Towards Optimal Adaptation of NFV Packet Processing to Modern CPU Memory Architectures

Christian Sieber, Raphael Durner, Maximilian Ehm, Wolfgang Kellerer
Chair of Communication Networks
Technical University of Munich, Germany
c.sieber@tum.de,r.durner@tum.de,maximillian.ehm@tum.de,wolfgang.kellerer@tum.de

Puneet Sharma
Hewlett Packard Labs
Palo Alto
puneet.sharma@hpe.com

ABSTRACT

Network Functions Virtualization (NFV) aims to move network functions away from expensive hardware appliances to off-the-shelf server hardware. NFV promises higher flexibility and cost reduction for the network operator. In order to achieve high throughput performance with this commodity hardware, fast packet processing frameworks like NetMap or the Data Plane Development Kit (DPDK) can be used. It is known that packet processing performance is very sensitive regarding copying of packets. In this paper we take steps towards quantifying the efficiency of NFV regarding packet copying overhead at hardware level. As modern servers are often built up of multiple CPUs with segregated memory, we evaluate the performance penalties resulting from this segregation in conjunction with DPDK. Additionally we evaluate the effects of cache misses on packet processing in detail. Subsequently a metric that quantifies the efficiency of a running VNF is introduced and an optimization scheme is outlined which describes the use of the metric. Our results show how both cache misses and memory segregation reduce the network efficiency.

CCS CONCEPTS

• **Networks** → *Network servers*; *Network performance analysis*; • **Computer systems organization** → **Processors and memory architectures**;

KEYWORDS

NFV, NUMA, LLC, performance, optimization, measurements

ACM Reference Format:
Christian Sieber, Raphael Durner, Maximilian Ehm, Wolfgang Kellerer and Puneet Sharma. 2017. Towards Optimal Adaptation of NFV Packet Processing to Modern CPU Memory Architectures. In *Proceedings of CAN'17: Cloud-Assisted Networking Workshop (CAN'17).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3155921.3158429

CAN'17, December 12, 2017, Incheon, Republic of Korea

ACM ISBN 978-1-4503-5423-3/17/12…$15.00
https://doi.org/10.1145/3155921.3158429

1 INTRODUCTION

Network functions, such as routers, firewalls or load balancers, are commonly realized using integrated solutions which contain the hardware as well as the software in one box. In 2012, ETSI proposed Network Functions Virtualization (NFV) in a white paper [5]. NFV is designed to enable network functions using commodity servers and virtualization techniques. This promises an increase in flexibility as the Virtual Network Functions (VNFs) can be placed in the network as required by the operator. Additionally, off-the-shelf components promise cost reduction.

One main challenge is the performance and lack of performance guarantees of the NFV environment. In contrast to traditional solutions, hardware and software are no longer developed together, but the VNFs should run on commodity hardware. One issue is the high number of packets that have to be processed. For simple VNFs, like for example a load balancer, the complexity of processing one packet is very low. If a hash table is used, it can be as low as O(1). Because of that, other effects which can be usually neglected gain more importance. [11] shows that shuffling a packet processing workload between cores can reduce the packet rate significantly, as the cache locality is not guaranteed. Fast packet processing frameworks address this and other issues to provide high packet rates on commodity servers. In this work we utilize virtual network functions realized with the Dataplane Development Kit (DPDK). The DPDK is a set of libraries for Linux to facilitate fast packet-processing on common server hardware. It was first introduced by Intel in 2013 and since then has become a Linux Foundation Project with broadening vendor-support.

In this work we evaluate the memory architecture of modern CPUs with regard to packet processing. The contribution of this work is as follows. First, we introduce a measurement methodology and show measurement results quantifying the overheads associated with different VNF placements. Second, we propose a new metric called Network Efficiency Index (NEI) which captures the efficiency of a running VNF in terms of packet handling. Third, we outline an optimization scheme which maximizes the NEI for a set of VNFs. The formulation of the optimization is left for future work.

The paper is structured as follows: Section 2 introduces the evaluated Non-Uniform-Memory-Access (NUMA) architecture. Section 3 describes Methodology and Implementation of the measurements. Section 4 shows the measurement results and evaluates the findings. Section 5 introduces the metric and outlines a possible optimization scheme. Section 6 gives an overview of the related work in this field and finally Section 7 concludes the work and depicts the next steps.

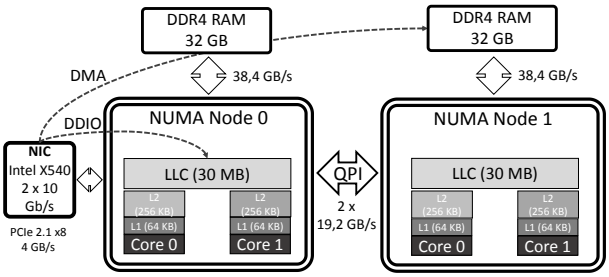


Figure 1: Non-Uniform-Memory-Access (NUMA) architecture with DDIO for the NIC-local NUMA node. DDIO allows direct access to the local LLC. DMA is used for packets destined to the remote node. Each Core has a L1/L2 cache, the LLC is shared between all cores on one node. A Quick-Path Interconnect (QPI) bus with 2x19.2 GBps connects the nodes.

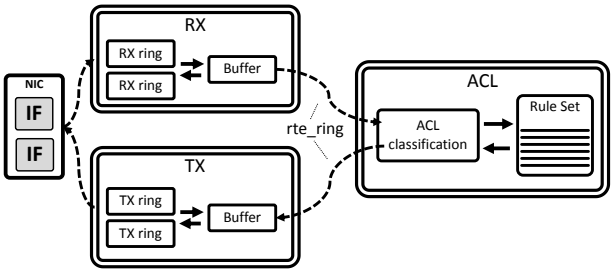


Figure 2: The function chain consisting of one RX, one ACL and one TX process. Each process runs exclusively on one core. The RX and TX processes move the packets between the respective buffers and rings. The ACL process checks each packets against the its rule set.

2 NUMA ARCHITECTURE

In the following we describe the NUMA architecture by example of the hardware used in our evaluation testbed. As the production of large chips is expensive, multi-processor systems are used for high end servers. Each socket houses one processor with multiple cores. The sockets are then connected using high speed bus connections.

The testbed consists of multiple Dell PowerEdge R530 servers with Intel Xeon E5-2650 v4 2.2GHz CPUs and Intel C610 chipsets. Figure 1 illustrates the NUMA cache hierarchy and interconnection technologies of the aforementioned testbed hardware. The servers have two CPUs, called NUMA nodes, connected via two bi-directional Intel QuickPath Interconnect (QPI) lanes with a bandwidth of 19,2 Gbyte/s each. An Intel Ethernet X540 NIC is connected to one NUMA node via PCI-express 2.1 x8. Each NUMA node contains 12 CPU cores with each 64 Kbyte of L1-cache and 256 Kbyte of L2-cache. A Last Level Cache (LLC) with a capacity of 30 Mbyte is shared among the CPU cores of a NUMA node. The caches are much faster than the system memory. Therefore, if multiple VNFs share the LLC, the performance is degraded compared to the VNF running alone on the processor due to interference effects. 32 Gbyte of DDR4 RAM capacity is attached to each node with a bandwidth of 38,4 Gbytes/s.

Intel Data Direct I/O (DDIO) allows NICs to copy packets straight to the LLC-cache of the NIC-local NUMA node, instead of doing the round-trip to the main memory and back to the LLC-cache when the processing application tries to read the packets. However, this is only possible for the NUMA node where the NIC is attached to, not for the remote NUMA node.

3 METHODOLOGY

In this section we discuss the measurement setup, the implementation of the VNFs and how the CPU load is measured.

The test environment consists of the load generation PC which runs DPDK-Pktgen Application and the Device-under-test (DUT), which is one of the servers described in Section 2 in detail. The PC generates packets with a limit of 2 Mpps and sends it to the DUT with a constant rate. At the DUT the packets are processed from one of the VNFs described in Section 3 and sent back to the PC. As we concentrate on CPU metrics in this work, no packets are received by the load generation machine, i.e. they are dropped by the NIC. Our study concentrates on CPU performance metrics, like core utilization and cache hit rates. In consequence all statistics are gathered directly on the DUT. For all cache related measurements we are using the Processor Counter Monitor (PCM) tool.

3.1 Minimal VNF

First we consider a minimal network function that only receives packets and does not do any processing or forwarding. Using this implementation we can show the overhead of the NUMA communication for packet processing without any side effects. The minimal VNF drops all packets after receiving.

3.2 Function Chain Implementation

The chain implementation consists of three separate entities as illustrated by Figure 2. A receiving, a sending and a packet classification process, each running exclusively on one CPU core following the DPDK best-practices. Two receiving rings, one for each physical NIC port, are connected to a software switch, which writes the packet pointers to a buffer. The interconnection between the three entities is implemented via a rte_ring data structure. A rte_ring is a lockless, fixed-size queue implementation provided by the DPDK. The ACL classification core matches the received packets against the loaded ACL rules and decides to either forward or drop the packets. The sending core moves the packets to be forwarded to the TX rings of the physical NIC ports.

3.3 Measuring CPU Load with Polling

One of the techniques that DPDK uses to increase the packet throughput is the change from an interrupt-based packet retrieval to a polling-based packet retrieval. Usually, if a packet arrives on the NIC, the CPU is interrupted and the packet is then copied and processed by the OS. DPDK does not use interrupts, instead it checks for packets at the NIC, processes these packets and then checks again for packets in an infinite loop. As a consequence, conventional CPU utilization tools do not work as the CPU is always fully utilized by the loop. Because of that we developed an algorithm shown in Algorithm 1 to evaluate the current CPU load. The main idea is to rely on the cycles reported by the CPU, as they are available without much overhead. During the main processing loop,

the cycles are read before and after the packet processing to determine the cycles needed for processing, i.e. the OPS. Additionally, the cycles needed for each loop are measured as reference counter REF. Both counters are reported regularly to the monitor. The CPU utilization can then be computed with the following formula:

$$CPU\ Utilization = \frac{OPS}{REF}$$

Using this definition of CPU utilization, a utilization of 100 % is reached with the maximum possible packet rate. As can be observed in the results plotted in Figure 3, the utilization is roughly linearly dependent on the packet rate.

4 PERFORMANCE IMPACTS

In the following we evaluate the influence of the memory architecture on the packet receiving performance. In the first and second part we evaluate the impact of the NUMA architecture with the minimal VNF and the function chain. The third part evaluates the performance degradation when using VNFs with a large working set, i.e. when the caches of the processor are exhausted.

4.1 NUMA Impact - Minimal VNF

In the first scenario, the receiver is placed on the same node as the NIC is attached to, denoted as node 0 in the following. In the second scenario, the receiver is placed on the remote NUMA node, in the following denoted as node 1. Figure 3 illustrates the receiver's core utilization for packet rates from 0 to 1 million packets per second for both scenarios. Confidence intervals are not visible as the measurements showed little variation.

The figure shows that the core utilization increases by 41 % (14.7 % utilization compared to 20.8 %) when the NIC is attached to a different NUMA node than the RX process. This is due to overhead which is required for transferring the packets first to the remote memory and afterwards to the processor cache. In case the NIC is attached to the same node, DDIO allows a direct transfer to the processor cache as shown in Figure 1.

Subsequently we take a look at the LLC hit ratio and memory read throughput for the two placements to confirm the source of the bottleneck described above. Figure 4 shows the hit ratio (4(a)) and the memory throughput (4(b)) for packet rates up to 1 million packets per second. For packet rates close to zero (200 packets per second) we observe a LLC hit ratio of 40 % for the node 0 placement

Algorithm 1 Measure CPU Utilization

```
OPS, REF = 0
cyc_last ← read CPU cycles from register
while True do
  cyc_before ← read CPU cycles from register
  REF + = cyc_before - cyc_last
  cyc_last ← cyc_before
  if packets received then
    process packets
    cyc_processed ← read CPU cycles from register
    OPS + = cyc_processed- cyc_before
  end if
end while
```

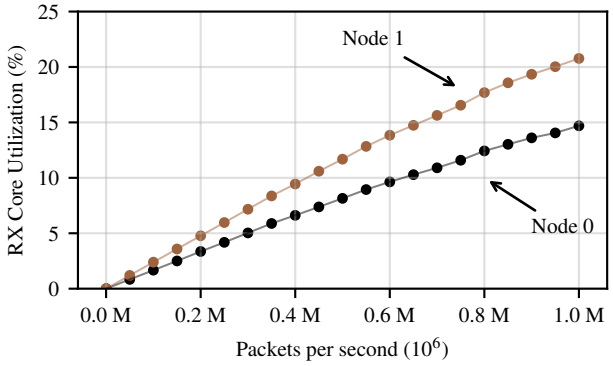
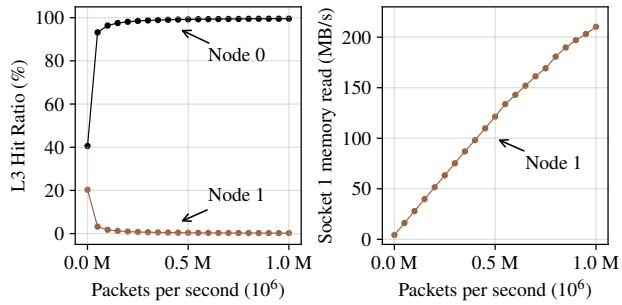


Figure 3: Simple Receiver: Utilization of the core for increasing packet rates and two different placements (node 0 and node 1). The NIC is attached to node 0. A 41 % increase in utilization is observed when the receiver is executed on node 1 (20.8 % utilization compared to 14.7 %), due to the overhead in fetching packets from the NIC through the QPI.



(a) LLC hit ratio (b) Memory read node 1

Figure 4: LLC cache hit ratio and memory read rate on node 1 for two placements of the minimal VNF for increasing packet rates up to 1 million packets per second. The figure confirms that a placement on the remote NUMA node without DDIO support results in cache misses and high main memory access rates.

and 20 % for the node 1 placement. For higher packet rates the hit ratio increases rapidly to about 100 % for the node 0 placement and 0 % for the node 1 placement. The unexpected hit ratios for low packet rates are due to the influence of the underlying operating system and measurement scripts. That influence diminishes with higher packet rates. Figure 4(b) gives the memory read throughput for the node 1 placement. The figure shows that the throughput increases linearly with the packet rate. The results confirm the previous statement that DDIO allows a direct transfer of the packets from the NIC to the LLC cache of the processor, which results in a 100 % cache hit ratio. In case of the placement on node 1, the packets are transferred via DMA to the main memory of the node first. The access by the receiver results in a cache miss for every packet and therefore the memory read throughput increases.

4.2 NUMA Impact - Function Chain

The influence of the NUMA placement on the performance of the function chain is described in the following. We place a chain of 3 elements on two possible locations, which results in 9 possible

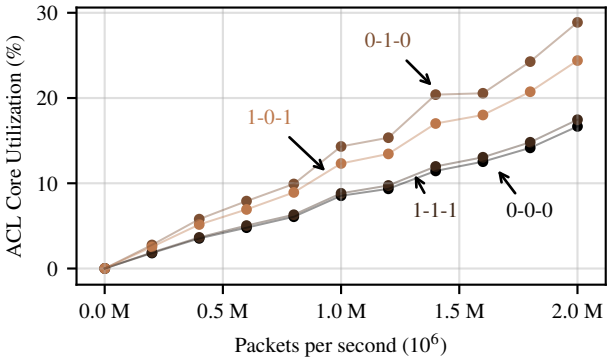


Figure 5: ACL core utilization for four different placements of the firewall function chain (RX-ACL-TX) depending on packet rates. Placement 0-0-0 represents the best case placement without remote NUMA access and 0-1-0 the worst case where remote NUMA memory has to be accessed. The results shows an increase in utilization for the worst case placement of roughly 73 % compared to the best case.

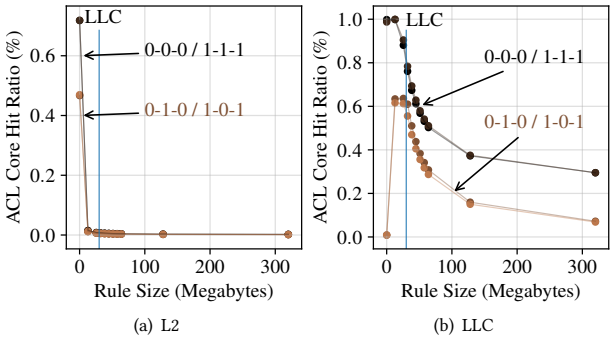


Figure 6: ACL core cache hit ratios of the L2 and LLC depending on the ACL size for the four different placements. As expected, the hit ratio of the small L2 cache falls fast to about 0 % for all four placements. For the LLC, we measure a hit ratio of 7 % for the worst case placements 0-1-0 and 0-1-0, where the packet is not yet in the LLC. For the best case placements 0-0-0 and 1-1-1, where the packet is already available in the LLC, we measure a hit ratio of of 30 %.

placements. The placements are denoted with RX-ACL-TX on the two NUMA nodes 0 and 1. E.g. 0-1-0 indicates a placement where the RX function is put on NUMA node 0, the ACL function on NUMA node 1 and the TX function is placed on NUMA node 0. The NIC is connected again to node 0. With placement 0-0-0 no packet copying between the NUMA nodes is necessary, therefore this placement is expected to be the best case. On the other hand, placement 0-1-0 is expected to be the worst case regarding the ACL utilization, as the packets first have to be copied to NUMA node 1 for ACL classification and then back to node 0 for transmitting.

Figure 5 illustrates the utilization of the ACL core for the four placements (0-0-0, 0-1-0, 1-0-1 and 1-1-1) of the function chain. The figure shows a linear increase of the core's utilization for increasing packet rates up to 2 million packets per second. For 2 Mpps, the measurements show an utilization of about 29 % for the worst case 0-1-0 and 17 % for the best case 0-0-0. Hence, there is a penalty of roughly 73 % regarding the CPU load between best case and worst case placement This means that NUMA-level copying caused by a non-optimized placement has severe performance impacts.

4.3 Impact of Cache Exhaustion

Next, we discuss the performance impact of cache exhaustion on the ability of a core to process packets. For this we keep the packet rate constant at 2 Mpps and increase the size of the ACL. We evaluate ACL sizes from 64 KB to 320 MB. For the LLC cache size of 30 MB, an ACL size of 320 MB results in a 10.6 times over-subscripted LLC.

Core utilization of the ACL function is caused by packet processing (copying packets, accessing headers, etc.) and additionally by accesses to the memory for the ACL. The performance penalty for accessing data in the memory largely depends on the locality of the data, i.e. if it is in a L1, L2 or LLC or if it is in DRAM, as the processing core has to wait for the data [4].

Figure 6 illustrates the ACL core cache hit ratios of the L2 and LLC depending on the ACL size for the four different placements 0-0-0, 0-1-0, 1-0-1 and 1-1-1. As expected, the hit ratio of the small L2 cache falls fast to about 0 % for all four placements. As the core

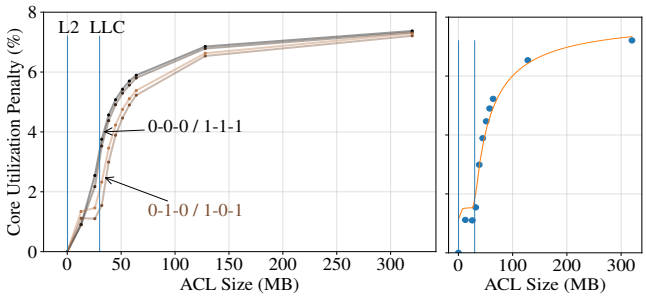


Figure 7: Impact of memory accesses on the core utilization of the ACL core. Increasing ACL size for a constant packet rate. L2 cache capacity is 256 KB and LLC cache capacity is 30 MB. The figure shows the additional utilization caused by memory access. The additional utilization follows for the placements 1-0-1 and 0-1-0 roughly the trend of a simple Average Memory Access Time (AMAT) model (right figure). The dots in the right figure are the measurement points of the 0-1-0 placement.

does not access the packet before the classification, the packet can not be available in the L2 cache. Furthermore, the chance that a specific ACL rule was accessed before decreases fast as the quotient between L2 size and ACL size gets very small.

For the LLC, we measure a hit ratio of 7 % for the worst case placements 0-1-0 and 0-1-0 where the packet is not yet in the LLC. For the best case placements 0-0-0 and 1-1-1, where the packet is already available in the LLC due to the TX process being placed on the same NUMA node, we measure a hit ratio of 30 %.

Figure 7(a) illustrates the ACL core utilization penalty for increasing ACL size for the four placements of the RX, TX and ACL cores. The two horizontal lines mark the capacities of the L2 and LLC. The figure shows that for the 0-0-0 placement, the utilization increases

about linearly for an ACL size between the L2 and LLC capacity. For the 0-1-0 and 1-0-1 placement, the penalty first jumps from 0 % to 1.2 % and subsequently stagnates until the LLC is exhausted. The stagnation is due to the fact that the 0-1-0 and 1-0-1 placements allow the ACL core to use the NUMA 1 node's LLC exclusively. Therefor, the impact of the LLC exhaustion is only visible for larger ACL sizes compared to the 0-0-0 and 1-1-1 placements where the LLC is shared between the RX, TX and ACL cores. After the LLC cache is exhausted, the utilization increase depends, in addition to the ACL size, on the cost in terms of time for accessing the memory hierarchy and on the cache hit ratios.

The behavior follows roughly the recursive Average Memory Access Time (AMAT) model from [6]. Equation (1) and Equation (2) define the AMAT model. H_x denotes the hit time, the time the processor needs to access the data on cache level x, if it is available in the cache. MR_x denotes the miss rate. We assume in our model that one ACL rule can be fetched by one cache access and that every packet triggers the same amount of cache accesses uniformly. Hence, we define MR_x as the chance of missing the cache based on the size of the cache S_x and size of ACL rule set S_{ACL}: $MR_x = max(0, 1 - \frac{S_x}{S_{ACL}})$.

$$AMAT = H_{L1} + MR_{L1} \cdot AMP_{L1} \tag{1}$$

with AMP_x as average miss penalty of cache level x:

$$AMP_x = H_{x+1} + MR_{x+1} \cdot AMP_{x+1} \tag{2}$$

The CPU load L_{CPU} for accessing data depending on the packet rate R_P can then be computed with Equation (3), where f_{CPU} is the CPU frequency.

$$L_{CPU} = AMAT \cdot \frac{R_P}{f_{CPU}} \tag{3}$$

Due to performance optimizations and pipelining in modern CPUs, the timings can not be named easily. We fitted the the cache timings to the measuring results using least squares method. Figure 7(b) illustrates the memory access penalty as a function of the ACL size S_{ACL} according to the AMAT model. The dots show the measurements with placement 0-1-0, the line denotes the model. The figure shows that the core utilization penalty follows roughly the AMAT model, with a small offset for smaller ACL sizes < 30 *MB*.

To summarize the findings we can see, that the additional CPU utilization caused by cache exhaustion is clearly visible but smaller than the NUMA penalty (roughly 15 % vs 7 %).

5 NETWORK EFFICIENCY INDEX

In the following we are using the results from the measurements to propose the Network Efficiency Index (NEI), a metric for quantifying the network efficiency of a deployed VNF. We describe how the metric can be used in a holistic optimization approach to maximize the performance of a NFV system and the influence of the traffic distribution on the NEI. Traffic distribution is here defined as the frequency of the 4-tuple packet headers (IP/port source and destination). The *NEI* of a specific *VNF* is defined as the ratio between the packets accessed by the VNF from the LLC and the total number of packets received by the VNF in a time interval, e.g. 1 second:

$$NEI_{VNF} = \frac{LLC\ hit\ on\ packets\ by\ VNF}{Total\ number\ of\ packets\ received\ by\ VNF}$$

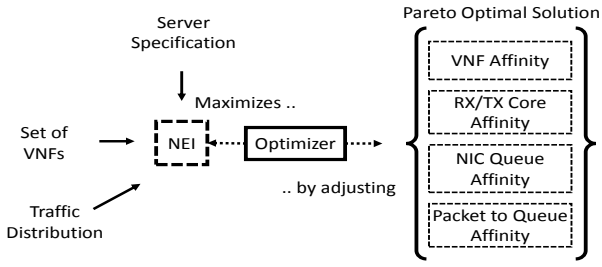


Figure 8: Illustration of how the Network Efficiency Index (NEI) can be used in an optimization to describe the cost of a placement of a VNF in terms of NUMA and cache overhead. This overhead depends on the NUMA node and core the VNF is placed on, the relative location of the RX and TX cores and the NIC queue packet distribution.

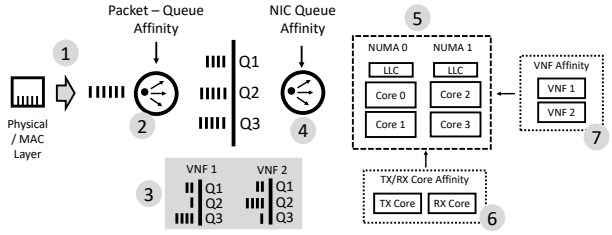


Figure 9: Detailed illustration of the influence of the traffic distribution, packet queue affinity and NIC queue affinity. At (1) Ethernet frames arrive at the physical layer. At (2) the packets are distributed by the NIC to internal queues. (3) shows a possible queue placement of packets for each VNF. (4), (5) and (6) illustrate the challenge of placing the TX/RX cores and the VNFs.

In the best case, the NEI of a VNF equals 1.0 where all packets are received from the LLC. In the worst case, the NEI of a VNF equals 0.0 where all packets have to be fetched from the local or remote main memory first.

Figure 8 illustrates how the NEI can be used in an optimization to describe the cost of a placement of a VNF. The NEI depends a) on the server specification, i.e. number of NUMA nodes, LLC architecture and how the NIC is connected to the NUMA nodes, b) the traffic distribution, i.e. which packets are placed in which NIC queues and how the NIC queues are connected to the cores and c) on the number and placement of the VNFs. The objective of the optimization is to maximize the NEI of all VNFs in terms of a higher level objective defined by the operator, e.g. max-min-fairness or overall performance. The optimization chooses a) the VNF affinity, i.e. which core executes which VNF, b) the RX/TX core affinity, i.e. which cores are designated for the receive and transmit loops, c) the NIC queue to TX/RX core affinity, i.e. which TX and RX cores handle which NIC queues and d) the packet to NIC queue affinity. We now discuss (d), the packet to NIC queue affinity in detail.

Figure 9 shows the path of received network packets through the system. In (1) Ethernet frames arrive at the physical layer in an order defined by the sending system and with a traffic distribution as defined by the routing policy of the network. The packets are then distributed by the NIC to internal queues (2) based on either rules,

e.g. in the case of Intel FlowDirector, or based on a hash algorithm like the Toeplitz hash[12]. The challenging part is how to fine-tune the hash or how to set the rules such that packet distribution per VNF (3) is optimal with respect to the NIC queue affinity (4), the architecture (5), the core affinity (6) and the VNF affinity (7).

The advantage of the approach can be summarized as follows. The end-to-end optimization considers the whole packet processing path to improve data locality and therefore increases the efficiency and reduces network latency of the system. Furthermore, no human intervention is necessary and the optimization can be performed autonomously. Also, the optimization can react on-demand to changes in the hardware and/or software environment. Finally, to implement this into existing NFV environments, no extensions to the system are required and all interfaces and metrics are already available.

The formulation of the optimization to maximize the NEI in a running system is left for future work.

6 RELATED WORK

Some works that study the performance of NFV with focus on memory and data locality bottlenecks already exist: Authors of [13] study the performance of DPDK in conjunction with single-root input/output virtualization (SR-IOV). SR-IOV is a passthrough I/O technology which enables VMs to access the NIC hardware directly. The work shows the sensibility of the performance to the NUMA placement, although the focus of the study is on the performance impact of the number of VMs.

Authors of [8] use DPDK for developing so called Mikro VNFs, which locally process the packets of user VMs. The evalution shows that the DPDK based approach has clear performance advantages. In contrast to our work the authors focus on introducing a new architecture for VNFs in a general purpose cloud.

Authors of [10] study the usage of DPDK for very high packet rates. Their results show that a data rate of 100Gb/s can be achieved using a single server. For multi-threaded packet processing multi-queue NICs are used. Authors evaluate the effects of different queue to core mapping strategies. They evaluate their system in terms of packet drop rate. Cerrato et. al [2] study the performance of VNFs using DPDK and different memory architectures. Their results show that a DPDK-based packet processing system with a high number (>100) of tiny network functions can deliver satisfactory throughput performance, although the experienced delay becomes high. Authors of [7] developed a NUMA aware thread scheduling approach, that reduces the slowdown caused by the NUMA architecture. The authors propose a performance slowdown index based on the inter-socket overhead caused by L3 cache misses.

Banerjee et. al [1] and Dobrescu et al. [3] show the performance impacts on VMs when comparing the VM on the same NUMA node as the NIC versus the remote node. Their results show a high number of cache misses when placing the VM on the remote node. They also provide an approach for determining a NUMA aware placement in virtualized environments. In [9], Kulkarni et al. introduce NFVnice, a framework for scheduling NFs on a server. They look at the problem of how computing resources can be allocated to NFs using rate-cost proportional fair shares.

In our work we concentrate on the performance bottlenecks in the CPU architecture and therefore do not use any virtualization techniques to avoid side effects. Furthermore we introduce a model that aims to quantify the network efficiency of NFV placements, with respect to architectural specifics.

7 CONCLUSION

With the move from dedicated hardware to multi-purpose hardware for packet processing applications, performance aspects of this hardware are getting increasingly important. In this work we outline an optimization scheme for VNF on-server placement and evaluate the effects of the memory architecture to the packet processing performance. The results show that copying packets between the NUMA nodes increases the CPU load drastically and should be avoided if possible. The measurements using an increasing ACL size in memory show that the effects of CPU cache exhaustion should be considered when designing VNFs. Additionally, we show that the CPU cycles needed for memory access follow the AMAT model. Overall performance penalty of copying between NUMA nodes is bigger than accessing the memory at the local socket.

In the future we want to use the measurements results to provide a complete formulation of the proposed optimization scheme and extend our measurements to additional architectures.

ACKNOWLEDGMENTS

This work has been partially carried out at the Hewlett Packard Labs, Palo Alto as part of Christian's internship. It was also partly funded by the German Research Foundation (DFG) under the grant number KE1863/6-1 and by the German Federal Ministry for Research and Education (BMBF) in the framework of the SARDINE project (Project ID 16KIS0261). The authors alone are responsible for the content of the paper.

REFERENCES

[1] Amitabha Banerjee, Rishi Mehta, and Zach Shen. 2015. NUMA Aware I/O in Virtualized Systems. In *IEEE 23rd Annual Symposium on High-Performance Interconnects (HOTI)*. IEEE.

[2] Ivano Cerrato, Mauro Annarumma, and Fulvio Risso. 2014. Supporting fine-grained network functions through Intel DPDK. In *Third European Workshop on Software Defined Networks (EWSDN)*. IEEE.

[3] Mihai Dobrescu, Katerina Argyraki, and Sylvia Ratnasamy. 2012. Toward Predictable Performance in Software Packet-Processing Platforms. In *Proceedings of NSDI 12*. USENIX.

[4] Dr. David Levinthal PhD. - Intel. 2009. *Performance Analysis Guide for Intel Core i7 Processor and Intel Xeon 5500 processors*.

[5] ETSI. 2012. Network Functions Virtualisation - Introductory White Paper. *SDN and OpenFlow World Congress* (2012).

[6] John L. Hennessy and David A. Patterson. 2012. *Computer Architecture a Quantitative Approach Fifth Edition*.

[7] Yang Hu and Tao Li. 2016. Towards efficient server architecture for virtualized network function deployment: Implications and implementations. In *49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*.

[8] Ryota KAWASHIMA and Hiroshi MATSUO. 2017. A Generic and Efficient Local Service Function Chaining Framework for User VM-dedicated Micro-VNFs. *IEICE Transactions on Communications* (2017).

[9] Sameer G Kulkarni, Wei Zhang, Jinho Hwang, et al. 2017. NFVnice: Dynamic Backpressure and Scheduling for NFV Service Chains. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication, SIGCOMM '17*. ACM.

[10] Peilong Li, Xiaoban Wu, Yongyi Ran, and Yan Luo. 2017. Designing Virtual Network Functions for 100 GbE Network Using Multicore Processors. (2017).

[11] Marek Majkowski. 2015. Cloudflare Blog: How to receive a million packets per second. (2015). https://blog.cloudflare.com/how-to-receive-a-million-packets/

[12] Bryan Veal and Annie Foong. 2007. Performance scalability of a multi-core web server. In *Proceedings of the 3rd ACM/IEEE Symposium on Architecture for networking and communications systems*. ACM.

[13] Chengwei Wang, Oliver Spatscheck, and Vijay et al. Gopalakrishnan. 2016. Toward High-Performance and Scalable Network Functions Virtualization. *IEEE Internet Computing* (2016).