Efficient serving of VPN endpoints on COTS server hardware

Daniel Raumer, Sebastian Gallenmüller, Paul Emmerich, Lukas Märdian, and Georg Carle
Technical University of Munich, Department of Informatics, Chair of Network Architectures and Services
{raumer|gallenmu|emmericp|maerdian|carle}@in.tum.de

***Abstract*—Of late an increasing amount of functionality in computer networks is provided by commodity x86 hardware wherein the CPU is the main bottleneck. Relieving the CPU from a portion of its computational stress leads to a lowered number of cycles spent on each single packet. Subsequently, servers are able to deal with millions of packets per second. We show a case study in which we used the cryptographic offloading functionality of commodity NICs to build a VPN IPsec gateway on an x86 server, where we required only one CPU core to serve 10 GbE line rate. The source code of the NIC-accelerated VPN gateway in our case study is publicly available. Our case study shows the trade-offs between manifold software- and high performance offloading hardware-provided functionality.**

I. Introduction

In today's network infrastructure, an increasing number of devices perform actions on commodity hardware, in software on a general purpose CPU. Although dedicated networking devices are still required where high performance is crucial, commodity hardware catches up in terms of throughput in bytes or packets per second and latency. Classical networking devices are being priced out of the market by masses of low-cost commodity hardware. Benefits of commodity hardware are manifold: functionality comes from software routines which are extensible and easy to update at scale. Functionality is no longer directly related to a certain hardware vendor. To increase the performance of software-based solutions, networking equipment is accelerated by parallelization across multiple systems, multi-core architectures, and components that help to offload frequent workloads from the CPU.

In this paper, we show an example of NIC offloading and how it helps commercial off-the-shelf (COTS) servers to cope with the performance provided by high-end networking hardware, at low prices. We focus on NICs equipped with features to help the CPU by receiving and transmitting encrypted packets which therefore saves CPU cycles for other tasks. We discuss how modern NICs speed up the systems by accelerating packet reception and transmission in Section II. In Section III, we present the state of the art and its performance in VPN handling. Section IV contains our case study in which we implemented a NIC-accelerated VPN endpoint. We analyze the performance of our implementation in Section V and conclude with finalizing remarks, in Section VI.

II. NIC and server performance

Traditionally, the upper layers of packet processing are implemented as part of the operating system and therefore executed on the CPU. With modern NICs performing tasks associated with layer 3 and 4 (IP/TCP), the workload moves from the OS to the NIC hardware. In the following, we will use the Linux operating system as a basis of our argumentation.

Cycles spent by the CPU on packet processing are dominated by large data and protocol processing overheads. Continuous small optimizations have reduced this overhead - e.g. from 2425 cycles/pkt in a Linux 2.6 based router [6] to 1979 cycles/pkt in Linux 3.7 [21]. However, with these overheads, it is difficult to handle and process incoming packets efficiently on standard computer systems: A 10 GbE link that is saturated with 64 byte packets requires processing a packet each 67 ns which means that a single 3 GHz core only has ∼200 cycles per packet. A 10-core system would be required to serve packets in line rate at 2000 cycles/pkt [21]. Specialized software for fast packet processing like the Data Plane Development Kit [14] (DPDK) can reduce the consumed cycles per packet for forwarding of a packet to roughly 120 cycles [9]. In these specialized software systems, any CPU overhead (e.g. for interrupt handling) is avoided.

However, processing packets in more complex routines, such as the TCP/IP stack or with cryptographic actions, at high rates is more challenging [15]. The workload increases linearly with the number of packets or bytes: cryptographic actions increase the consumed cycles in terms of per-byte-costs, TCP stack processing increases the costs in terms of per-packet-costs. In both cases, performing actions beside the CPU reduces the load on the CPU.

Commonly used techniques are: IP, TCP, and UDP checksum offloading; Receive Side Coalescing (RSC); and TCP segmentation offloading (TSO). NICs help the CPU with checksum computation and validation and reduce the per packet costs by combining and splitting packets before and after receiving and sending by the OS. Hosts can send and receive packets bigger than the MTU and the fixed per-packet costs are distributed on the data of different lower level packets: splitting (TSO) and combination (RSC) is done transparently by the NIC.

III. VPN handling on COTS servers

In the following sections, we describe the background that will lead to our case study about VPN NIC-offloading.

A. IPsec for VPN

IPsec is a set of protocol extensions for authentication and encryption on the IP layer. IPsec has two extensions: The *Authentication Header* provides authentication of IP packets by adding a header containing a cryptographic hash of the IP packet payload and parts of the header to the packet and the *Encapsulating Security Payload* for encryption (and authentication) of the IP packet payload (only). These extensions can either operate in transport mode, directly applied to the IP packet, or in tunnel mode where the packets are encapsulated and transported to the tunnel endpoint where the encapsulating headers are again removed. Each IPsec secured communication has *security associations* which are encryption parameters such as an encryption algorithm and secret keys. These parameters have to be negotiated before the secured communication starts.

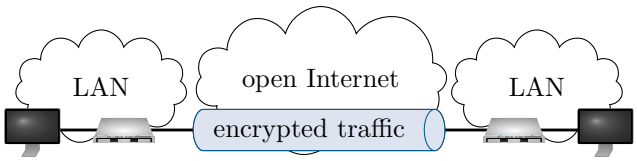


Figure 1. IPsec VPN tunnel mode

A Virtual Private Network (VPN) is a private network that is realized as an overlay network. The connection of private networks is realized via point-to-point tunneling protocols. Figure 1 shows a typical setup with IPsec in tunnel mode used for establishing a VPN that virtually connects two physically separated LANs.

B. State of the art

Commercially available routers of vendors like Cisco, have integrated IPsec VPN functionality with very limited performance if the routers are not accelerated by hardware add-ons. The Cisco ASR1000 series (Aggregation Services Router) provides an IPsec throughput of up to 7 Gbit/s in its default configuration [4]. However, if additional hardware add-ons (Embedded Services Processors) and their corresponding licenses are purchased and aggregation is used, these Cisco routers can deliver an IPsec throughput of up to 78 Gbit/s [5]. Other vendors provide similar routers with VPN security gateway functionality, such as gateprotect, WatchGuard, etc. The prices for such hardware VPN solutions, which can deliver a VPN throughput of around 10 Gbit/s, start at about 20,000 EUR. A new class of routers that performs arbitrary data plane and control plane actions in software is gaining momentum. For example, the MikroTik Cloud Core Router CCR1036-8G-2S+ can be purchased for less than 1,000 EUR. This router allows to perform arbitrary functionalities in software; amongst others routing protocols, firewall functionality, OpenFlow, and IPsec. Due to the many cores (i.e. 36×1.2 GHz), the versatile configurations, their side effects, and the continuously improving software performance, an analysis is difficult. Depending on the configuration, an IPsec throughput from 1.7 Gbit/s (1500 MTU) to 7.5 Gbit/s (9000 MTU) is possible on such a router [20].

Hybrid VPN solutions are implemented in software but make use of acceleration hardware. Through a cooperation between Microsoft and Intel, the *IPsec Task Offload* functionality was integrated into the network drivers of Windows Server 2008 which enables an IPsec throughput at line speed of the NIC [18]. Also, 6WIND offers proprietary IPsec solutions (6WINDGate). Their solution is based on DPDK and makes use of cryptography co-processors, such as the *Intel QuickAssist* or *Cavium NITROX III*, each of which delivering between 5 and 35 Gbit/s VPN throughput [3], depending on the co-processor in use. Our presented approach shares the use of DPDK with 6WINDGate. Sabin and Rashti [23] described 10 GbE NICs that can speed up applications by offloading network tasks (including cryptography) via a user layer API provided by special software drivers onto programmable firmware of special NICs. For Windows, Microsoft announced the azure smart NICs [10] which make hardware acceleration available to applications. PacketShader [11] is an open-source GPU-accelerated software router platform. Although its implementation suffers from high overheads for the transfer of data to the GPU memory, the offloading of CPU workload onto a GPU increases performance especially for high workload processing tasks. In the concrete scenario, a server with four 2.66 GHz cores configured as IPsec gateway was able to process between 10 and 20 Gbit/s depending on the packet size (accelerated by an NVIDIA GTX480 with 480 cores, 1.4 GHz, 1.5 GB) instead of 3 to 6 Gbit/s [11]. Although the NIC-accelerated software was not a router but an OpenFlow Switch in the next example, Tanyingyong et al. used an Intel 82599-based NIC to offload parts of the forwarding information base to the NIC and thus increased the throughput significantly [26].

The Linux network stack does IPsec in software: RouteBricks [6] achieved a Linux IPsec performance of 1.9 Gbit/s for 64 B traffic and 6.1 Gbit/s for larger packets on 8 cores with 2.4 GHz each. In our experiments with Linux 3.16 on an *Intel Xeon E3-1230 v2* CPU with 3.30 GHz, we achieved comparable processing rates: in our setup, the Linux kernel was able to process a VPN throughput between 0.5 and 1.3 Gbit/s per core, heavily dependent on the packet size. If we used the *Intel AES-NI* instruction set, by loading the *aesni_intel* kernel module, the VPN throughput increased to between 0.9 and 4.3 Gbit/s per core.

In addition to the Linux implementation, other software solutions like *Turbo IPsec*, provided by 6WIND, exist. Although Turbo IPsec is based on Intel DPDK it is different to 6WINDGate: it does not use IPsec hardware offloading capabilities. TurboIPsec reaches an IPsec VPN throughput of 5 Gbit/s per core on an Intel Xeon E7-4890v2 processor clocked at 2.8 GHz [1] and a throughput of 47.9 Gbit/s on 8 cores of an Intel Xeon E5-2680 v3 at 2.50 GHz [25].

IV. Case study

We performed a case study of NIC acceleration by exploiting the IPsec capabilities of the 10 GbE controllers Intel X540 [12] and Intel 82599 [13].

Both Intel 10 GbE NICs (X540 and 82599) support offloading, encryption, and authentication for up to 1024 security associations in each direction and for up to 128 IPs at the receiver side. They support IPsec cryptography and authentication in both the ESP (Encapsulating Security Payload) and AH (Authentication Header) modes. The cryptographic algorithm implemented on the chips is AES-128-GCM/GMAC – up-to-date considered as secure [16]. The *AES-128-GMAC* algorithm is implemented for authentication (AH or ESP without encryption) and the *AES-128-GCM* algorithm for encryption plus authentication (ESP) [12], [13]. In addition to IPsec the *AES-128-GMAC* algorithm finds use in other state-of-the-art cryptographic protocols, such as TLS 1.2 [17], [24]. The data sheets do not mention any performance limitations indicating potential operation speed in line rate [12], [13]. However, the open source driver implementation *ixgbe* in the Linux network stack, provided by Intel themselves, does not make use of this functionality.

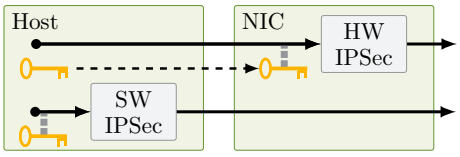


Figure 2. NIC accelerated and software IPsec processing path

Figure 2 visualizes the role of the NIC as an IPsec offloading component compared to a software-based IPsec connection. For the hardware supported approach, encryption happens later in the data path and the security association –visualized by a key– has to be passed onto the NIC. Using a software-based approach, an IP packet with a valid IPsec header must be prepared by the software with the unencrypted data as a payload before the NIC is ordered to finish the incomplete IPsec packet by encrypting it and creating an authentication hash. On the receiving path, the NIC decrypts the payload and checks the packet for authenticity. The host is then notified about the success or failure of the decryption and the authenticity check.

A. Implementation

Our NIC-accelerated VPN gateway application is built on DPDK for optimum performance. We decided for Intel DPDK [14] as other frameworks like netmap do not allow access to NIC specific features [7] and provide less performance [9]. We implemented IPsec offloading in the `librte_pmd_ixgbe` userspace driver that is included in DPDK Our proof-of-concept application is built directly on top of MoonGen [7], [19] a framework for packet processing in Lua. It provides an abstraction layer on top of DPDK for prototyping of DPDK applications. The LuaJIT compiler achieves roughly the same speed as native implementations based on C/C++ and DPDK for simple packet processing tasks [7]. The entire implementation was merged into MoonGen and is available under the MIT license in our repository[19], but IPsec functionality has not been applied or discussed in our previous publications.

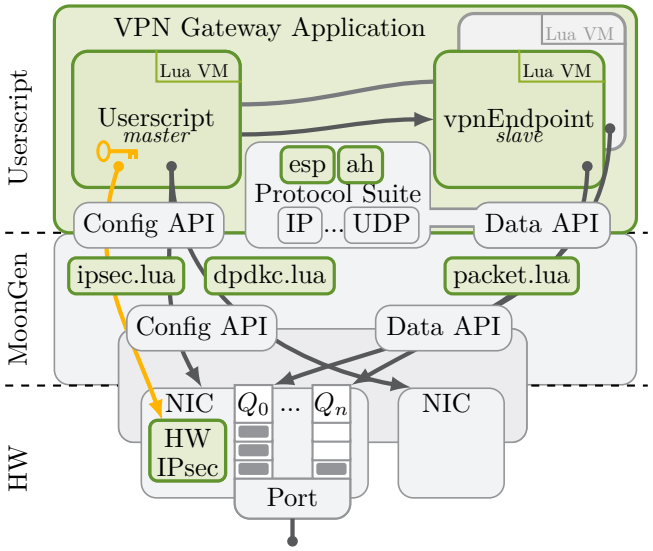


Figure 3. MoonGen architecture: IPsec components

Figure 3 shows the components that were added to MoonGen for the hardware accelerated IPsec functionality (new and modified components are highlighted with green-colored, thick borders). The **IPsec library** (`ipsec.lua`) provides utility functions. It contains functions like *esp_vpn_encapsulate()* which accesses the NIC registers to trigger the encryption. The `dpdkc` interface (`dpdkc.lua`) was extended to support IPsec security associations and to configure the IPsec mode. **ESP and AH protocol support** was added to the MoonGen protocol suite (`esp.lua` and `ah.lua`). The packet library (`packet.lua`) was extended to support options like the IPsec offload flag.

The proof-of-concept **VPN Gateway Application** was implemented as MoonGen userscript (`vpn-esp-forwarder.lua`). Listing 1 shows the new *vpnEndpoint()* function. It is executed in a separate thread as LuaJIT VM that is started after the cryptographic engine is activated via *ipsec.enable()* and the required IPsec security association is installed, via a call of *ipsec.tx_set_key()*. First, pre-filled IP packets (*new_mem*) are allocated (line 2 – 10). In the main loop (line 12 – 29) the VPN IPsec ESP tunnel gateway receives a burst of packets (*tryRecv()*) and encapsulates them into the pre-allocated IP packets with an ESP (*esp_vpn_encapsulate()*). The encapsulation function is part of `ipsec.lua` and calculates the new packet size and the ESP padding to achieve a 4 byte alignment required by the cryptographic algorithm on the NIC. Afterwards the encapsulation function uses performance optimized functions to copy all bytes of the received IP packet into the newly allocated ESP packet buffer. Additionally, the encapsulation function appends an ESP trailer, containing the new ESP packet length, the number of padding bytes and 16 bytes of zeros, where the network hardware can place the integrity check value (ICV) later on. This final packet buffer is then returned to the main loop where the DPDK offload flags (*ol_flags*) for hardware-based

IPsec encryption and authentication (*offloadIPSec()*) are activated. The *offloadIPSec()* function sets IPsec specific offload flags (*ol_flags*) in DPDK via the `dpdkc.lua` interface. Finally, the encapsulated packets are sent to the TX interface of the IPsec tunnel and buffers are freed.

Listing 1. MoonGen VPN IPsec ESP encapsulation

```
function vpnEndpoint(rxQ, txQ, src_mac, src_ip,
    dst_mac, dst_ip, spi, sa_idx)
  local bufs = memory.bufArray()
  local new_mem = memory.createMemPool(
  function(buf)
    buf:getEspPacket():fill{
      ethDst = dst_mac,
      ip4Dst = dst_ip,
      espSPI = spi,
    }
  end)
  while dpdk.running() do
    local rx = rxQ:tryRecv(bufs,0)
    local esp_bufs = new_mem:bufArray(rx)
    for i = 1, rx do
      local pkt = bufs[i]:getIPPacket()
      local len = pkt.ip4:getLength()
      if not pkt.ip4:getProtocol() ==
        ip.PROTO_ESP then
        esp_bufs[i] =
          ipsec.esp_vpn_encapsulate(bufs[i],
            len, new_mem)
      end
    end
    esp_bufs:offloadIPChecksums()
    esp_bufs:offloadIPSec(sa_idx,"esp",1)
    txQ:send(esp_bufs)
    bufs:freeAll()
  end
end
```

The RX side of the VPN tunnel works in an analogous way: That is, in-coming packets are decrypted by the hardware using the installed IPsec RX security associations. The packet's RX descriptor is analyzed by the *ixgbe_recv_pkts()* function `ixgbe_rxtx.c` in DPDK in order to extract the IPsec offloading status (*SECP*) and error (*SECERR*) flags, which are transfered to MoonGen via the packet's offload flags (*ol_flags*). These flags can be read, using the *getSecFlags()* function of `packet.lua`. If the IPsec decryption was offloaded (*SECP = 1*) and no errors are reported (*SECERR = 0*), the IP packet contained in the decrypted ESP packet can be decapsulated using the *esp_vpn_decapsulate()* function of MoonGen's IPsec library in order to re-create the original IP packet which can then be forwarded into the destination network.

B. Dealing with limitations

Our implementation currently does not support IPsec AH mode via IPv6 or IP fragmentation. Beside limitations of our implementation, IPsec NIC offloading is inherently limited by the available algorithms of the NIC. As mentioned in Section IV, AES-128-GCM/GMAC is the (only) cryptographic algorithm available on our NICs. Another drawback is that security offloading is limited to fixed IP packet header lengths. Therefore, our implementation also cannot handle IPv6 extension headers and IPv4 options.

Due to the limitations, a seamless fall back to software implementations is necessary whenever hardware support is not given. To make IPsec offloading functionality available as a fast path in Linux [2] configuration interfaces beyond the scope of this paper are required to operate with software like strongSwan and Racoon (cf. Section III-B).

V. Evaluation

While different network flows may pass a VPN gateway, traffic in transport mode is limited by the application. Therefore, we evaluate the VPN security gateway (IPsec/ESP tunnel mode). We assume a statically configured tunnel as the security negotiation process can not be offloaded to the NIC. Although different tunnels can be handled by the NIC without additional stress for the CPU, we only configured one tunnel to have a comparable setup.

Our evaluation comprises VPN throughput and energy consumption. We decided against a comparison with rather unused software based on DPDK or even an own implementation based on MoonGen as the expressiveness of such a comparison is very limited. For comparison to our implemented PoC, we selected Linux-based IPsec as we consider it to be the state of the art implementation which has been used as a point of reference in the past [11], [6]. Therefore, we performed measurements with the *hardware accelerated MoonGen IPsec stack*, a *Linux 3.16 IPsec stack*, and a *Linux 3.16 IPsec stack applying the Intel AES-NI instruction set*.

A. Test setup

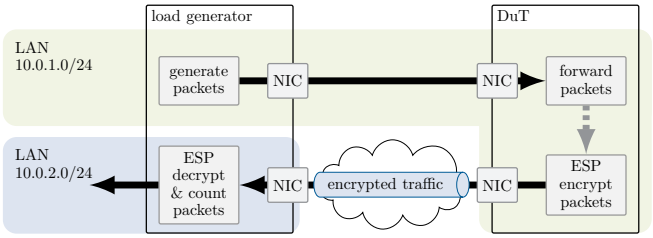
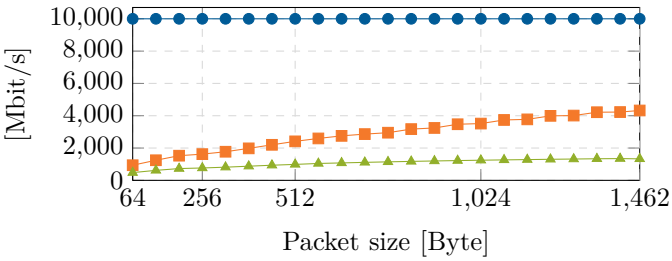


Figure 4. Test setup

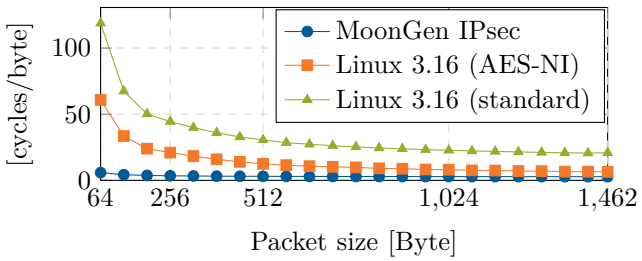
Our test setup (cf. Figure 4) consists of two directly connected servers one acting as load generator and sink and the other as device under test (DuT). The hardware used in the DuT was an Intel Xeon E3-1230 v2 CPU, clocked at 3.30 GHz, and an Intel 82599 based fiber optic NIC. We configured the DuT to only utilize a single CPU core, to compare the different network stacks. As load generator, we configured MoonGen to generate IPv4/UDP packets from the subnet 10.0.1.0/24, destined for the subnet 10.0.2.0/24, at line rate (10 Gbit/s). The generated packets are sent to the DuT which acts as a VPN gateway router that is configured to route the packets back to the load generator through an encrypted and authenticated IPsec ESP tunnel between 1.1.1.1 and 2.2.2.2. Finally, the sink decrypts and authenticates the packets arriving through the tunnel and counts the inbound data.

B. VPN throughput

We measured the maximal throughput for IPv4/UDP packet encryption and authentication in IPsec ESP tunnel mode applying AES-128-GCM. The offered load comprises



(a) VPN throughput at different packet sizes



(b) Consumed CPU cycles per byte

Figure 5. Performance evaluation of MoonGen IPsec

of packet streams differing in size at 10 GbE line rate. Figure 5(a) shows the rates we achieved with the Linux IPsec stack and with our approach. The VPN throughput performance of the Linux IPsec stack heavily depends on the size of the received packets and shows an approximately linear growth starting at 0.94 Gbit/s (0.48 Gbit/s without AES-NI) and maxing out at 4.32 Gbit/s (1.33 Gbit/s without AES-NI). However, the VPN throughput of the MoonGen IPsec stack, using the security offloading capabilities of the Intel 82599 NIC, constantly reaches the line speed of 10 Gbit/s for all packet sizes.

Figure 5(b) shows the CPU cycles per byte (Y). It was computed by dividing the CPU clock rate of 3.30 GHz by the number of processed packets per second ($pktCount$) and the size of the packets ($pktSize$). To exclude idle CPU cycles the clock rate is multiplied with the CPU usage. This is expressed by the following formula:

$$Y[\frac{Cycles}{byte}] = \frac{CPUload[\frac{Cycles}{s}]}{pktCount[\frac{1}{s}] \cdot pktSize[byte]}$$

C. Accounting for the used CPU cycles

To explain the performance limitations (cf. Figure 5) two aspects of software-based packet processing systems are important:

(1) Packets are processed by different software functions. We classify these into functional groups for packet reception, packet processing (mostly encryption in our case), and packet transmission similar to other publications [21], [9].

(2) For different task(s) the cycles consumed for it depend on other factors: The number of packets determines consumed CPU cycles for tasks that operate on header data and the number of bytes for tasks that process the whole packet data. Deeper analysis may require more complex performance modeling, e.g. considering the number of batches and the batch size.

Here a simplified view is appropriate: CPU cycles for packet reception and transmission do not depend on the packet size but on the number of packets [8]. Cryptographic operations also affect the packet payload and therefore depend on the bytes. The increased Mbit/s for the Linux setups can be explained by overhead for packet reception and transmission which is constant per packet [21] and by a workload in terms of both bytes/s and pkt/s for cryptography that is also applied on the payload. When the cryptographic workload is offloaded, the CPU has only packet dependent workload. MoonGen requires about 3 cycles/byte, and a bit more at very low packet sizes, whereas the Linux IPsec stack uses 10 to 20 times as many CPU cycles per byte of a packet.

Another obstacle comes from the fact that polling-based packet reception always produces 100% CPU utilization. However, CPU cycles spent for polling an empty or not fully utilized batch size can be used for other tasks without lowering the rate of received packets. Metering an effective CPU usage as input for computing the cycles per byte Y mitigates this. To account for potentially unused CPU capacity, an effective CPU usage ratio can be determined in setups where the maximum throughput is below line rate. These setups can be achieved by reducing the CPU clock (cf. Rizzo et al. [22]), by adding additional load (cf. Gallenmüller et al. [9]), or using a sleep time after each poll that is increased until throughput falls below line rate. For MoonGen IPsec we measured an effective CPU usage of 10%-20%. If we apply the effective CPU usage to the computation of Y, MoonGen IPsec performance is even better as depicted in Figure 5.

D. Comparing to commercial software

The 6WIND Turbo IPsec solution is a commercial software that is also based on DPDK and comes with its own IPsec implementation. As we did not own a license, we had to rely on published performance values of 6WIND Turbo IPsec. For a better comparison we make basic assumptions that have been shown to hold true in similar scenarios: The maximum throughput scales linearly with the number of CPU cores and the clock rate [7], [22]. We extract our support vector from the best available source of performance data [25]: The 47.9 Gbit/s achieved with 6WIND Turbo IPsec (cf. Section III-B) relate to ∼7.9 Gbit/s (≈0.66 Mpps) when normalized to one core of our DuT. The cycles per byte for Turbo IPsec are ∼3.4 cycles/byte which is less efficient than our approach.

E. Energy savings

In addition to throughput, power consumption is an important performance indicator. We used a *GUDE Expert Power Control 8225-1* system for our measurements. The idle system has a power consumption of 88 W. The Linux 3.16 IPsec stack at a VPN throughput rate of 1 Gbit/s requires one CPU core permanently using 100% of its capacity, leading to a power consumption of 108 W (107 W

Table I. Energy comparison

	power drain [W]	throughput [Gbit/s]	CPU load at 3.3 GHz	Energy/B [mJ]
MoonGen	100	1.00	20%	0.100
Linux 3.16 (AES-NI)	108	1.00	100%	0.108
Linux 3.16	107	0.48	100%	0.208

without Intel AES-NI drastically decreased throughput). To compare the MoonGen IPsec stack, we reduced the VPN throughput rate to 1 Gbit/s, by artificially slowing down the CPU via a sleep for a few microseconds in between each burst of processed packets. Thus we reduced the CPU load to 20% on that CPU core lowering the power consumption to 100 W. Activating the NICs hardware cryptographic engine uses just 0.2 W of extra power [13], which we consider negligible compared to 80% savings in CPU load.

Detailed results of the energy measurements are displayed in Table I. The worst case scenario – using minimum sized packets – results in a CPU utilization of 20% for MoonGen. For packet sizes of $\geq$ 256 bytes, MoonGen reaches its maximum efficiency ($\approx$ 3 cycles/byte) and thus just requires about 10% of the CPU's capacity.

VI. Conclusion

We demonstrated that commodity hardware performance can cope with expensive networking devices even in challenging CPU consuming scenarios with cryptographic actions. The open source device driver for the Intel X540 and 82599 NICs, based on DPDK and MoonGen, were improved to make use of their security offloading capabilities. A VPN security gateway application, based on MoonGen and implemented using the IPsec ESP tunnel mode, was created to demonstrate and evaluate the VPN throughput and energy saving performance of the improved device driver. All our used source code is available in the MoonGen main git repository.

Although IPsec capabilities of NICs are not new, this is the first published case study. It discloses the trade-off between diversity of functionality and performance. Further steps have to be taken to transparently include such functionality, e.g. via the Linux kernel's *ixgbe* driver and to connect with applications like strongSwan and Racoon where possible. The DPDK driver can be used as reference implementation of the NIC's security offloading capabilities.

Finally, we showed that the use of offloading features lowers the energy consumption of servers when using the NIC hardware based security offloading functionality instead of calculating cryptographic operations on the CPU.

References

[1] 6WIND Turbo IPsec. http://www.6wind.com/products/6wind-turbo-ipsec. Last visited 2015-09-21.

[2] H. Agrawal and M. Dev. Accelerating network packet processing in Linux. http://www.embedded.com/design/operating-systems/4403058/Accelerating, 2015.

[3] Cavium. NITROX III Family of Security Processors. 2014.

[4] Cisco. Network Security Features for Cisco ASR 1000 Series Routers. 2010.

[5] Cisco. Cisco ASR 1000 Series Embedded Services Processors Data Sheet. 2015.

[6] M. Dobrescu, N. Egi, K. Argyraki, B.-G. Chun, K. Fall, G. Iannaccone, A. Knies, M. Manesh, and S. Ratnasamy. RouteBricks: Exploiting Parallelism to Scale Software Routers. In *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles*, SOSP, 2009.

[7] P. Emmerich, S. Gallenmüller, D. Raumer, F. Wohlfart, and G. Carle. MoonGen: A Scriptable High-Speed Packet Generator. In *15th ACM SIGCOMM Conference on Internet Measurement (IMC'15)*, 2015.

[8] P. Emmerich, D. Raumer, F. Wohlfart, and G. Carle. Assessing Soft- and Hardware Bottlenecks in PC-based Packet Forwarding Systems. In *Fourteenth International Conference on Networks (ICN 2015), Best Paper Award*, 2015.

[9] S. Gallenmüller, P. Emmerich, F. Wohlfart, D. Raumer, and G. Carle. Comparison of Frameworks for High-Performance Packet IO. In *ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, 2015.

[10] A. Greenberg. SDN for the Cloud. Keynote at the ACM SIGCOMM 2015.

[11] S. Han, K. Jang, K. Park, and S. Moon. PacketShader: A GPU-accelerated Software Router. *SIGCOMM Comput. Commun. Rev.*, 40(4), 2010.

[12] Intel. Ethernet controller x540 datasheet. March 2014. Revision 2.7.

[13] Intel. 82599 10 GbE Controller Datasheet. February 2015. Revision 3.1.

[14] Data Plane Development Kit. http://dpdk.org/. Last visited 2015-10-27.

[15] S. Makineni and R. Iyer. Receive Side Coalescing for Accelerating TCP/IP Processing. In *Proceedings of the 13th international conference on High Performance Computing(HiPC'06)*, 2006.

[16] D. McGrew and P. Hoffman. RFC 7321: Cryptographic Algorithm Implementation Requirements and Usage Guidance for Encapsulating Security Payload (ESP) and Authentication Header (AH). 2006.

[17] D. McGrew and J. Viega. RFC 4543: The Use of Galois Message Authentication Code (GMAC) in IPsec ESP and AH. 2006.

[18] Microsoft and Intel. IP Security features: Intel Ethernet Server Adapters and Microsoft Windows Server 2008. 2009.

[19] MoonGen. https://github.com/emmericp/MoonGen.

[20] K. Myers. 10 Gbps of Layer 2 throughput is possible using MikroTik's EoIP tunnel. http://www.stubarea51.net/2015/10/16/10/, 2015. Last visited 2015-10-27.

[21] D. Raumer, F. Wohlfart, D. Scholz, and G. Carle. Performance Exploration of Software-based Packet Processing Systems. In *Proceedings of Leistungs-, Zuverlässigkeits- und Verlässlichkeitsbewertung von Kommunikationsnetzen und Verteilten Systemen, 6. GI/ITG-Workshop, MMBnet*, 2015.

[22] L. Rizzo. netmap: a novel framework for fast packet I/O. In *USENIX Annual Technical Conference*, 2012.

[23] G. Sabin and M. Rashti. Security offload using the smartnic, a programmable 10 gbps ethernet nic. In *2015 National Aerospace and Electronics Conference (NAECON)*. IEEE, 2015.

[24] J. Salowey, A. Choudhury, and D. McGrew. RFC 5288: AES Galois Counter Mode (GCM) Cipher Suites for TLS. 2008.

[25] SDxCentral Labs Networking Software Performance Test. 6WIND Speed Series Performance Validation. 2015.

[26] V. Tanyingyong, M. Hidell, and P. Sjodin. Improving performance in a combined router/server. In *High Performance Switching and Routing (HPSR), 2012 IEEE 13th International Conference on*, 2012.