Network Function Virtualization: Virtual Carrier Grade NAT with Linux and Intel Architectures

Alexander Shalimov
Applied Research Center For Computer Networks
ashalimov@arccn.ru

Alexander Britkin
NFWare
abritkin@nfware.com

Igor Ryzhov
Applied Research Center For Computer Networks
iryzhov@arccn.ru

Pavel Ivashchenko
Applied Research Center For Computer Networks
pivashchenko@arccn.ru

Ruslan Smeliansky
Applied Research Center For Computer Networks
rsmeliansky@arccn.ru

Abstract

This paper demonstrates and evaluates the capabilities of the Virtualized Carrier Grade NAT with Linux and Intel Architecture which leverages abilities of the multicore systems with reducing network communication overhead in Linux OS.

The paper describes how virtualized CG-NAT is implemented on an Intel architecture server platform to overcome network virtualization challenges using Intel Data Plane Development Kit and achieve the carrier grade service.

1 Introduction / Motivation

CG-NAT [1-5] is a «centralized NAT» placed in the service provider's network. It can be an addition to the NAT at the customer edge (NAT444) or instead to the customer NAT (DS-Lite approach) and follows the concept to pull away public IPv4 addresses from the customer site, where their multiplexing capacity is not efficiently utilized, to the outside of the centralized CG-NAT where many customer networks can share a single public IPv4 address.

Presented in this paper NFWare CG-NAT is working in NAT444 operational mode and maps each application flow on customer edge to the public IPv4 address and one of its TCP or UDP ports as identified by the combination of a private IPv4 address and a TCP or UDP port. CG-NAT multiplexes the addresses of many inside devices to a single outside address by mapping application flows. The typical use-case of NAT444 for service provider is to continue to assign addresses to large numbers of new customers when there are no new IPv4 addresses to use.

One of the biggest challenges of virtualization of data plane intensive functions like CG-NAT is to achieve both a data plane with high-performance forwarding and carrier grade quality in a virtualized environment. In order to improve the data plane forwarding performance in a virtualization environment of virtual CG-NAT we adopted the Intel Data Plane Development Kit (DPDK).

The Intel DPDK is a set of data plane libraries designed for high-speed networking [6]. Compatible across many types of Intel CPU, the Intel DPDK offers a software programming model that can accommodate different network applications and system configurations.

For virtual network functions, like CG-NAT, Intel DPDK via its set of API offers low-overhead alternatives to traditional Linux system calls and enables the creation of virtual network functions that can effectively scale in performance. According to this concept, CG-NAT by leveraging Intel DPDK processes the majority of packets outside the Linux kernel thus avoiding Linux OS overheads such as timers, lockers and threads, however Linux networking stack is still involved for processing of signaling and control protocols (e.g. BGP.) With this approach most of the CPU cores can be dedicated to perform network address translation functions maximizing the overall throughput of CG-NAT.

In this paper, we present a novel Virtual Carrier Grade NAT that works on commodity Intel architecture server platform and meets carrier grade requirements such as throughput, low latency and scalability.

2 Architecture

Logically every CG-NAT cosists of two main translation modules: MATCH and RW (see figure 1).

Module MATCH performs translation table lookup for each incoming packet from an internal network. The lookup is done using 5 fields of a packet: ip_src, ip_dst, type, port_src, port_dst (actual number of fields depends on mapping policy: endpoint independent, address&port dependent, and so on). Translation rules are stored in a mapping table: ip_src and port_src which should be changed during translation.

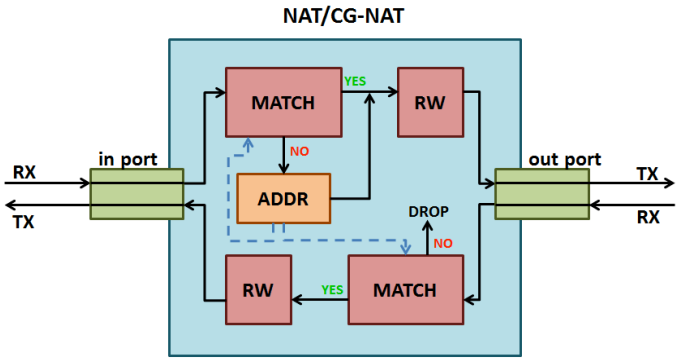

Figure 1. CG-NAT basic scheme.

If the mapping is successful, the packet is forwarded to RW module which performs the field rewrite operation and

calculates the control sum. If the mapping is not successful, the packet is forwarded to the module which identifies the translation rule and according to translation modes (pooling, port assignment, port parity, port contiguity) chooses the correct ip_src and port_src . Then packet is forwarded to RW module and to an external network. The process of packet processing from external network is almost the same with one exception: if there is no rule in a reverse address translation table, the packet is dropped.

Figure 2 shows our advanced scalable multicore design to provide high performance communicating in a virtualized environment. Main core is responsible for running maintaining tasks: starting other working threads, providing command line interface to the system, displaying statistics and other infromation about the system, answering on network request to the system like ARP and PCP. NAT cores process packets coming from an internal and external networks as decribed in the basic scheme. We assign a core per a physical port and separate inbound and outbound traffic processing via marking physical ports as internal or external. Several cores are dedicated for cleaning up the tables after sessions timeout. The logging task runs on the separate cores as well in order not to slow down main packet processing while writing log messages on a hard disk. Note the numbers of core per tasks are configurable depending on performance requirements.

All cores have access to main memory (1Gb huge pages) where all required tables resides: arp/route table, client tables, flows tables, sessions table, external IPs pool table, etc.

Intel DPDK is mainly used for interaction with NICs, intercore communication, working with the main memmory (buffer allocation, etc).

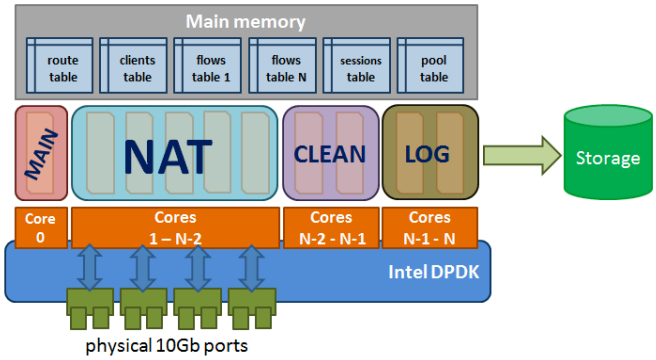

Figure 2. CG-NAT multicore design.

The Application Layer Gateway (ALG) functionality allows CG-NAT to transparently translate IP addresses and ports in messages for protocols like FTP, SIP, RTSP, PPTP, ICMP and others. Each ALG is implemented as a pluggable module.

3 Experimentation results

For performance evaluation we used our netmap-based custom traffic generator [7] sending as much as possible UDP packets to CG-NAT. CG-NAT translates and forwarded packets to the external network. The result has been measured on the hardware switch directly connected to CG-NAT. CG-NAT runs on COTS hardware with 2 CPU 8 cores each, 3GHz, 16GB RAM and 10Gbit network cards.

Table 1 summarizes the overall performance numbers for CG-NAT. It supports maximum 12Mpps with first packet delay equal to 80us and 40us for further packets. Currently we can maintain 7.5 million sessions due to main memory limitation. Connection setup time is 0.6 Million for the most hard scenario with enabled arbitrary mode. With enabled port block allocation mode we achieve 5 million connections per second.

Table 1. CG-NAT performance numbers (per 10Gbit)

Throughput, packets per second (64 bytes) • 1 flow • 1000 flows	 10 Million 6 Million
New connections per second	0.6 Million
Concurrent connections	7.5 Million
Latency, us • first packet delay • next packets delay	 80us 40us

The figure 3 shows the performance results for different packet sizes in kilopackets per second. The red line is theoretical maximum number of packets per second for 10Gb channel. The blue line is our CG-NAT system. The green line is Open vSwitch (OVS)-based kernel NAT. It has the same rate for all sizes due to limitation of the Linux kernel networking stack. Our system demonstrates the higher numbers and is able to process all 10Gb channel bandwidth starting from medium packet sizes

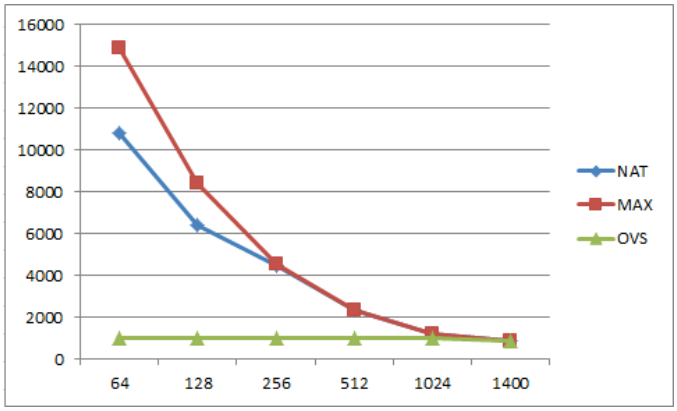

Figure 3. CG-NAT Performance Results (kpps)

References

[1] RFC 2663, IP Network Address Translator (NAT) Terminology and Considerations

[2] RFC 4787, Network Address Translation (NAT) Behavioral Requirements for Unicast UDP

[3] RFC 5382, NAT Behavioral Requirements for TCP

[4] RFC 5508, NAT Behavioral Requirements for ICMP

[5] RFC 6888, Common Requirements for Carrier-Grade NATs (CGNs)

[6] Intel DPDK, http://intel.com/go/dpdk

[7] Netmap, http://info.iet.unipi.it/~luigi/netmap/