Are Our OSes Prepared for Edge Computing?

Pekka Enberg
University of Helsinki
pekka.enberg@cs.helsinki.fi

ABSTRACT

Edge computing is an emerging computing paradigm, which is aimed at latency-sensitive applications such as autonomous vehicles and augmented reality that cannot be deployed to centralized cloud because of high wide area network latencies. Single-board microservers have been proposed as one platform for edge computing because they are low-cost and low energy, and there is a large installation base at the edge of network. However, edge devices currently run commodity OSes, which are evolving largely to cater for the needs of high-end smart devices and servers. Single-board microservers lack hardware capabilities such as multiqueue NICs that are currently needed for low latency on multicore systems. Virtualization has been proposed as a solution for edge computing but virtualization techniques have overheads, which could affect their applicability to edge use cases. The objective of this research is to understand the overheads and limitations in OS designs, OS interfaces, and virtualization techniques that hinder low latency on edge devices and propose an OS design and interfaces that address the needs of edge computing use cases.

1 INTRODUCTION

Edge computing is an emerging computing paradigm, which is aimed at latency-sensitive applications such as autonomous vehicles and augmented reality that cannot be deployed to centralized cloud because of high wide area network latencies [27]. Edge devices are particularly networking intensive because they communicate continuously with low-power sensors, smart devices, and the centralized cloud. Single-board microservers have been proposed as one platform for edge computing because they are low-cost and low energy, and there is a large installation base at the edge of network [20].

Edge devices currently run commodity OSes like Linux. However, commodity OSes are evolving largely to cater for the needs of high-end smart devices and servers. Single-board micro servers aim for low-cost and low-energy and lack hardware capabilities such as multiqueue NICs [3, 25]. Furthermore, virtualization techniques have been long assumed to enable edge computing, but they virtualization techniques have overheads [8], which could affect their applicability to edge use cases.

The primary goals for the design of an OS for edge devices are:

- **Security:** Commodity OSes are complex and implemented in unsafe programming languages, which has made them a target for exploitation [5]. While there is a large community researching commodity OS security issues and problems tend to be fixed quickly, there is still a window of opportunity to exploit edge devices, which are deployed outside of protection of centralized cloud and are largely autonomous.
- **Low latency:** One of the main drivers of edge computing is low latency. However, shared memory kernels have overheads on multicore systems [6] and kernel network stacks and network APIs have overheads [10, 13, 26], which hinder low latency solutions. Furthermore, attempts to reduce latency under virtualization has focused on hardware virtualization capabilities such as SR-IOV [24], which are not available on many edge devices.
- **Energy efficiency:** Communications technology is forecast to consume up to half of global electricity by 2030 [1], but energy efficiency remains an open issue in OS design. For example, polling is often used as an optimization to reduce latency, but it hurts energy efficiency [21]. Virtualization techniques have energy overheads that can be significant depending on the workload [14].

The objective of this research is to (1) understand the overheads and limitations in OS designs, OS interfaces, and virtualization techniques that hinder low latency on edge devices and (2) propose an OS design and interfaces that address the needs of edge computing use cases.

2 RESEARCH QUESTIONS

Q1. What overheads current OS designs, OS interfaces, and virtualization techniques have on latency and energy efficiency that hinders edge computing?

Edge devices are networking intensive because they communicate continuously with low-power sensors, smart devices, and the centralized cloud. The OS interfaces for networking and the kernel network stacks implementing them have various overheads that hinder low latency. For example, the POSIX socket API has overheads from system calls and lack of connection locality [10, 13]. Kernel network stacks that implement traditional socket API have high per-packet processing cost from dynamic memory allocation, memory copies, and kernel data structures such as socket buffers (SKBs) [26]. Furthermore, shared-memory kernel designs have overheads from locking and context switches [6, 18], which further hinders low latency solutions.

Virtualization is proposed as a technique to allow multitenancy and isolation on edge devices [27]. *Hypervisors* provide both multitenancy and strict isolation but they have various performance overheads [8]. *Containers* have less overheads than hypervisors but provide less isolation because all containers share the same host kernel [19]. Container isolation has been proposed to be improved with hardware features such as memory enclaves [2] but recent attacks on them [23, 28] makes their effectiveness unclear. Furthermore, containers are hard to secure because they have access to the full host kernel system call interface by default and require additional security mechanisms to restrict containers to a subset of the system call interface [19]. *Light-weight hypervisors* [19] and *unikernels* [18] are a promising solution for edge computing but their overheads are not well understood nor are they widely deployed because applications need to use OS specific interfaces. Virtualization techniques have energy overheads depending on the workload. For example,

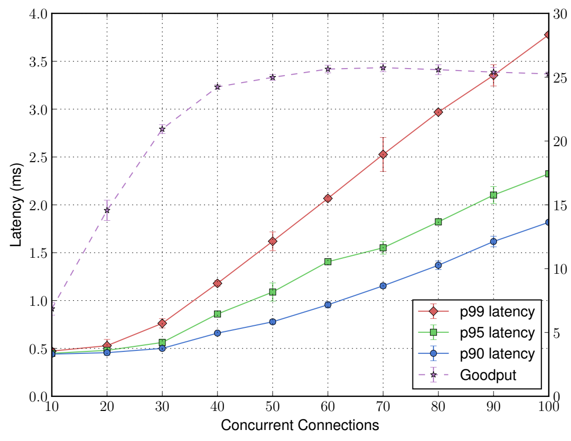

Figure 1: The goodput and the 90th, 95th, 99th percentile latency reported by the Mutilate benchmarking tool with Memcached running on an ASUS Tinker Board single-board microserver. The numbers shown are the average and standard error over 10 iterations of 30 second runs. The average 99th percentile latency is 1.62 ms with a standard error of 0.10 ms when 50 concurrent connections make read requests to Memcached running on 4 threads with steering enabled. The high latency with moderate number of concurrent connections suggests a problem for network intensive edge use cases.

Jin *et al.* [14] report energy overheads between 59% and 273% over bare metal for the KVM hypervisor.

In summary, there are various known overheads in both OS design and interfaces that hinder low latency networking and energy efficiency on edge devices. Furthermore, while virtualization techniques improve security, they have additional overheads to bare metal.

Methodology: We plan to conduct a series of experimental evaluations to measure latency and energy overheads of OS design, OS interfaces, and virtualization techniques to understand if the current state of practice is sufficient for edge computing. We plan to instrument system components and monitor system statistics to quantify the overheads in detail. Specifically, we plan to modify Linux kernel sources and also use a custom system stats monitor that we implemented, ustat [7].

Our initial focus is on NIC-to-NIC latency – the time it takes for request arriving on the NIC receive queue to be processed in userspace so that response is placed on the NIC transmission queue. To measure NIC-to-NIC latency, we plan to use Memcached server [9] and Mutilate benchmark tool [16]. These tools allow us to focus on networking system components but still maintain a realistic experimental setup. Furthermore, we focus on single-board microservers because they are a promising platform for edge computing because of their low-cost and low-energy requirements, and a large existing installation base [20].

Expected outcomes: We have conducted initial experimental evaluation on an ASUS Tinker Board single-board microserver [3] to measure networking latency. The experimental evaluation results are shown in Figure 1. We used Memcached server on an ASUS Tinker Board with Linux kernel 4.14.14-rockchip that is installed with the Armbian 5.38 distribution. We use the Mutilate benchmarking tool on a separate load generator machine. Both systems have a 1 GbE NIC and they were connected back-to-back with an Ethernet cable to eliminate other network functions such as switches from the experiment. Memcached was configured to use 512 MB of system memory and Mutilate was configure to run a read-only workload to avoid measuring Memcached LRU cache replacement performance. The numbers shown are the average and standard error over 10 iterations of 30 second runs.

Latecy is high when moderate number of concurrent connections make read request to Memcached server running on 4 threads with Linux network steering enabled, which is a Linux kernel optimization to distribute TCP/IP stack to multiple cores with single-queue NICs [11]. For example, the average 99th percentile latency is 1.62 ms with a standard error of 0.10 ms with 50 concurrent connections. Measured goodput is well below the bandwidth limitations of 1 GbE NICs and CPU utilization monitoring with `top` utility shows that CPU resources are fully utilized at 40 concurrent connections.

The high latency with moderate number of concurrent connections suggests a problem for edge use cases that need to communicate with low-power sensors, smart devices, and centralized cloud. We expect virtualization techniques to further increase latency on single-board microservers, which lack hardware virtualization features such as SR-IOV. The empirical results indicate that revising OS design and interfaces has high potential to reduce NIC-to-NIC latency on edge devices.

Q2. What OS design and interfaces strike a balance between security, low latency, and energy efficiency?

The goals of security, low latency, and energy efficiency can be conflicting at times. For example, polling is often used as an optimization to reduce latency, but it has energy overheads [21]. Virtualization techniques improve security by providing better isolation, but they have various overheads compared to bare metal [8].

Most commodity OSes use shared-memory kernel design, which has overheads from kernel data structure locking and context switch costs. Multikernels are a kernel design that attempts to address shared-memory kernel overheads by partitioning system resources between CPU cores and using explicit message passing for intercore communication [6]. Unikernels are a single-address space kernel design, which also attempt to reduce overheads in hypervisor environments [18].

Networking performance is particularly important in edge computing. However, commodity OS networking stacks has known overheads that can be attributed to POSIX socket API [13, 24] and its in-kernel implementations [12]. Kernel-bypass networking techniques [26] and clean-slate networking APIs [10], for example, improve networking performance significantly. While backward compatibility is an important factor when considering the OS interfaces for edge computing, there is increasing evidence that the standard POSIX interfaces are no longer adequate for modern applications [4]. Furthermore, the emerging serverless computing paradigm, which is one candidate for edge computing deployment model, makes POSIX API compatibility less relevant because applications are implemented using higher level APIs than what POSIX provides. It is also unclear whether current OS designs and interfaces are adequate to efficiently support serverless computing paradigm at all [15].

Commodity OSes are often implemented in the C programming language. The use of unsafe languages such as C has lead to commodity OSes being targeted for large scale exploitation [5]. However, OSes are starting to be implemented in high-level languages, which exploitation much harder. For example, MirageOS, a unikernel, is implemented in OCaml, a fast and safe language to leverage type system for safety in a single-address space environment [18]. Recently, the Rust programming language has emerged as a high-level systems programming language that follows the zero-cost abstraction principles of C++, but with a much stronger type system [5]. On example of an OS implemented in Rust is Tock, an OS for low-power microcontrollers [17]. Tock effectively leverages the Rust type system to provide isolation on hardware that does not provide isolation support.

Methodology: We plan to explore OS design and OS interfaces by building a prototype OS. We are first targeting virtualized x86 architecture using the virtio device model for simplicity, observability, and ease of development. The main goal is to explore the design of OS interfaces that reduce NIC-to-NIC latencies. We plan to reimplement Memcached to the new OS interfaces and conduct an experimental evaluation to compare its network latency and energy efficiency to Linux. For the former, we plan to use the Mutilate benchmarking tool and for the latter, we plan to use hardware performance counter such as Intel's Running Average Power Limit (RAPL) [22] and an external power meter.

Expected outcomes: We expect to show that an OS design and interfaces that combine multikernel model [6] and kernel-bypass networking [12] reduces NIC-to-NIC latency and improves energy efficiency because (1) system calls are eliminated from network processing fastpath, (2) partitioning system memory all the way in userspace improves performance and scaling, and (3) CPU cores that complete their work are able to enter deep sleep modes without interference from other cores that are working. We also expect to show that using a high-level language for kernel implementation makes the system more secure while retaining the efficiency of kernels implemented in unsafe languages.

REFERENCES

[1] Anders S. G. Andrae and Tomas Edler. 2015. On Global Electricity Usage of Communication Technology: Trends to 2030. *Challenges* (2015).

[2] Sergei Arnautov, Bohdan Trach, Franz Gregor, Thomas Knauth, Andre Martin, Christian Priebe, Joshua Lind, Divya Muthukumaran, Dan O'Keeffe, Mark L. Stillwell, David Goltzsche, David Eyers, Rüdiger Kapitza, Peter Pietzuch, and Christof Fetzer. 2016. SCONE: Secure Linux Containers with Intel SGX. In *OSDI*.

[3] ASUS. 2017. Tinker Board. (2017). Retrieved March 21, 2018 from https://www.asus.com/Single-Board-Computer/Tinker-Board/

[4] Vaggelis Atlidakis, Jeremy Andrus, Roxana Geambasu, Dimitris Mitropoulos, and Jason Nieh. 2016. POSIX Abstractions in Modern Operating Systems: The Old, the New, and the Missing. In *EuroSys*.

[5] Abhiram Balasubramanian, Marek S. Baranowski, Anton Burtsev, Aurojit Panda, Zvonimir Rakamarić, and Leonid Ryzhyk. 2017. System Programming in Rust: Beyond Safety. In *HotOS*.

[6] Andrew Baumann, Paul Barham, Pierre-Evariste Dagand, Tim Harris, Rebecca Isaacs, Simon Peter, Timothy Roscoe, Adrian Schüpbach, and Akhilesh Singhania. 2009. The Multikernel: A New OS Architecture for Scalable Multicore Systems. In *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles (SOSP '09)*. ACM, New York, NY, USA, 29–44. https://doi.org/10.1145/1629575.1629579

[7] Pekka Enberg. 2017. ustat - an unified system stats collector tool. (2017). Retrieved March 21, 2018 from https://github.com/penberg/ustat

[8] Wes Felter, Alexandre Ferreira, Ram Rajamony, and Juan Rubio. 2015. An updated performance comparison of virtual machines and Linux containers. In *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. Institute of Electrical & Electronics Engineers (IEEE). https://doi.org/10.1109/ispass.2015.7095802

[9] Brad Fitzpatrick. 2004. Distributed Caching with Memcached. *Linux J.* 2004, 124 (Aug. 2004). http://dl.acm.org/citation.cfm?id=1012889.1012894

[10] Sangjin Han, Scott Marshall, Byung-Gon Chun, and Sylvia Ratnasamy. 2012. MegaPipe: A New Programming Interface for Scalable Network I/O. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation (OSDI'12)*. USENIX Association, Berkeley, CA, USA, 135–148. http://dl.acm.org/citation.cfm?id=2387880.2387894

[11] Tom Herbert and Willem de Bruijn. [n. d.]. ([n. d.]).

[12] Tomas Hruby, Teodor Crivat, Herbert Bos, and Andrew S. Tanenbaum. 2014. On Sockets and System Calls: Minimizing Context Switches for the Socket API. In *Proceedings of the 2014 International Conference on Timely Results in Operating Systems (TRIOS'14)*. USENIX Association, Berkeley, CA, USA. http://dl.acm.org/citation.cfm?id=2750315.2750323

[13] Eun Young Jeong, Shinae Woo, Muhammad Jamshed, Haewon Jeong, Sunghwan Ihm, Dongsu Han, and KyoungSoo Park. 2014. mTCP: A Highly Scalable User-level TCP Stack for Multicore Systems. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation (NSDI'14)*. USENIX Association, Berkeley, CA, USA, 489–502. http://dl.acm.org/citation.cfm?id=2616448.2616493

[14] Yichao Jin, Yonggang Wen, and Qinghua Chen. 2012. Energy efficiency and server virtualization in data centers: An empirical investigation. In *2012 Proceedings IEEE INFOCOM Workshops*. 133–138. https://doi.org/10.1109/INFCOMW.2012.6193474

[15] Ricardo Koller and Dan Williams. 2017. Will Serverless End the Dominance of Linux in the Cloud?. In *HotOS*.

[16] Jacob Leverich. 2012. Mutilate: high-performance memcached load generator. (2012). Retrieved March 21, 2018 from https://github.com/leverich/mutilate

[17] Amit Levy, Bradford Campbell, Branden Ghena, Daniel B. Giffin, Pat Pannuto, Prabal Dutta, and Philip Levis. 2017. Multiprogramming a 64kB Computer Safely and Efficiently. In *SOSP*.

[18] Anil Madhavapeddy, Richard Mortier, Charalampos Rotsos, David Scott, Balraj Singh, Thomas Gazagnaire, Steven Smith, Steven Hand, and Jon Crowcroft. 2013. Unikernels: Library Operating Systems for the Cloud. In *ASPLOS*. https://doi.org/10.1145/2451116.2451167

[19] Filipe Manco, Costin Lupu, Florian Schmidt, Jose Mendes, Simon Kuenzer, Sumit Sati, Kenichi Yasukata, Costin Raiciu, and Felipe Huici. 2017. My VM is Lighter (and Safer) Than Your Container. In *SOSP*.

[20] Filipe Manco, Joao Martins, Kenichi Yasukata, Jose Mendes, Simon Kuenzer, and Felipe Huici. 2015. The Case for the Superfluid Cloud. In *Proceedings of the 7th USENIX Conference on Hot Topics in Cloud Computing (HotCloud'15)*. USENIX Association, Berkeley, CA, USA. http://dl.acm.org/citation.cfm?id=2827719.2827726

[21] Stephen Marz and Brad Vander Zanden. 2016. Reducing Power Consumption and Latency in Mobile Devices Using an Event Stream Model. *ACM Trans. Embed. Comput. Syst.* 16, 1, Article 11 (Oct. 2016), 24 pages. https://doi.org/10.1145/2964203

[22] Mozilla. 2015. Power profiling overview. (2015). Retrieved March 21, 2018 from https://developer.mozilla.org/en-US/docs/Mozilla/Performance/Power_profiling_overview

[23] Dan O'Keeffe, Divya Muthukumaran, Pierre-Louis Aublin, Florian Kelbert, Christian Priebe, Josh Lind, Huanzhou Zhu, and Peter Pietzuch. 2018. Spectre attack against SGX enclave. (2018). Retrieved March 21, 2018 from https://github.com/lsds/spectre-attack-sgx

[24] Simon Peter, Jialin Li, Irene Zhang, Dan R. K. Ports, Doug Woos, Arvind Krishnamurthy, Thomas Anderson, and Timothy Roscoe. 2014. Arrakis: The Operating System is the Control Plane. In *OSDI*.

[25] Raspberry Pi Foundation. 2012. Raspberry Pi. https://www.raspberrypi.org/. (2012).

[26] Luigi Rizzo. 2012. Netmap: A Novel Framework for Fast Packet I/O. In *Proceedings of the 2012 USENIX Conference on Annual Technical Conference (USENIX ATC'12)*. USENIX Association, Berkeley, CA, USA, 9–9. http://dl.acm.org/citation.cfm?id=2342821.2342830

[27] Mahadev Satyanarayanan, Paramvir Bahl, Ramón Caceres, and Nigel Davies. 2009. The Case for VM-Based Cloudlets in Mobile Computing. *IEEE Pervasive Computing* 8, 4 (Oct. 2009), 14–23. https://doi.org/10.1109/MPRV.2009.82

[28] Adrian Tang, Simha Sethumadhavan, and Salvatore Stolfo. 2017. CLKSCREW: Exposing the Perils of Security-Oblivious Energy Management. In *USENIX Security*.