Universal Node: Towards a High-Performance NFV Environment

Gergely Pongrácz
TrafficLab, Ericsson Research, Budapest, Hungary
gergely.pongracz@ericsson.com
(invited paper)

ABSTRACT
Software-Defined Networking (SDN) and Network Function Virtualization (NFV) both promises the vision of more flexible and manageable networks, but they also require certain level of programmability in the data plane.

Current industry insight holds that programmable processors (both CPUs and NPUs) are of lower performance than their hard-coded counterparts, such as Ethernet chips. We show some results that this does not seem to be true.

Today, OpenFlow (OF) is probably the most suitable protocol to achieve network programmability and together with virtual network functions (VNFs) running in either virtual machines or containers there is no limitation for the implementable features. This combination forms the basis of the Universal Node (UN) that aims to be the backbone of future NFV based networks, since it can run "classical" VNFs, but it can also run VNFs that are implemented right inside the optimized OF pipeline. In this paper we focus on the latter: we show that OF itself can be used for implementing various network functions, such as routing, and the performance we can get can match the non-OF implementations of the given network function once some bottlenecks are removed. Besides the 'usual' optimizations like using Intel DPDK for faster I/O, we target two bottlenecks: the inefficient lookup and the slow action handling.

With these optimizations our UN prototype can do 100 Gbps (gigabit per second) even with small packets on one Intel blade, reaching a noticeable 110 Mpps packet speed with 64 byte long packets.

Keywords
Network Function Virtualization (NFV), Software Defined Networking (SDN), Data Plane, Power, Performance, Network Processor, OpenFlow

1. INTRODUCTION
Software Defined Networking (SDN) and Network Function Virtualization (NFV) promise to offer of freedom for operators to refactor their heterogeneous and management-intensive networks and turn them to a more centralized, more manageable one where workloads are easier to scale and service deployment is much faster than today.

Such a mode of operation has a price: the data plane must expose a rich set of packet processing features, for the controllers to program. Such programmability in the data plane is widely regarded as expensive compared to simple purpose-built ASICs, such as Ethernet chips.

The dominating view in the industry thus contrasts programmability with high performance [13]. In this paper we question this view by showing some modelling results and also by describing a high-performance OpenFlow switch prototype and showing its performance results.

First we argue that a programmable network processor can achieve throughput in the same ballpark as todays Ethernet chips – if the balance between its major components, such as processing cores, on-chip, memory, I/O and memory controllers fits the use case.

Then we demonstrate that a software-based OpenFlow switch can reach hardware-router like performance when some care is taken in programming an Intel CPU the right way.

The paper is organized as follows. In section 2 we briefly overview the current chip landscape. Section 3 explains the modelling work and its results. In section 4 we discuss the OpenFlow pipeline and some optimization ideas for software switches. In section 5 we present the measurement scenario and the results, while section 6 contains the summary.

2. HARDWARE SURVEY
This section contains the result of a short hardware survey based on a previous paper of the author [18]. The additional item over this previous paper is the x86 based model and a new use case.

Table 1 summarizes the chips we surveyed (relying on product briefs and tech. specifications [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [14]) showing their power consumption per 10G port and total throughput in bits and packets. There are numerous other examples (e.g. EzChip NPS, Broadcom Trident and Arad), but we do not see that they would change the picture significantly. Looking at the table one could come to a conclusion that there is a

significant performance difference between purpose-built hardware and programmable devices.

Table 1. DP hardware at first glance – values may not represent fair comparison (type PP means programmable pipeline)

Chip name (nm)	Gbps	Mpps	Power / 10G	Type
Ericsson SNP 4000 (45)	200	300	4W	NPU
Netronome NFP-6 (22)	200	300	2.5W	NPU
Cavium Octeon III (28)	100	?	5W	NPU
Tilera Gx8036 (40)	40	60	6.25W	NPU
Intel X-E5 4650 DPDK (32)	50	80	24W	CPU
EzChip NP4 (55)	100	180	3.5W	PP
Marvell Xelerated AX (65)	100	150	?	PP
EzChip NP5 (28)	200	?	3W	PP
Intel FM6372 (65)	720	1080	1W	Switch
BCM56840 (40)	640	?	?	Switch
Marvell Lion 2 (40)	960	720	0.5W	Switch

In the following section we argue that this is misleading. The values above cannot be directly compared because they reflect different use cases. The real difference is around 20-25% instead of 10x.

The KPI we use will be the packet processing efficiency of the chip measured in million packets per second per Watt (Mpps/Watt). We also express the above KPI in power consumption per a 10G port of the chip, measured in Watts/10Gbps.

3. MODELLING AND RESULTS

We defined a simple model for programmable NPUs. The model contains the basic building blocks of a programmable chip, such as:

- I/O: the number of 10G lanes, either towards the external world (e.g. Ethernet port), to the fabric/backplane (e.g. Interlaken) or to the system (e.g. board processor) [parameter = number]
- Accelerators: in this model we have optimized lookup engines only [number]
- Processing units: Packet processing cores [number, clock frequency]
- On-chip memory: integrated memory (SRAM, eDRAM, TCAM) [number, size, transactions per second, deployment]
- External resource control: in this model we have external memory controllers (MCT) only with RLDRAM or DDR SDRAM [number, size, transactions per second, deployment]

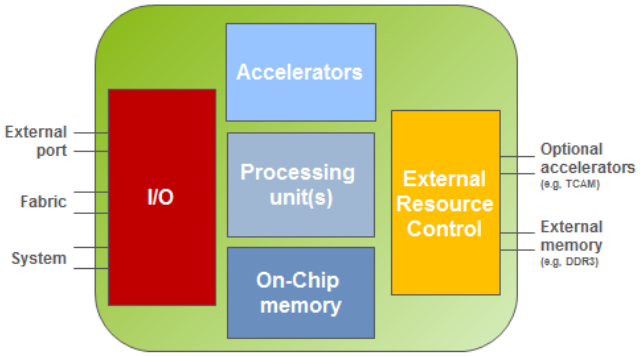

Figure 1
Model components

3.1 Chip design constraints

There are constraints both for the number of internal units (cores, memory, and accelerators) and the number of external units the chip can handle.

The package area of chips has an upper limit due to manufacturing costs and signal-propagation delay. These days this practical limit is around 60x60 mm. Both internal and external units are limited by the available area of the chip.

The first key balance in chip design is between internal memory and the number of processing units.

The second key balance is between external memory bandwidth and I/O bandwidth.

3.2 Model limitations

In the model we make certain simplifications that do not impact our conclusions. We don't take into account ingress and egress hardware acceleration. Packets arriving on an I/O port are transferred directly to an internal memory bank. Packets sent to the external world are transferred from the internal memory either directly to an I/O port or to an external memory bank in case of queuing.

During memory access calculations there is a difference between SRAM and DRAM. In case of SRAM, linear model is used with 4 bytes per core clock. In case of DRAM fixed transaction rate is used. Although DRAM access has a fixed latency + linear read/write time, since the latency dominates the total time in cases when we are below the length of the cache line, we simulate the total access time with a single number. When several cache lines are to be read, we multiply this with the number of cache lines. This way it is an overestimation of the real cost of a long burst.

3.3 Model parameters for NPU

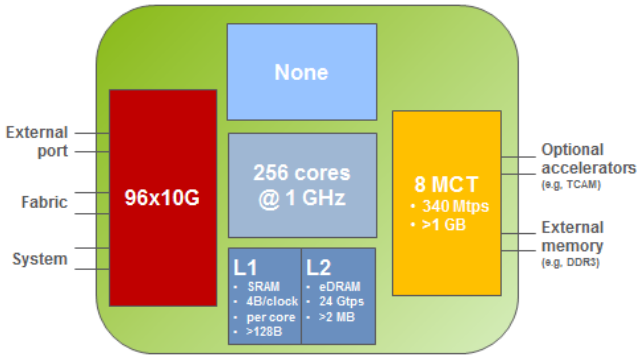

Figure 2
Model parameters using state of the art NPU numbers

The figure above shows a hypothetical network processor with a certain choice of the two balances discussed above.

Besides NPUs we also have to consider using classic CPUs, such as the Intel x86 family.

3.4 Model parameters for CPU

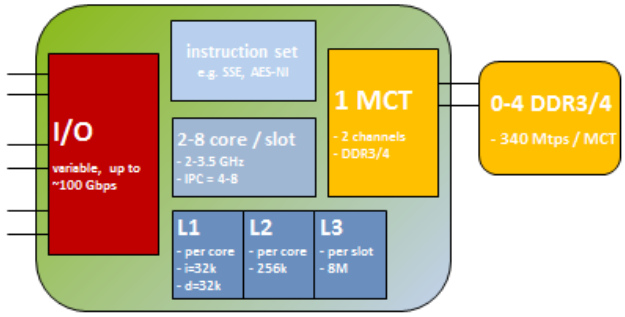

Figure 3
Model parameters using basic CPU numbers

For example our test system with Xeon 2630 CPU contains 2 slots, 160 Gbps I/O, 2x6 CPU cores and 4 DDR3 memory channels.

3.5 Ethernet PBB use case

This use case is a real challenge for CPUs and NPUs: lookup from multiple tables, encapsulation, header manipulation – so it is far from the "look at the destination and select the port" simplicity, yet is supported with line rate by state of the art Ethernet switches like the Marvel Lion 2/3 or the Intel Alta.

For a PBB switch roughly the following steps are required:

1. Read Ethernet frame from I/O, store header in L1, payload in L2 or L3 memory
2. Parse fixed parameters from header: physical ingress port, VLAN ID, DMAC
3. Find extended VLAN {source physical port, VLAN ID → eVLAN}
4. MAC lookup and learning {eVLAN, DMAC → B-DMAC, B-SMAC, egress physical port, flags}
5. Encapsulate: create new header, fill values – no further lookup is needed
6. Send frame to I/O

Altogether the following resources are required to process an average packet (details are in the HotSDN paper)

- 104 clock cycles
- 25 L2 or L3 memory operations
- 1 external memory operation

3.6 Results, NPU

Using the model parameters the following packet processing capacity can be calculated for the different system components in case of using the selected NPU:

Cores + L1 mem: 2462 Mpps

L2 memory: 960 Mpps

Ext. memory: 2720 Mpps

So the packet processor described in the model could process around 1 Gpps with the L2 memory as the bottleneck. Note that this is valid for an average packet size of 750 byte. In this case the supported bandwidth would be 6 Tbps.

With the usual worst case 64 byte packet length – since L2 memory is not much used in that case – the modeled processor's pps performance climbs up to 2462 Mpps with the processing cores (and the L1 memory) being the new bottleneck. This results in a bandwidth of 1260 Gbps.

That means that the given packet processor can forward roughly 1 Tbps with all possible packet length distribution, so it could handle 96 x 10 Gbps ports.

3.7 Results, CPU

The following packet processing capacity can be calculated for our test CPU:

Cores + L1/L2: 1006 Mpps (IPC = 4)

L3 cache: 280 Mpps

Ext. memory: 680 Mpps

This is valid with 64 byte long packets, with larger packets the L3 cache is more loaded, so the packet rate is smaller. To obtain per slot results the above values simply have to be divided by 2.

Table 2 shows the theoretical results for some more packet processors. As it is visible the efficiency difference between purpose-built hardware and programmable devices is much smaller when we use the same functionality on both. The results are theoretical because today most of the programmable chips are designed for more complex tasks meaning that for reaching the Mpps value in the table they would need more I/O ports.

Table 2. The (almost) real picture: comparison of the same (PBB) use case

Chip name	Power	Mpps	Mpps / Watt	Type
Ericsson SNP 4000	80W	1080	13.5	NPU
Netronome NFP-6	50W	493	9.9	NPU
Cavium Octeon III	50W	480	9.6	NPU
Tilera Gx8036	25W	180	7.2	NPU
Intel Xeon 2630 DPDK	80W	140	1.75	CPU
EzChip NP4	35W	333	9.5	PP
Intel FM6372	80W	1080	13.5	Switch
Marvell Lion 2	45W	720	16	Switch

Current tests show that the Intel Atom family has great potential when it comes to Mpps / Watt measurements. While the performance is roughly 50% compared to Xeon its power consumption is 10-15% only.

Figure 3 shows the main result obtained by our HotSDN paper in 2013. Note that use case complexity (axis X) is not a continuous axis, while the efficiency axis (axis Y) uses logarithmic scale.

Purpose-built hardware solutions have around 20-30% advantage, but are limited to simpler scenarios. Generic, non-optimized CPUs, such as Intel x86 have clear disadvantage in simpler, more packet processing specific use-cases [10], although our recent measurements show that low-power CPUs can be relatively close to NPUs. On the other hand generic CPUs can outperform NPUs when complex instructions and floating point arithmetic is used [16]

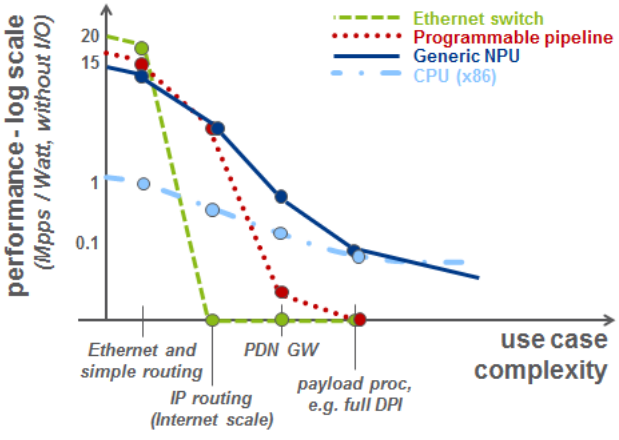

Figure 4
Chip efficiency vs. use case complexity

4. OPENFLOW OPTIMIZATIONS

In this section we describe our OpenFlow prototype which is developed as a proof of concept work aiming to show that the Universal Node (UN) concept can reach high performance while maintaining the flexibility we have when we use OF.

4.1 The OpenFlow pipeline

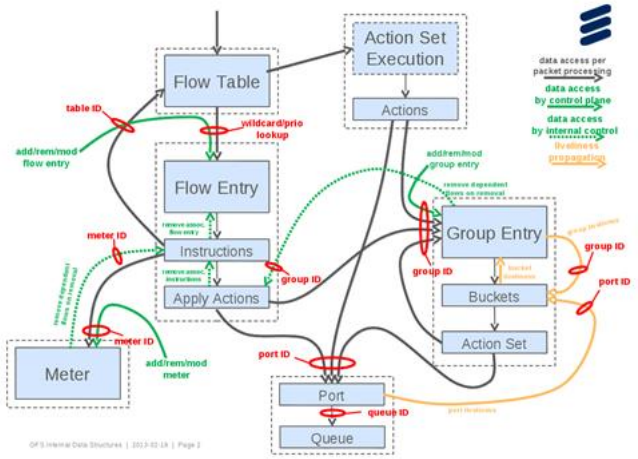

Figure 5
Generic OpenFlow pipeline

In this section, the packet processing pipeline of OpenFlow 1.3 is discussed generally. To understand the way an OpenFlow switch handles data packets, consider figure 5. As it can be observed, **flow entries** are grouped into **flow tables**, and packet is first checked against the **flow entries** in table 0.

First, the flow entry with the highest priority is checked, the fields of the data packet are compared with the **match fields** of that entry, and if they fit, some instructions are executed. If not, the entry with the second greatest priority is taken and so on until some fitting rule is found. If no rule fits, the packet is dropped, but this default behavior can be redefined.

When the matching entry is selected, its **instructions** are executed. There are many kinds of instructions, please note that figure 5 depicts only the most important ones.

Apply actions instruction contains a list of actions, which must be executed on the packet immediately in exactly the same order, as they were put into the list. A single **action** in OpenFlow is an entity describing an operation.

In contrast to apply actions instruction, with **write actions instruction**, an **action set** can be manipulated. Actions in the action set are not executed immediately, but only when there is no more instructions in the flow entry.

Meter instructions are used to check the packet against some predefined **meter**. Depending on the result of this check, packets can either be dropped, be put into different

queues at the ports (see later), or their DSCP field can be changed.

As previously mentioned, there are **goto table instructions**. A goto table instruction is always the last instruction in a flow entry, and it causes the continuation of the packet processing in a new table. If there is no goto table instruction, processing continues with those actions in the action set that were added by previous write actions instructions.

Sometimes, it is also beneficial to collect actions into **groups**, and refer them together with a single action called group action.

In the end, if the packet is not dropped, packet processing must get to an output port action, which describes the **port** the packet needs to be forwarded on. It is possible to define multiple **queues** for each of the ports, and the proper queue can also be selected by a set queue action.

4.2 Runtime code generation

For improving the performance of the OpenFlow pipeline, we have applied a template-based code generator, i.e. some code pieces are generated with (optionally) holes in them for constant values, and these pieces are linked together in runtime.

The rules described by OF use a relatively small, well defined set of match functions and actions, which can be precompiled as templates. The advantage of template-based code generation is its speed, which is a crucial point when a huge amount of OF rules are needed to be updated.

In general, we used the optimization techniques below. Although we used x86 processors in 64 bit mode, similar optimization mechanisms can be applied for all current architectures.

- **Eliminating function call overhead**: Function call overhead gets especially significant for small functions, which is exactly the case for checking OF match fields or executing OF actions. This overhead can be removed, if these small functions are compiled together.
- **Eliminating conditional branch instructions**: When the CPU fails to predict a branch, the pipeline gets empty. So it is useful to compile the code in runtime without the checks and branches which are not needed.
- **Using "folded in" constants**: If the value is added as a constant to the code, it is loaded together with the instruction and no additional time is needed. Although using the instruction cache for storing data may result longer instruction in some cases, our OF rules need typically only short code, so this extra burden (when exist) is not significant.

The above optimizations can be used for parsing, for match field optimization and for instruction/action optimization too.

4.3 Intel DPDK

Intel's Data Plane Development Kit (DPDK) is a toolset that tries to overcome the performance limitations of Intel CPUs when it comes to packet processing. It can be seen as bare metal operation mode on Intel CPUs. There are the following main features:

- **Poll mode driver:** To avoid using interrupts and context switchings, DPDK uses poll mode driver (PMD) where the userspace process polls the selected ingress queues and processes packets in the same context. Besides minimizing context switching overhead memory copies are also minimized.
- **Direct Cache Access:** When enabling Direct Cache Access (DCA, also called DDIO) the packets are directly copied into the last level (in this case L3) cache of the selected processor, this way avoiding high-latency external DRAM.
- **Huge pages:** Huge pages are useful for reducing misses in data translation lookaside buffers (D-TLB)

Besides DPDK there are other initiatives as well to tackle with these issues, such as netmap.

5. MEASUREMENTS

To prove the performance of the implemented solution we made measurements with the setup seen in figure 6.

5.1 Measurement setup

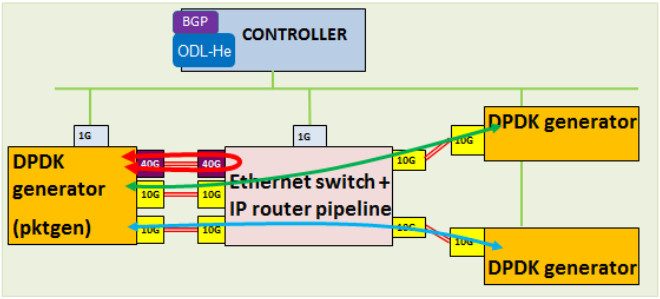

Figure 6
Measurement setup

The load generators were running the Intel DPDK based PktGen application which is capable of loading 10G interface with 64 byte long packets using 1 core.

In the testbed an SDN controller is also present, but it does not have influence on our current measurements.

The workstations were Supermicro workstations with the following configuration:

- Intel Xeon E5-2630 (Sandy Bridge) processors, each having 6 cores running at 2.3 GHz
- 1x Intel 710 (dual 40 GbE) and 4x Intel 82599EB (dual 10GbE) based NICs

Direct NIC port – CPU core mapping was used in our case. If required the NIC would be capable of basic load sharing.

5.2 Scenario (routing) details

In the measurements the system was configured using a real forwarding information base (FIB) table that was downloaded from an access router in the Hungarian Internet backbone (hbone).

The prefix length distribution was quite uneven, more than 50% of the prefixes was /24 prefix, and the vast majority was between /16 and /24. The statistics can be seen in Figure \ref{fig:fib-stat}. The FIB contains more than 400k entries (410454) and the number of next hops is 194.

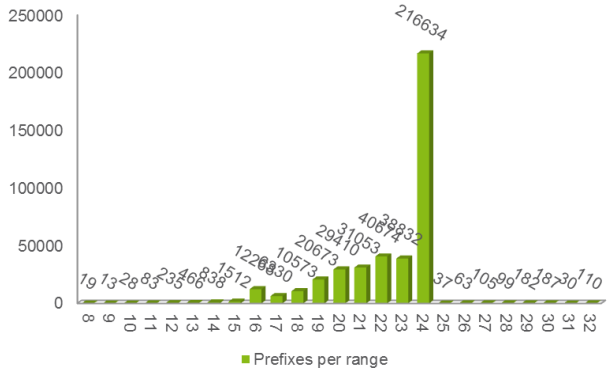

Figure 7
Route statistics

To simulate a real router and also not to waste memory space the OpenFlow pipeline was designed as shown in figure 8.

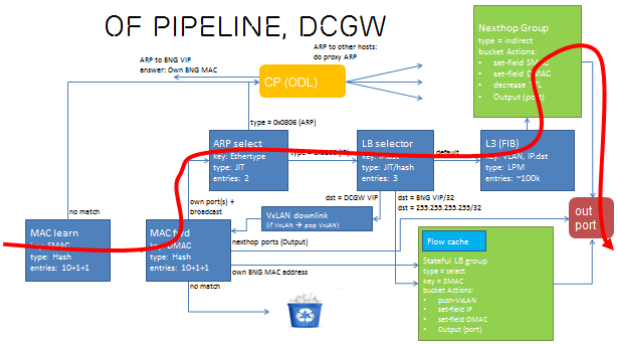

Figure 8
OpenFlow pipeline setup, DC gateway / router

The router pipeline contains 5 tables and 1 group. The MAC tables contain a few dozen rules, while the selector tables contain a few rules only. The L3 table contains the entire routing table (~400k routes), while the nexthop group contains 193 indirect groups.

5.3 Measurement results

Measurements were done for two use cases. Besides the 5 table + 1 group based IPv4 forwarding case we also tested a "lightweight virtual BNG" use case which contains 5-7 tables, 1 group and 1 meter.

Note that in the vBNG case the measurements were done on 10 Gbps interfaces, while in the routing case the 40 Gbps interface was used. Also note that these are per core results, so only 1 x86 core was used for processing the traffic.

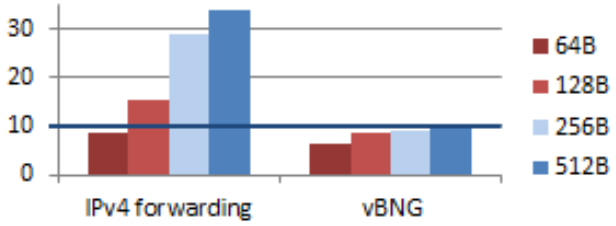

Figure 9
Per core results of 2 use cases (Gbps)

Scalability can be seen on figure 10.

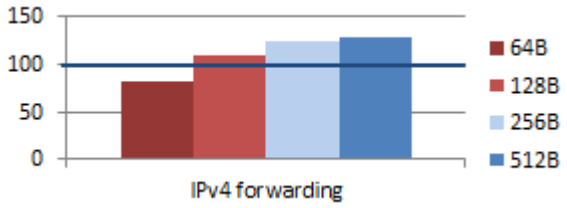

Figure 10
Scalability results on 10 cores (Gbps)

Note that in this case the PCIe interface became a limiting factor for the dual port cards and also the interface limit was reached for the 10GbE cards even at 128B packet size.

Except of that effect scalability is almost linear: while 1 core could handle ~9 Gbps traffic with 64B, 10 cores handle ~80 Gbps.

6. SUMMARY

We compared the relative performance of programmable CPUs, NPUs and Ethernet chips using an example use case. After observing that the difference is much lower than what we would think initially, we made a prototype to prove that the theoretical performance can be reached on a programmable device. It was shown that 100Gbps – which is the practical I/O limit of a recent x86 server – can be reached relatively easily even when the use case is slightly more complicated than simple switching.

Programmability alone, in other words, does not incur high performance penalty: there is little to gain by hardcoding (parts of) the use case in the chip design.

7. REFERENCES

[1] Marvell Prestera 98CX8297 (a.k.a. Lion 2) – Next Generation Switch Fabric for Modular Platforms – Product Brief

[2] FocalPoint FM6000 Series – Data Center CEE/DCB Switch Chip Family – Product Brief

[3] Intel® FM6000 Ethernet Switches 72-Port 10G Ethernet L2/L3/L4 Chip – Users Guide

[4] BCM56840 Series – High-Capacity StrataXGS® Trident Ethernet Switch Series with Integrated 10G Serial PHY

[5] EzChip NP-4 100-Gigabit Network Processor for Carrier Ethernet Applications – Product Brief

[6] Roy Rubenstein: Processor developers respond to new demands on telecom carriers and content providers – tech article, newelectronics online news portal

[7] Marvell Xelerated AX Family of Programmable Ethernet Switches – product brief

[8] Netronome Goes With the Flow – NFP-6xxx Transcends Network-Processing Categories – Linley Microprocessor Report

[9] TILE-Gx8036™ Processor – Specification Brief

[10] Impressive Packet Processing Performance Enables Greater Workload Consolidation – Packet Processing on Intel Architecture – Solution Brief

[11] Chandrakant D. Patel, Amip J. Shah: Cost Model for Planning, Development and Operation of a Data Center

[12] R. A. Philpott, J. S. Humble, R. A. Kertis, K. E. Fritz, B. K. Gilbert, E. S. Daniel: A 20Gb/s SerDes Transmitter with Adjustable Source Impedance and 4-tap Feed-Forward Equalization in 65nm Bulk CMOS

[13] Recep Ozdag: Intel® Ethernet Switch FM6000 Series – Software Defined Networking

[14] Ericsson strengthens its 4th generation IP portfolio – press release of SNP 4000

[15] Cortex™-M3 Technical Reference Manual – Processor instruction timings

[16] Géza Szabó, János Szüle, Bruno Lins, Zoltán Turányi, Gergely Pongrácz, Djamel Sadok, Stenio Fernandes: Capturing the Real Influencing Factors of Traffic for Accurate Traffic Identification

[17] Ulrich Drepper: What Every Programmer Should Know About Memory

[18] Pongrácz, G., Molnár, L., Kis, Z. L., and Turányi, Z. Cheap silicon: A myth or reality? Picking the right data plane hardware for software defined networking. In Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (New York, NY, USA, 2013), HotSDN '13, ACM, pp. 103–108.