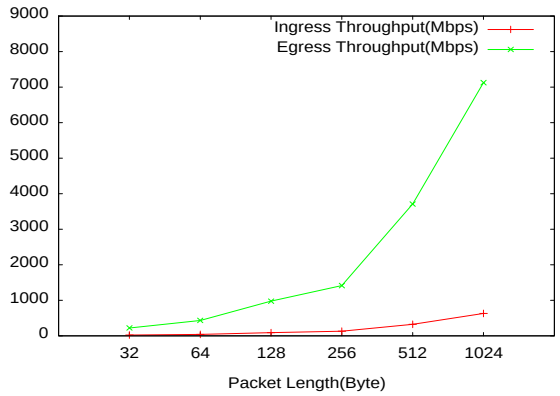

Figure 13. Ingress and max egress throughput compare(1 packet/core * 11)

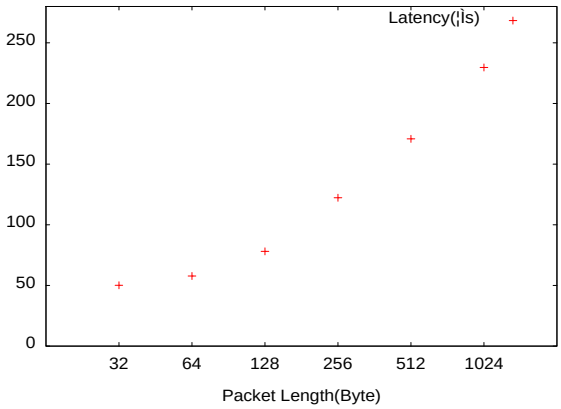

Figure 14. Latency (1 packet/core * 11)

Figure 13 depicts the ingress throughput and egress throughput comparison when each core reproduces one copy. It follows the procedure depicted in Figure 12. An ingress packet descriptor passed 11 cores and every core changes the packet including source and destination IP addresses before sends the packet out. Each core sends out a new copy of packet with different source and destination in this scenario. After the 11 packets including 10 synthetic packets plus 1 original packet are sent out for one ingress packet and the egress throughput is 11 times of the ingress throughput. The packet sizes ranges from 32 Bytes to 1024 Bytes for all benchmarking cases. The egress throughput is limited by the

packet rate processing cores can handle, the bandwidth of the main memory and also network interfaces.

Figure 14 depicts the latency test results of the Traffic Multiplier with different packet size. The latency is measured as the first bit in and first bit out, i.e., the extra latency introduced by the Traffic Multiplier when a original packet pass through it. As discussed in section IV the original packet is sent out after all synthetic packets and it is the maximum latency for all egress packets. Figure 14 also shows the bigger packet size the more significant latency. For application layer test the latency has very minor impact on test results because it is far from the propagation delay of the ideal line speed.

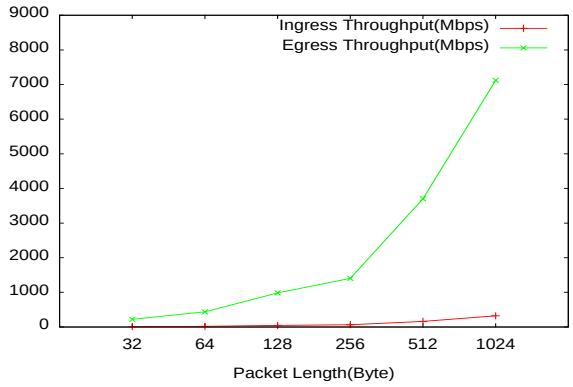

Figure 15. Ingress and Egress throughput compare(2 packets/core * 11)

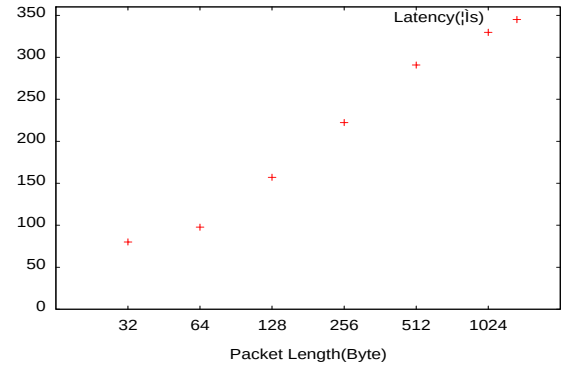

Figure 16. Latency (2 packets/core * 11)

Figure 15 and Figure 16 discloses the similar results. The only different is each core reproduces two packets. The core 11th, the last core depicted in Figure 12, sends out a synthetic packet before send the original packet out. The egress throughput is 22 times of ingress throughput.

The other test scenarios include hybrid packet size, more variety of packet sizes and more packets reproduced once by each core showed the overall performance cannot get more performance improvement. The peak packet processing capacity of a single CN3850 chip is around six million packets per second in this condition. The latency increased significantly when more than once 3 copies per core.

VI. Conclusion and Futrue Work

This paper presents a new concept of many-core processor based pseudo-stateful Traffic Multiplier. A proof of concept system basing on Cavium CN3850 EVB is implemented and benchmarked. Basing on the analysis of application layer protocols, the new concept does not touch the complex procedure of those protocols. It reproduces packets of ingress traffic to produce tens of times egress traffic. The experimental data show the Traffic Multiplier can simulate high throughput network traffic for complex test scenarios. The implementation of Traffic Multiplier makes full use of the on chip dedicated hardware and provides up to 7 Gbps egress traffic. It is an alternative for expensive dedicated hardware based traffic generators. It works with software based traffic generators and real applications can provide complex test bed especially for high layers benchmarking.

In this implementation the system performance is limited by the processor. One possible throughput scale-up is daisy chain connection for more Traffic Multipliers. Another possible improvement is to adopt hybrid hardware platforms. For example, put X86 processor platform with NSP based traffic generation cards in the same box for more complex traffic reproducing. X86 processor keeps more complex state machines for traffic generation and NSP transmits packets in high speed.

References

[1] A. Botta, A. Dainotti, A. Pescapé. “Do you trust your software-based traffic generator?” *Communications Magazine, IEEE*, vol.48, no.9, pp.158-165, Sept. 2010.

[2] R. Bolla, R. Bruschi, M. Canini, and M. Repetto. “A High Performance IP Traffic Generation Tool Based On The Intel IXP2400 Network Processor. ” *Distributed Cooperative Laboratories: Networking, Instrumentation,and Measurements*, Springer Berlin Heidelberg, 2006, pp. 127–142.

[3] Intel Inc., “Intel Internet Exchange Architecture Network Processors: Flexible, Wire-Speed Processing from the Customer Premises to the Network Core White Paper.”, White Paper, 2002.

[4] IXIA Inc., “Application Replay.”, Internet: www.ixiacom.com/products/display?skey=ixload_application_replay[Oct. 15, 2012].

[5] Cavium Networks Inc., “OCTEON Multi-Core Processor Family.”, Internet: www.cavium.com/OCTEON_MIPS64.html [Oct. 15, 2012].

[6] “TILEPro Processor Family.”, Internet: www.tilera.com/products/processors/TILEPro_Family[Oct. 15, 2012].

[7] “OCTEON CN38XX/CN36XX Multi-Core MIPS64 Based SoC Processors.”,Internet:www.cavium.com/OCTEON_CN38XX_CN36XX.html [Oct. 15, 2012].

[8] “XLR700 Series.”, Internet:www.broadcom.com/products/Processors/Carrier-and-Service-Provider/XLR700-Series[Oct. 15, 2012].

[9] Cavium Networks Inc, “Cavium Networks OCTEON CN38XX Hardware Reference Manual.”, White Paper, Sept, 2008.

[10] Spirent Commnuications Inc., Internet: www.spirent.com/Solutions-Directory/Avalanche[Oct. 15, 2012].

[11] IXIA Inc., “IXIA 400T.” Internet: www.ixiacom.com/pdfs/datasheets/ch_400t.pdf[Oct. 15, 2012].

[12] Juniper Networks Inc., Internet: www.juniper.net/us/en/products-services/switching/ex-series/ex4200/[Oct. 15, 2012].