Autonomous NIC Offloads

Boris Pismenny[†◇] Haggai Eran[†◇] Aviad Yehezkel[◇] Liran Liss[◇] Adam Morrison[‡] Dan Tsafrir[†±]

[†] Technion – Israel Institute of Technology

[◇] Mellanox

[‡] Tel Aviv University

[±] VMware Research

1. Motivation

Layer-5 networking protocols (L5Ps) built on top of TCP are widely used. Examples include (1) the transport layer security (TLS) cryptographic protocol [1, 2], which provides secure communications; (2) storage protocols, such as NVMe-TCP [3], which allow systems to use remote disk drives as local block devices; (3) remote procedure call (RPC) protocols, such as Thrift [4] and gRPC [5]; and (4) key-value store protocols, such as Memcached [6] and MongoDB [7].

It is in the nature of L5P processing to be bottlenecked by *data-intensive* operations, which move date while optionally transforming and/or computing some function over it. In most cases, this data-intensive processing consists of some subset of the following: (1) encryption/decryption, (2) copying, (3) hashing, (4) (de)serialization, and (5) (de)compression [8]. Consequently, it is beneficial to accelerate these data-intensive operations, to improve the performance of systems that utilize and depend on the L5Ps.

2. Limitations of the State of the Art

We classify prior approaches for accelerating L5P processing into three categories. The first is *software-based*, which includes in-kernel L5P implementations, such as NVMe-TCP [9, 10] and TLS [11] in the Linux kernel, and specialized software stacks that bypass the kernel and leverage direct hardware access [12, 13]. These techniques are good for reducing the cost of system software abstractions. But they are largely irrelevant for accelerating data-intensive operations.

The second category consists of *on-CPU acceleration*, and it encompasses specialized data-processing CPU instructions, such as those supporting the advanced encryption standard (AES) [14, 15], secure hash algorithms (SHA) [16, 17], and cyclic redundancy check (CRC) error detection [18, 19]. These instructions can be effective in accelerating L5Ps. But they themselves then become responsible for most of the L5P processing cycles, motivating the use of *off-CPU accelerators*, which comprise the third category. We subdivide the latter into two: accelerators that are off and on the networking path.

The goal of *off-path* acceleration is to offload the operation onto a separate device, such as GPUs in APUNet [20] and Intel QuickAssist in QTLS [21]. The problems with off-path accelerators are: (1) that the CPU still spends cycles on feeding the accelerator and retrieving results; (2) that applications may need to be rewritten so that the CPU is kept busy while waiting for the accelerator (e.g., QTLS re-engineers the TLS software stack and Nginx to increase parallelism); and (3) that they consume memory bandwidth and increase latency due to accelerator communication overheads, such as DMA. Overall, off-path acceleration is suboptimal [21, 22].

On-path accelerators—namely, NICs—do not suffer from the above problems. CPUs necessarily operate NICs in any case as part of the L5P, and NICs may process the data while it flows through them without incurring additional overheads; they are *ideally* situated for L5P processing. In fact, NICs already seamlessly handle offloaded computation for the underlying layer$\leq$4 protocols, such as packet segmentation, aggregation, and checksum computation/verification [23, 24, 25].

Despite their ideal suitability, however, L5P NIC offloads are not pervasive. The reason: existing designs assign NICs with the role of handling the L5P, which in turn necessitates that the NICs also handle layer$\leq$4 functionality upon which the L5P depends—notably TCP/IP [26, 27, 28, 29]. Such *offload dependence* is undesirable, because a hardware TCP implementation encumbers innovation in the network stack [30] and slows down fixes when robustness [31] or security issues [32, 33] arise. Consequently, Linux kernel developers refused to support TCP offloads [34, 35], and Windows has recently deprecated such support [36].

3. Key Insights

We propose *autonomous NIC offloads*. Autonomous offloads consist of a combined software/NIC architecture for moving data between L5P memory and TCP packets, optionally transforming the data and/or computing some function in the process. This architecture allows L5P software to offload data-intensive operations to the NIC, without migrating the entire TCP/IP stack into the NIC. Autonomous offloads target L5P implementations in which the L5P and NIC driver can communicate directly, e.g., in-kernel L5P implementations or implementations with userspace TCP/IP stacks.

The main idea of autonomous offloads is for the L5P and NIC to *collaborate* in processing of L5P messages (which can consist of multiple TCP segments) in a way that is *transparent* to the intermediating TCP stack. When sending a message, the L5P code "skips" performing the offloaded operation, thereby passing the "wrong" bytes down the stack to the NIC. The NIC then performs the operation, resulting in a correct message being sent on the wire. In the reverse direction, the NIC parses incoming messages, performs the offloaded operation, and the partially-processed message is passed up the stack to the L5P, which completes its processing.

For this approach to work seamlessly, we require that the

offloaded operation satisfy certain conditions, notably, that the offloaded operation only manipulates bytes in L5P messages and never adds or removes bytes. This property allows the offload to process the TCP stream without modifying TCP control information. While this limitation means we cannot offload every data-intensive operation, a key practical observation is that the lion's share of L5P data-intensive computations satisfy our conditions. Out of (1) encryption/decryption, (2) copying, (3) digests/checksums, (4) (de)serialization, and (5) (de)compression—which are the primary data-intensive L5P bottlenecks [8]—only (4) and (5) are partially out of scope.

A major challenge in realizing the conceptually simple idea of autonomous offloads is handling "interference" from the TCP layer, in the form of lost, duplicated, or retransmitted bytes. Our design handles this challenge through three main ideas: (1) we optimize for the common case by maintaining a small context at the NIC required to process the next in-order TCP packet; (2) we fall back on L5P software processing upon out-of-sequence packets caused by reordering or loss; in which case (3) the L5P software helps the NIC to synchronize and reconstruct a new valid context using a simple interface between the NIC and the L5P.

A potential pitfall of such a collaborative resynchronization mechanism is that the NIC might get permanently "left behind." That is, by the time L5P software updates the NIC's context, more TCP segments have arrived, thereby making that new context stale and thus requiring another resynchronization.

Our solution to this problem relies on two insights. First, we observe that in our targeted offload operations, the required offload context can be recomputed after identifying L5P message boundaries. This property reduces the problem of reconstructing offload context to the problem of identifying L5P message boundaries. Second, we observe that in most modern L5Ps, message boundaries are identifiable with specific "magic patterns" and contain length fields. This property enables us to design a *hardware-driven* context resynchronization process, in which the NIC speculatively identifies arriving messages and relies on software to *confirm* its speculation. While the NIC waits for software to resolve its speculation, it tracks messages within TCP packets using the L5P header's length field, verifying that each subsequent L5P message begins where it is expected to. If an unexpected pattern is encountered, the NIC realizes its speculation was incorrect and retries. If, however, the NIC's speculation is correct, then it remains synchronized until receiving confirmation from L5P software. At this point, the NIC can resume offloading as soon as possible.

4. Main Artifacts

The paper describes autonomous offload implementations (hardware and software) for TLS and NVMe-TCP. Our TLS offload is already implemented Mellanox ConnectX6-Dx NICs [37]; it offloads TLS's authentication, encryption, and decryption functionalities. Our NVMe-TCP offload will become available in the subsequent NIC model; it offloads data placement at the receiving end (which thus becomes zero-copy) and also CRC computation and verification at either end. Linux kernel support for the TLS offload has been upstreamed, and support for the NVMe-TCP offload is currently under review.

L5P	*application*	*offloads*	*max improvement*		
			t/put	*util*	*lat*
NVMe-TCP	fio	copy, CRC	2.2x	2.0x	1.3x
TLS	iperf	crypto	3.3x	2.4x	N/A
NVMe-TCP	nginx/http	copy, CRC	1.4x	1.3x	1.2x
TLS	nginx/https	crypto	2.7x	1.3x	1.2x
both (NVMe-TLS)	nginx/https	all above	2.8x	1.7x	1.4x
both (NVMe-TLS)	redis	all above	2.3x	1.9x	N/A

Table 1: *Summary of evaluation results – throughput ("t/put"), CPU utilization ("util"), and latency ("lat"). Data is served either from memory (TLS), or from a remote disk over NVMe-TCP, which can also be encrypted with TLS (NVMe-TLS). In NVMe-TLS, crypto is offloaded for both sent HTTPS and received NVMe-TLS traffic. "Crypto" is AES128-GCM encryption/decryption and authentication.*

We evaluate the throughput, latency, and CPU utilization improvements achieved using our offloads on both microbenchmarks and macrobenchmarks. As microbenchmarks, we use iperf [38] for TLS and fio [39] for NVMe-TCP. As macrobenchmarks, we use the nginx http web server [40] and the Redis-on-Flash (RoF) key-value store [41, 42]. Both applications serve files to clients: When serving files from memory, they are network bound and stress TLS. When serving files from a remote NVMe device, they stress NVMe-TCP. We also evaluate a combined workload: serving files located on a remote device over a TLS connection. All experiments use real TLS offloading hardware and emulated NVMe-TCP offloading, since the NVMe-TCP offload NIC is not yet available.

5. Key Results and Contributions

- Defining, designing, and implementing autonomous NIC offloads, which provide a stateful, nonintrusive software/device architecture that accelerates TCP-based L5Ps by leveraging the existing TCP/IP stack rather than subsuming it.
- A cooperative software/NIC design for handling TCP packet loss/reordering that (1) allows the NIC to perform computations in hardware in the common case; (2) falls back on L5P software processing upon out-of-sequence packets; and then (3) re-synchronizes NIC state with minimal help from software so the NIC can resume offloading.
- An implementation of autonomous offloads for (1) NVMe-TCP copy and CRC, and (2) TLS encryption, decryption, and authentication, and (3) their composition (NVMe-TLS).
- Experimental evaluation of a full (ASIC) 100 Gbps TLS offload and an emulation of the upcoming NVMe-TCP offload. Our autonomous offloads improve throughput by up to 3.3x and lower CPU consumption and latency by as much as 2.4x and 1.4x. Table 1 summarizes the results.

References

[1] Tim Dierks and Eric Rescorla. The transport layer security (TLS) protocol version 1.2. RFC, RFC Editor, 2008. https://rfc-editor.org/rfc/rfc5246.txt.

[2] Eric Rescorla. The transport layer security (TLS) protocol version 1.3. RFC, RFC Editor, 2018. https://rfc-editor.org/rfc/rfc8446.txt.

[3] NVM Express Workgroup. NVMe/TCP transport binding specification. https://nvmexpress.org/wp-content/uploads/NVM-Express-over-Fabrics-1.0-Ratified-TPs.zip, Nov 2018. Accessed: Jan 2020.

[4] Mark Slee, Aditya Agarwal, and Marc Kwiatkowski. Thrift: Scalable cross-language services implementation. *Facebook White Paper*, 5(8), 2007. https://thrift.apache.org/static/files/thrift-20070401.pdf.

[5] Google. gRPC: a high-performance, open source universal RPC framework. https://grpc.io/, 2015. Accessed: 2020-03-05.

[6] Brad Fitzpatrick. Distributed caching with memcached. *Linux Journal*, 2004(124):5, Aug 2004. http://dl.acm.org/citation.cfm?id=1012889.1012894.

[7] Rick A. Jones. MongoDB: The database for modern applications. https://www.mongodb.com/, 2009. Accessed: August, 2020.

[8] Svilen Kanev, Juan Pablo Darago, Kim Hazelwood, Parthasarathy Ranganathan, Tipp Moseley, Gu-Yeon Wei, and David Brooks. Profiling a warehouse-scale computer. In *ACM International Symposium on Computer Architecture (ISCA)*, pages 158—-169, 2015. https://doi.org/10.1145/2872887.2750392.

[9] Jaehyun Hwang, Qizhe Cai, Ao Tang, and Rachit Agarwal. TCP ≈ RDMA: CPU-efficient remote storage access with i10. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 127–140, 2020. https://www.usenix.org/conference/nsdi20/presentation/hwang.

[10] Sagi Grimberg. TCP transport binding for NVMe over Fabrics. https://lwn.net/Articles/772556/, 2018. Accessed: 2020-03-24.

[11] Watson Dave, Pismenny Boris, Lesokhin Ilya, and Yehezkel Aviad. kernel TLS. https://lwn.net/Articles/725721/, 2017. Accessed: 2020-03-24.

[12] Ilias Marinos, Robert N.M. Watson, Mark Handley, and Randall R. Stewart. Disk|Crypt|Net: Rethinking the stack for high-performance video streaming. In *ACM SIGCOMM Conference on Applications Technologies Architecture and Protocols for Computer Communications*, pages 211–224, 2017. https://doi.org/10.1145/3098822.3098844.

[13] Ilias Marinos, Robert N.M. Watson, and Mark Handley. Network stack specialization for performance. In *ACM SIGCOMM Conference on Applications Technologies Architecture and Protocols for Computer Communications*, pages 175–186, 2014. http://doi.acm.org/10.1145/2619239.2626311.

[14] Joan Daemen and Vincent Rijmen. *The design of Rijndael: AES-the advanced encryption standard*. Springer Science & Business Media, 2013.

[15] Shay Gueron. Intel advanced encryption standard instructions (AES-NI). *Intel White Paper*, 2010. https://www.intel.com/content/dam/doc/white-paper/advanced-encryption-standard-new-instructions-set-paper.pdf.

[16] D. Eastlake 3rd and P. Jones. US secure hash algorithm 1 (SHA1). RFC 3174, Internet Engineering Task Force, September 2001. http://www.rfc-editor.org/rfc/rfc3174.txt.

[17] Sean Gulley, Vinodh Gopal, Kirk Yap, Wajdi Feghali, J Guilford, and Gil Wolrich. Intel sha extensions. *Intel White Paper*, 2013. https://software.intel.com/content/dam/develop/external/us/en/documents/intel-sha-extensions-white-paper-402097.pdf.

[18] D. Sheinwald, J. Satran, P. Thaler, and V. Cavanna. Internet protocol small computer system interface (iSCSI) cyclic redundancy check (CRC)/Checksum considerations. RFC 3385, Internet Engineering Task Force, September 2002. http://www.rfc-editor.org/rfc/rfc3385.txt.

[19] Vinodh Gopal, J Guilford, E Ozturk, G Wolrich, W Feghali, J Dixon, and D Karakoyunlu. Fast CRC computation for iSCSI polynomial using CRC32 instruction. *Intel Corporation*, 2011. https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/crc-iscsi-polynomial-crc32-instruction-paper.pdf.

[20] Younghwan Go, Muhammad Asim Jamshed, YoungGyoun Moon, Changho Hwang, and KyoungSoo Park. APUNet: Revitalizing GPU as packet processing accelerator. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 83–96, 2017. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/go.

[21] Xiaokang Hu, Changzheng Wei, Jian Li, Brian Will, Ping Yu, Lu Gong, and Haibing Guan. QTLS: High-performance TLS asynchronous offload framework with intel quickassist technology. In *ACM Symposium on Principals and Practice of Parallel Programming (PPoPP)*, pages 158–172, 2019. http://doi.acm.org/10.1145/3293883.3295705.

[22] Muhammad Shoaib Bin Altaf and David A. Wood. Logca: A high-level performance model for hardware accelerators. In *ACM International Symposium on Computer Architecture (ISCA)*, pages 375–388, 2017. https://doi.org/10.1145/3079856.3080216.

[23] Alexander Duyck. Segmentation offloads. https://www.kernel.org/doc/html/latest/networking/segmentation-offloads.html, 2016. Accessed: 2020-03-24.

[24] Overview of receive segment coalescing. https://docs.microsoft.com/en-us/windows-hardware/drivers/network/overview-of-receive-segment-coalescing, 2017. Accessed: January 2020.

[25] Edward Cree. Checksum offloads. https://www.kernel.org/doc/html/latest/networking/checksum-offloads.html, 2016. Accessed: 2020-03-24.

[26] Chelsio Communications. Chelsio cryptographic offload and acceleration solution overview. https://www.chelsio.com/crypto-solution/, 2018. Accessed: 2018-12-13.

[27] Zsolt István, David Sidler, Gustavo Alonso, and Marko Vukolic. Consensus in a box: Inexpensive coordination in hardware. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 425–438, 2016. https://www.usenix.org/conference/nsdi16/technical-sessions/presentation/istvan.

[28] Zsolt István, David Sidler, and Gustavo Alonso. Caribou: Intelligent distributed storage. *Proceedings of the VLDB Endowment*, pages 1202–1213, 2017. https://doi.org/10.14778/3137628.3137632.

[29] Haggai Eran, Lior Zeno, Maroun Tork, Gabi Malka, and Mark Silberstein. NICA: An infrastructure for inline acceleration of network applications. In *USENIX Annual Technical Conference (ATC)*, pages 345–362, 2019. https://www.usenix.org/conference/atc19/presentation/eran.

[30] Jeffrey C Mogul. TCP offload is a dumb idea whose time has come. In *USENIX Workshop on Hot Topics in Operating Systems (HOTOS)*, pages 25–30, 2003. https://www.usenix.org/conference/hotos-ix/tcp-offload-dumb-idea-whose-time-has-come.

[31] Steven Pope and David Riddoch. 10Gb/s Ethernet performance and retrospective. *SIGCOMM Comput. Commun. Rev.*, 37(2):89–92, March 2007. https://doi.org/10.1145/1232919.1232930.

[32] RedHat. SegmentSmack and FragmentSmack: IP fragments and TCP segments with random offsets may cause a remote denial of service. https://access.redhat.com/articles/3553061, 2019. Accessed: 2020-08-07.

[33] JSOF research lab. Ripple20: 19 zero-day vulnerabilities amplified by the supply chain. https://www.jsof-tech.com/ripple20/, 2019. Accessed: 2020-08-07.

[34] Linux Foundation. Why Linux engineers currently feel that TOE has little merit. https://wiki.linuxfoundation.org/networking/toe, 2016. Accessed: 2018-11-06.

[35] Linux and TCP offload engines. https://lwn.net/Articles/148697/, 2005. Accessed: 2018-11-06.

[36] Microsoft. Why are we deprecating network performance features (kb4014193)? https://techcommunity.microsoft.com/

t5/Core-Infrastructure-and-Security/Why-Are-We-Deprecating-Network-Performance-Features-KB4014193/ba-p/259053, 2017. Accessed: 2019-08-30.

[37] Mellanox. ConnectX®-6 Dx En Card. https://www.mellanox.com/sites/default/files/related-docs/prod_adapter_cards/PB_ConnectX-6_Dx_EN_Card.pdf, 2020.

[38] Ajay Tirumala, Feng Qin, Jon Dugan, Jim Ferguson, and Kevin Gibbs. Iperf: The tcp/udp bandwidth measurement tool. *dast. nlanr. net/Projects*, page 38, 2005. https://iperf.fr/.

[39] Jens Axboe. Fio-flexible io tester. http://freecode.com/projects/fio, 2014.

[40] Will Reese. Nginx: The high-performance web server and reverse proxy. *Linux J.*, 2008(173), September 2008.

[41] Intel. Optimize redis with nextgen nvm. https://www.snia.org/sites/default/files/SDC/2018/presentations/PM/Shu_Kevin_Optimize_Redis_with_NextGen_NVM.pdf, 2018.

[42] Intel. Accelerating redis with intel dc persistent memory. https://ci.spdk.io/download/2019-summit-prc/02_Presentation_13_Accelerating_Redis_with_Intel_Optane_DC_Persistent_Memory_Dennis.pdf, 2019.