The Limits of Architectural Abstraction in Network Function Virtualization

Balazs Nemeth, Xavier Simonart, Neal Oliver
Network Platforms Group
Intel Corporation

Wim Lamotte
iMinds/tUL/UHasselt-EDM
Diepenbeek, Belgium

***Abstract*—Network Function Virtualization (NFV) is the paradigm of implementing network services as a network of functions on top of commodity off-the-shelf (COTS) servers. It is of profound interest to telecommunications network operators because of the promise of bringing the economics of data centers into the network. However, existing CPU, memory, and network interface architectures cause network service performance to be sensitive both to the implementation of individual network functions, but also to their placement within an NFV platform. For this reason, emerging NFV standards will confront the challenge of exposing certain platform architectural parameters to enable services to be orchestrated in an effective manner. The goal of the paper is to show that underlying technologies must be exposed to the Orchestrator during service deployment as incorrect placement can have a very significant impact on performance. We illustrate this by describing a "proof-of-concept" implementation of Quality of Service (QoS) for a Broadband Remote Access Service (BRAS)/Border Network Gateway (BNG). Our work focuses on studying performance implications related to PCIe bandwidth constraints, Virtual Network Function (VNF) placement, message buffer (mbuf) size and memory channel utilization.**

Keywords*—*BRAS, Broadband Remote Access Server, BNG, Border Network Gateway, NFV, Network Function Virtualization, QoS, Quality of Service

I. Introduction

Network function virtualization (NFV) is the paradigm of implementing telecommunications network services as a network of functions running on commodity off-the-shelf (COTS) hardware. NFV has as its goal consolidating the many generations of network equipment, along with its operational costs, into a more uniform "data center"-like model, in which new services can be introduced by deploying software rather than hardware-based network elements. [1]

The framework in which network services exist in NFV is depicted in Fig. 1. Virtual Network Functions (VNFs), virtualized implementations of network functions, run on top of an NFV Infrastructure, in which computing, storage, and network resources are furnished to the VNF when they are instantiated. The virtual resources are based on a hardware infrastructure, in the general case a server or data center network.

In this framework, VNFs are composed into Network Function Forwarding Graphs (NFFG) to implement an end-to-end network service, as depicted in Fig. 2. VNFs exchange packets with each other in the control and/or data planes

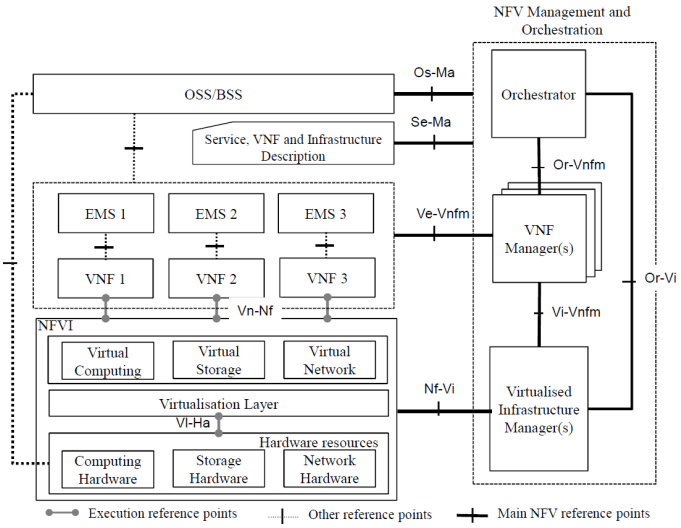

Fig. 1. NFV reference architecture framework [2] (© European Telecommunications Standards Institute 2013. Further use, modification, copy and/or distribution are strictly prohibited).

to perform their function. The Orchestration function of the NFV Framework and the VNFs themselves instantiate the NFFG on the framework. This requires that available resources in the NFV framework, and requirements of the VNFs, be described with enough precision and completeness to enable a performant NFFG to be instantiated.

Modern computing architectures incorporate a vast array of CPU, memory, and network technologies in order to achieve high performance (e.g., [3]). The performance of virtualized network services are sensitive both to the implementation of VNFs and to the orchestration of VNFs into a NFFG. NFV does not mandate a specific infrastructure architecture, but performance will play a significant role in the success of NFV in the real world.

The goal of the paper is to show that many of these technologies should be taken into account during deployment of VNFs. This claim is supported by a study of a high performance prototype of Border Network Gateway (BNG) with its constituent VNFs developed to determine the feasibility of implementing this network service in NFV.

The remainder of this paper is structured as follows: The BNG network service, and, in particular, its constituent VNFs that implement QoS are described in Section II. The results

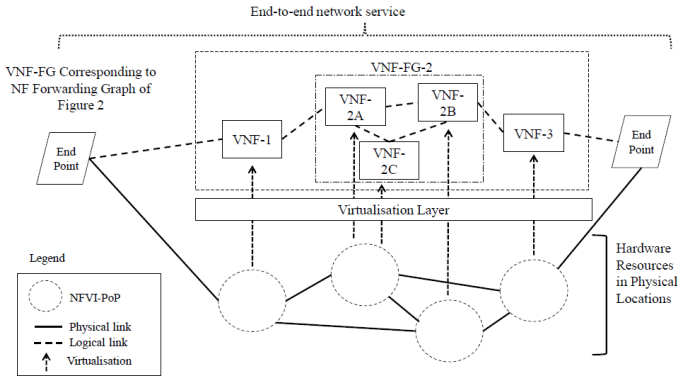

Fig. 2. Example of a network service [2] (© European Telecommunications Standards Institute 2013. Further use, modification, copy and/or distribution are strictly prohibited).

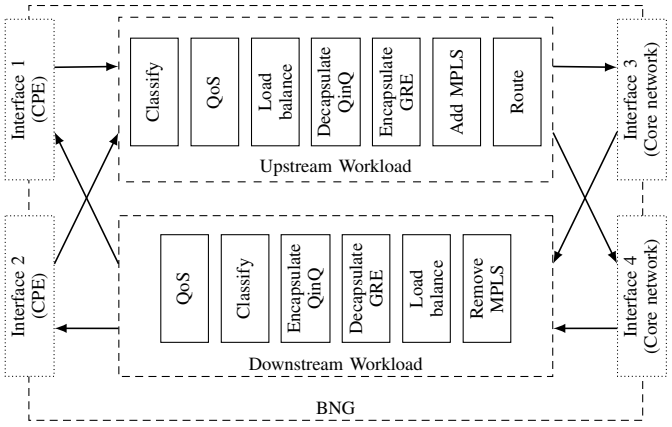

Fig. 3. Upstream and downstream workload for the prototype BNG.

and generalizations are reported in Section III. Section IV lists related work and Section V concludes with a discussion of the implications of this work to NFV.

II. Border Network Gateway

A. BNG context

A BNG is an operator network service that provides an IP interface to customer premises equipment (CPE) (e.g., [4], [6]). A BNG is an extremely complex network element, responsible for authorization/authentication/accounting (AAA), access control, session management, packet forwarding between CPE and the core network, and providing QoS for subscribers. A BNG is typically fronted by a multiplexer, e.g., a Digital Subscriber Line Access Multiplexer (DSLAM), so that an aggregation of many thousands of user sessions are presented on a single BNG network interface. While current BNGs are built on large-scale aggregation routers (e.g., [10]), VNFs are already being developed as proofs of concept (e.g., [4], [5], [9]).

B. BNG Architecture

The work reported in this article builds on the prototype BNG described in [9], available at [11], as depicted in Fig. 3. In this prototype, the network interfaces to the CPE appear on the left, and the network interfaces to the core network appear on the right.

The network functions implemented in the prototype consist of load balancers, routing, QoS and various other processing functions. These elements are assigned to threads, which are allocated to cores of a server as required by the demands of a benchmarking run.

C. Use-cases

The use-cases implemented for the prototype consist of encapsulation and decapsulation of MPLS tags, and routing between QinQ tunnels on the CPE side and GRE tunnels on the core network side. This is a subset of the functionality that a deployed BNG supports, but it allows performance issues to be investigated, and in fact it demonstrates the NFV use-case of the incremental substitution of VNFs for physical network functions.

This prototype receives CPE sessions represented by double-tagged VLAN (QinQ) packets, routes user packets to the correct destination (i.e., the correct core network interface), adds or removes MPLS tags as required, and terminates GRE tunnels at the core network interface.

The CPE packets are double-tagged in order to allow a large number of users (32768) to be defined by varying 7 bits in the service VLAN tag and 8 bits in the customer VLAN tag. The CPE packets are 64 bytes in length, to present the worst-case network workload (When we refer to the size of packets, we are describing the length of the Ethernet frame, i.e. starting at the destination MAC address and including the Frame Check Sequence). Each CPE interface receives a traffic set of 32768 users. On the core network side, 65536 different GRE tags are used, and the packet size is 78 bytes.

The user packet streams are assumed to be limited to 10 Mbps. For traffic engineering reasons, the QoS element is configured to enable up to at least 5 ms of data to be buffered during QoS enforcement.

D. QoS Architecture

A QoS implementation must provide the ability to guarantee throughput and latency properties for network flows based on a classification of those flows by subscriber, traffic type, or other features. This is done by policing and shaping the flows so that they obey certain limits. Fig. 4 depicts the architecture proposed for systems using Data Plane Development Kit (DPDK) [12]. A general pipeline consists of: packet reception, classification, policing, load balancing, workers, dropper, scheduler and packet transmission.

A minimal QoS implementation thus requires packet reception, classification, a scheduler and packet transmission. Even in this case, QoS parameters, classification implementation (mapping from packet to a tuple) and core layout (e.g. Simultaneous Multi-threaded (SMT) vs. non-SMT, combining QoS and TX on a single core) have implications on performance and must be chosen carefully as we will show later.

E. Focus on the QoS Element

At the highest level, QoS is a buffer that can hold thousands or millions of packets. In reality, this buffer is a collection of

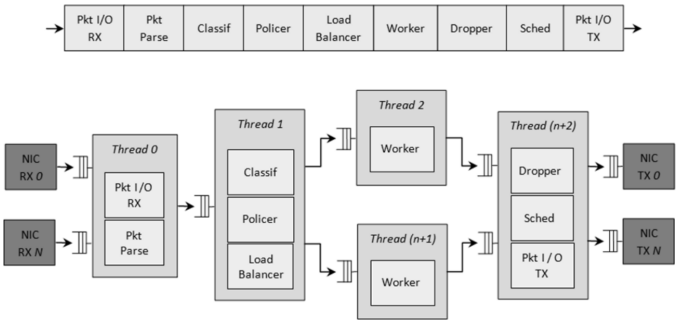

Fig. 4. Prototype QoS architecture (from [12]).

TABLE I. QOS PARAMETERS

Parameter	Definition
subports per port	Number of user sets; QoS enforcement is constrained within a user set, and each user set receives a configured fraction of the total port rate.
number of pipes	Equivalent to the number of users; performance measurement in this work assumes 32768 pipes per interface and per QoS instance.
tb_size	Token bucket size, the maximum number of credits the bucket can hold.
tb_rate	Token bucket rate, the rate at which tokens enter the bucket.
tc_period	Traffic class period, the time interval between traffic class token updates.
queue size	Capacity of the QoS queue in unit of packets; it is governed both by memory constraints and by use-case buffering requirements.
p	Token bucket period, the time interval between token updates. DPDK calculates this value from tb_size and tb_rate.
c	Credit per period, the number of credits entering the token bucket in each period. DPDK calculates this value from tb_size and tb_rate.

queues organized in a hierarchy of five levels. The port (i.e. the physical interface) is at the root of the hierarchy followed by the subport (a set of users), the pipes (individual users), the traffic classes (each with a strict priority) and at the leaves, the queues.

The QoS element consists of two operations: enqueueing of packets from reception and dequeueing of packets for transmission. The enqueue and dequeue operations are governed by a "leaky bucket" algorithm, in which credits are posted to the bucket at a uniform (but configurable) rate and consumed when packets are sent. Each credit represents the ability to transmit one byte. Enqueueing and dequeueing are constrained to run on the same logical core in order to avoid the use of locks. Because of this hierarchy, the implementation actually supports two token bucket types, one associated with a pipe, and one associated with a traffic class. In the performance characterizations reported in this paper, one subport was used.

The enqueue operation uses the information from the classification step to select a destination queue for the packet, and to drop the packet if the destination queue is full. This is the only point where the QoS element can decide to drop the packet.

The hierarchy of features implemented by the QoS element corresponds to parameters of the algorithm. These parameters are summarized in TABLE I.

The values of c and p are calculated by DPDK at configuration time as an approximation of tb_rate. While they respectively determine the credits per period and the period between updates, it does not mean that updates always happen after p units of time. QoS automatically trades off shaping quality and precision for processing time if the implementation on the CPU is not executing fast enough.

Fig. 5 illustrates conceptually (assuming one traffic class) how these parameters influence the QoS data structures and how traffic is shaped for a single user. In reality, approximations and impacted CPU performance also have an influence. The graph shows how the number of tokens in the token bucket, buffered packets, queue occupancy and the output rate evolve over time depending on the input rate.

Initially, if the input rate is zero (no packets), tokens in the token bucket can accumulate up to a moment when the token bucket is full (1). When the first packet enters the system, the output rate and the input rate are the same. There are enough tokens in the token bucket to allow packets to pass through (2). If the input rate is lower than (3) or equal to (4) the shaping rate, then the number of tokens in the token bucket does not decrease. Consumed tokens are replenished immediately. When the input rate rises above the shaping rate (5), the number of tokens in the token bucket starts to decrease. If the input rate remains high, after some time all tokens will be consumed, and the output rate decreases up to the shaping rate (6). At the same time, packets are buffered (7). If the rate remains high, the packet queue will become full, and packet drop will occur (8). The output rate remains the same as the shaping rate (8). When the input rate finally drops below the shaping rate (9), the output rate can remain at the shaping rate as long as there are packets queued. If the input rate remains below the shaping rate, the number of tokens in the token bucket can again increase when the packet queue is empty (10).

III. IMPACT OF DESIGN DECISIONS ON PERFORMANCE

Through performance evaluation of the BNG network functions, we show the principal aspects of the underlying technologies that determine overall system performance. These features include: PCIe bandwidth considerations, VNF placement and memory utilization. We explain why each factor plays a role for the BNG so that they can be generalized to other VNFs.

While implementations can be fine-tuned and optimized by hand in a test environment, where all aspects of the system can be controlled, the abstraction provided by the virtualization layer and virtualized resources can negate the optimizations and can make it difficult or impossible to achieve acceptable performance. We argue that an Orchestrator that is aware of these factors can make better decisions during VNF deployment.

A. PCIe Bandwidth Considerations

Some work (e.g. [8], [13]) has shown that systems using traditional dual port 10GbE PCI Express 2.0 network cards may hit PCIe bandwidth limitations when trying to handle the smallest packet sizes (64 bytes) at line rate.

It is therefore important to note that the packets at the CPE side are smaller than at the core network side. When

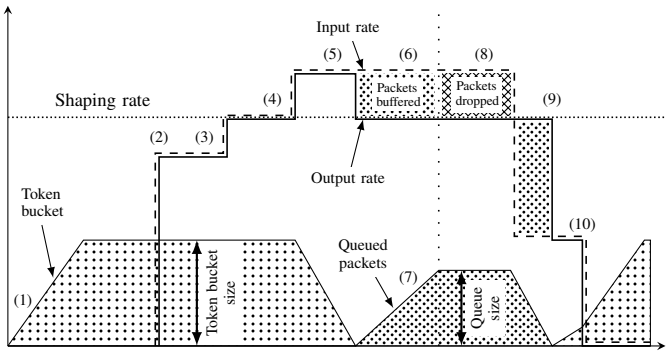

Fig. 5. QoS data structures and traffic shaping for single user (conceptual).

both interfaces at the core network side are located on the same network card and both interfaces at the CPE side are located on another network card (configuration 1), the pressure on the PCIe bus is higher for the core network card than when each network card supports one interface at the CPE side and one interface at the core network side (configuration 2). In the second configuration, the load is distributed more equally. The hardware can be used more efficiently by taking these limitations into account. TABLE II compares the throughput of the two configurations. With the smallest packet size (64 bytes at the CPE side and 78 byte on the Core network side), total throughput can be improved by 1.22 Mpps simply by using configuration 2.

Any VNF, or a chain thereof, that increases or decreases the size of packets between ingress and egress should be aware of which interfaces are residing on the same physical network card.

B. Impact of Socket

When dual socket systems are used, it is important to realize that PCIe cards are attached to one specific CPU socket, and that accessing data from the other socket has a cost.

Fig. 6 highlights this, using the same BNG workload, without QoS. In both configurations, the network cards are inserted in the first PCIe slots of the system, attached to CPU socket 0. In the first configuration ("correct socket"), the cores being used by the workload are also located on CPU socket 0. In the second configuration ("incorrect socket"), the cores being used are located on CPU socket 1. Hence, it is very important for the Orchestrator to be aware of these details during VNF placement on systems with multiple sockets. Special care should be taken to ensure that cores allocated to the VNF reside on the same socket as the network cards.

C. Impact of Core Layout

A minimal QoS configuration requires at least four functions: Receive, Classify, QoS, and Transmit. For a QoS implemented as a unitary VNF, the implementer has the ability to choose the layout of the elements among cores, which in turn must inform the Orchestrator when choosing the QoS function for a network service. The layout choice has a material impact on the performance of the QoS element.

TABLE II. THROUGHPUT IN MPPS FOR INTERFACE LAYOUTS

Packet Size (bytes)	Theoretical Maximum (Mpps)	Throughput per configuration (Mpps)	
		configuration 1	*configuration 2*
64-78	51.02	42.80	44.02
128-152	29.07	28.36	28.50
256-280	16.67	16.56	16.56
512-536	8.99	8.92	8.92
1024-1048	4.68	4.56	4.56
1494-1518	3.25	3.24	3.24

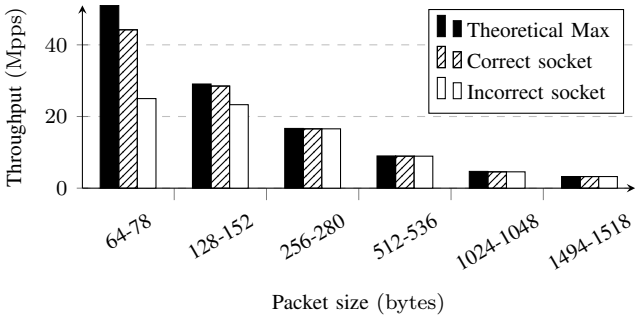

Fig. 6. Influence of socket on throughput.

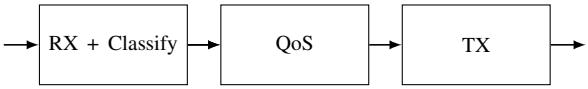

Fig. 7. Functions allocated to separate cores.

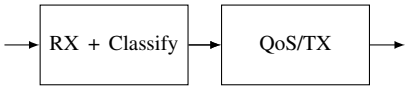

Fig. 8. QoS/transmit co-allocated to a core.

Five alternate layouts were tested. The "baseline" layout allocates the functions to separate cores (while combining Receive and Classify), as shown in Fig. 7. Using this layout, line rate (14.88 Mpps @ 64 bytes) can be achieved as can be seen in Fig. 10. As an alternative to this layout, QoS and Transmit were also co-located in a single core, as shown in Fig. 8.

In addition to dedicated cores, combinations of functions were assigned to SMT threads within a core, such as shown in Fig. 9. This is similar to the layout of Fig. 7, except that the functions are allocated to SMT threads rather than dedicated cores. Other layouts were tried, but will not be shown in figures here in the interest of brevity.

The throughput performance of the different core layouts is shown in Fig. 10. The effects of Data Translation Lookaside Buffer (DTLB) misses will be discussed later. One possible conclusion to be drawn from these results is that increasing the frequency at which dequeueing from the QoS element is done has a material effect on end-to-end performance. For example, the second alternative is slower than the first because QoS and Transmit run in a common core, which reduces the time that can be spent processing QoS packet dequeueing.

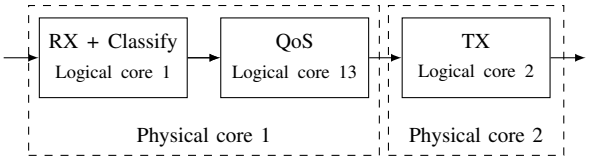

Fig. 9. Use of separate SMT threads rather than separate cores.

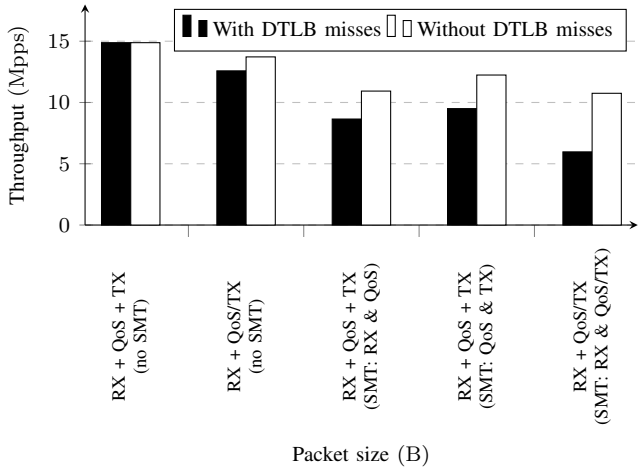

Fig. 10. Performance as a function of core/SMT thread layout.

In the configurations where SMT is used, overall performance is lower compared to the non-SMT case. We would see improved performance if memory access was slowing down execution but the DPDK QoS framework has been optimized to deal with cache misses. The motivation is that the QoS will need to access its data structures from memory instead of cache because the sheer amount of packets being buffered would cause those data structures to be evicted. Note however that, in some SMT configurations, performance per physical core is increased.

The third configuration, where Receive and QoS are allocated on the logical cores of the same physical core, shows different behavior than the other four cases. In this case, not all message buffers (40K out of the 2M that were provisioned) are stored in the QoS hierarchy (i.e. in the QoS buffers). In this case, the bottleneck is the Receive core. Comparing this configuration to the fourth configuration where QoS and TX reside on the same physical core instead of RX and QoS, we can see that by supplying more packets to the QoS, throughput can be increased from 8.65 Mpps to 9.50 Mpps. Performance is still limited by the QoS dequeue operation which now has to block on packet transmission to the TX core. The TX core performance is impacted by being allocated on the same physical core as the QoS.

In general, the Orchestrator should instantiate the VNF with a CPU topology that reflects the underlying hardware for VNFs running workloads which are sensitive to SMT thread layout.

D. Impact of Memory Usage

In DPDK, a message buffer (mbuf) contains both the packet data and metadata, such as the packet length. For the following experiments, the number of mbufs in the QoS elements was held constant, but their size was increased. Different mbuf sizes were tested with different function layouts in cores, to understand how these configurations interacted with each other and with the CPU's memory cache and DTLB. Note that in real-world implementations, the size of the part of the mbuf that holds the packet data is determined by the maximum-transmission unit (MTU).

For these experiments, the Transparent Huge Pages feature of Linux was disabled, so that the huge page management algorithms of DPDK would be the only elements allocating data in huge pages.

Fig. 11 shows the throughput of the QoS element as a function of the total memory used by the mbufs. When the buffer size reaches 1 GB, throughput begins to drop because entries in the DTLB begin to thrash. The point at which this occurs is CPU-specific.

These results are specific to the layout shown in Fig. 9. Studying which physical addresses are chosen by the memory allocation algorithm reveals that three DTLB entries would be required when between 1 GB and 2 GB of memory is used by the buffers to access all allocated memory. On the specific test system the DTLB is fully associative, has four entries for 1GB huge pages and is shared by SMT threads. From this, we conclude that two additional DTLB entries are required to prevent thrashing.

The cost associated with using more mbufs depends on the probability that the corresponding entry is in the DTLB. For this reason, the performance impact is the highest when we have a few mbufs inside a page (the page is mostly unused) and lowest when nearly all the memory accessible through the entry is used.

Repeating the same test for the layout shown in Fig. 7, the impact of DTLB misses cannot be seen directly by measuring throughput, while the CPU is fully utilized even without DTLB misses. While packets should only be buffered when the input rate rises above the shaping rate (refer back to Fig. 5), due to the time required to handle packets (which now includes DTLB misses), the QoS implementation compensates by buffering more packets to reduce the amount of work required to find outstanding packets in the hierarchy. Fig. 12 shows the number of packets buffered. With this core layout, the DTLB is trashed starting from 3GB confirming that QoS element is not sharing the DTLB with another logical core. The effect of DTLB misses for other core layouts is shown in Fig. 10.

While these results are specific to the QoS implementation, they can be generalized to other VNFs that require similar amount of memory.

E. Influence of Queue Utilization

To increase the available bandwidth to memory, modern systems use multiple memory channels. Memory bandwidth should be maximized to minimize the time required to move data from main memory to the CPU caches.

The QoS implementation allocates a contiguous block of memory to store the QoS hierarchy. This allows to access a specific queue by calculating the correct offset within the

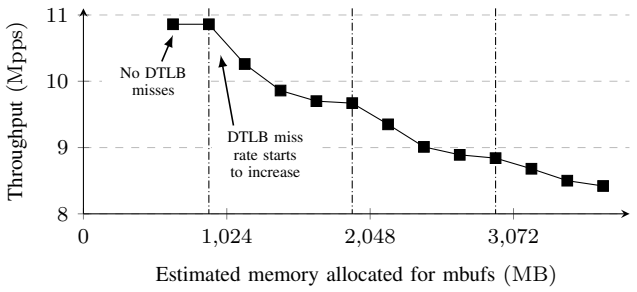

Fig. 11. Throughput as a function of estimated memory allocated for mbufs.

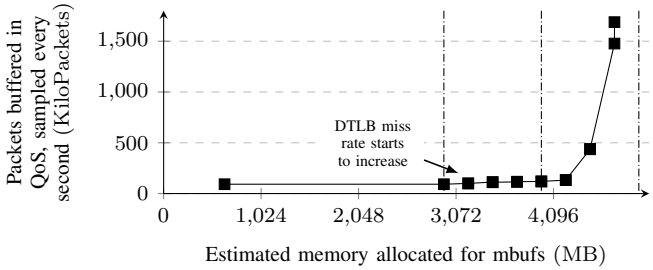

Fig. 12. Number of packets buffered in QoS as a function of estimated memory allocated for mbufs.

allocated block. There is no need to walk through the hierarchy using pointers.

As noted previously, the whole hierarchy cannot be held in the CPU cache at all times and main memory will need to be accessed.

The system under test implements four memory channels. Two packet streams were used to load the system under test. In both cases, Intel® PCM tools were used to measure memory bandwidth utilization expressed in megabytes per second (MBps).

While both streams contained packets from all users, the difference was that packets from the first stream would be classified to one of the traffic classes and one of the queues randomly and packets from the second stream would be classified to the same traffic class and the same queue for each user. The result of this stream is that only a portion of the allocated memory block would be utilized and the remaining portion would never be touched.

With the first packet stream, the memory channel bandwidth utilization for the first, second, third and fourth memory channel has been measured at 519MBps, 562MBps, 587MBps and 562 MBps respectively. For the second packet stream, the memory bandwidth per channel changed to 842 MBps, 453MBps, 478MBps and 454MBps.

We argue that, due to the limited amount of memory bandwidth available per channel, performance can be impacted if memory allocated to a VNF does not take the access patterns into account.

IV. Related Work

Other groups have been active in related fields, either with implementations of NFV functions using DPDK, or with implementations of virtualized BNG.

Matsusani [5] describes a BNG implementation, but focuses on a vendor-neutral architecture suitable for standardization. Our work differs from this in that we study the impact of VNF layout and memory usage on service performance.

Pongrácz [7] describes the implementation of a router using DPDK. The routing functions of our prototype are substantially similar to theirs, as both implementations depend on the functionality of DPDK. Our work focuses on the QoS function, the most resource demanding function of our prototype, and the impact of core layout and memory usage on performance.

Bifulko [4] describes a virtualized implementation of a BNG using an alternative framework called Click. Their BNG implementation is similar to the BNG described in [9] which is the basis of our current work with QoS, but uses different technology in its implementation, and the results reported focus on overall performance rather than the impact of design decisions on performance.

Cerrato [14] compares performance between different designs of software components responsible for moving data between many small virtual machines, each with a tiny workload, and a vSwitch. Our work differs in that we focus on performance implications of the underlying hardware parameters within one big network function that occupies a significant portion of the server's resources.

Hwang [15] describes the design of NetVM, a flexible platform providing zero-copy inter-VM communication. Instead of running into hardware limitations, their implementation is constrained by the available processing capacity.

V. Conclusion

We describe a high-performance prototype implementation of a BNG and of a QoS network function, implemented by means of DPDK. This prototype is used to investigate design and orchestration choices that a VNF implementer or an NFV Orchestrator would need to make in order to obtain maximum performance of these network functions in a real world deployment.

By examining achievable performance of our prototype implementation, we have found examples where hardware features play a significant role in determining performance. We have shown how and when their impact can be measured.

We believe that the VNF implementer should specify platform features, such as CPU, memory, and network parameters, and function layout constraints on a platform, that must be taken into account when instantiating a network service. We have shown that, without this information, it is difficult to make optimal use of the underlying hardware and to optimize the system to reach performance requirements.

We have not focused on how information about these platform features can be exploited during deployment. Future work in this area should include a more complete treatment of these parameters and constraints as well as Orchestration algorithms that can make use of them.

REFERENCES

[1] White Paper, "Network Functions Virtualization: An introduction, Benefits, Enablers, Challenges, & Call for Action", [Online]. Available: http://portal.etsi.org/NFV/NFV_White_Paper.pdf

[2] ETSI Group Specification, "Network Functions Virtualisation (NFV); Architectural Framework", ETSI GS NFV 002, V1.1.1 (2013), [Online]. Available: http://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.01.01_60/gs_NFV002v010101p.pdf

[3] Ulrich Drepper, "What Every Programmer Should Know About Memory", Red Hat White Paper, 2007, [Online]. Available: http://www.akkadia.org/drepper/cpumemory.pdf

[4] Roberto Bifulco, et. al., "Rethinking Access Networks with High Performance Virtual Software BRASes", Second European Workshop on Software Defined Networks, 2013.

[5] Hitoshi Masutani, et. al., "Requirements and Design of Flexible NFV Network Infrastructure Leveraging SDN/OpenFlow", ONDM 2014, 19-22 May 2014, Stockholm, Sweden.

[6] Ramesh Nagarajan and Sven Ooghe, "Next-Generation Access Network Architectures for Video, Voice, Interactive Gaming, and Other Emerging Applications: Challenges and Directions", Bell Labs Technical Journal 13(1), 69-86, 2008.

[7] Gergely Pongrácz, et. al., "Removing Roadblocks from SDN: OpenFlow Software Switch Performance on Intel DPDK", Second European Workshop on Software Defined Networks, 2013.

[8] "Network Function Virtualization: Quality of Service in Broadband Remote Access Servers with Linux and Intel® Architecture®", Intel Reference Architecture, 2014, [Online]. Available: http://networkbuilders.intel.com/docs/Network_Builders_RA_NFV_QoS_Aug2014.pdf

[9] "Network Function Virtualization: Virtualized BRAS with Linux* and Intel® Architecture", Reference Architecture, 2014, [Online]. Available: http://networkbuilders.intel.com/docs/Network_Builders_RA_vBRAS_Final.pdf

[10] "Cisco ASR 9000 Series Aggregation Services Router Broadband Network Gateway Configuration Guide", Release 4.3.x, [Online]. Available: http://www.cisco.com/c/en/us/td/docs/routers/asr9000/software/asr9k_r4-3/bng/configuration/guide/b_bng_cg43xasr9k/b_bng_cg43asr9k_chapter_01.html

[11] "Intel® Data Plane Performance Demonstrators", version 0.11, [Online]. Available: https://01.org/intel-data-plane-performance-demonstrators

[12] "Intel® Data Plane Development Kit: Programmer's Guide", Number 326003-006. [Online]. Available: http://www.intel.com/content/www/us/en/intelligent-systems/intel-technology/intel-dpdk-programmers-guide.html

[13] Dong Zhou et. al.. "Scalable, High Performance Ethernet Forwarding with CUCKOOSWITCH" http://www.cs.cmu.edu/~dongz/papers/cuckooswitch.pdf

[14] Ivano Cerrato et. al.. "Supporting Fine-Grained Network Functions through Intel DPDK" https://www.fp7-unify.eu/files/fp7-unify-eu-docs/Results/EWSDN-Unify_Supporting%20Fine-Grained%20Network%20Functions%20through%20Intel%20DPDK.pdf

[15] Jinho Hwang et. al.. "NetVM: High Performance and Flexible Networking Using Virtualization on Commodity Platforms" https://www.usenix.org/system/files/conference/nsdi14/nsdi14-paper-hwang.pdf