Modern data centers — the programming platform of choice for increasing numbers of applications and global services — pose a number of challenges for systems software and hardware. With millions of processors, network components, and storage devices data centers have become "exploded computers", requiring innovation across the entire systems stack. Data centers continue to evolve: the number of components is increasing, network link bandwidths continue to scale at a rapid rate, and storage hierarchies are growing deeper and more complex. This evolution puts pressure on data center software and hardware to scale up while delivering performance close to the hardware limit. On the applications side, growth is being fueled by multiple tenants running ever larger and more diverse applications. Existing data center abstractions need to adapt to efficiently support new and larger applications.

My research interests are to address these challenges from a systems perspective, with solutions that span multiple layers of the systems stack, from operating systems through networks down to hardware, but also programming languages and applications.

Thesis Work

My thesis work focuses on a particularly challenging aspect of modern data center design: delivering high-performance network communication through the operating system to applications. Data center network bandwidth is increasing at a rapid pace, while processor performance is only improving at a much slower rate. To manage, operating systems must deliver increasingly efficient software network processing. The operating system must also securely isolate multiple data center tenants, by enforcing resource allocation limits, and ensure correct network protocol operation. Finally, data center network processing needs to be agile in adapting to new protocols, network infrastructure, and application requirements.

On a traditional operating system, an optimized server with a typical remote procedure call (RPC) heavy workload can spend up to 80% of its CPU time executing kernel network processing, making the host network stack a large barrier to scalability. To address this, I propose a new network processing architecture that splits network processing between a programmable hardware network interface card (NIC) and a combination of application and kernel software. The key idea is that all normal, common case processing skips the kernel while still implementing the same functionality as a traditional operating system design. The model is rich enough to implement TCP, the most common protocol used in data centers today. Applications send and receive packets directly through the NIC, bypassing the kernel. The NIC is designed with a special engine to allow it to directly perform most common protocol processing steps. The kernel is left to handle less frequent protocol events and out-of-band protocol processing. I show our architecture can improve throughput by $8.2\times$ and latency by $4.6\times$ for a key-value store relative to Linux.

I start by developing a network interface card hardware model (FlexNIC) that supports reconfigurable processing [8, 7]. Using this hardware model I develop FlexTCP [9], a TCP stack that partitions processing between FlexNIC, the operating system kernel, and applications. Finally, I demonstrate applications can further improve network processing performance by integrating the NIC with application logic [8].

Reconfigurable network interface hardware model. There is a spectrum of capabilities in modern NICs. At a high level, NICs transfer packets between the network and the processor, with a few fixed-function offloads for common protocols built in. For example, with large segment offload, the NIC splits up chunks of TCP data from software into multiple packet-sized segments to reduce software overhead. NICs transfer packets to and from software through descriptor queues that decouple hardware and software, allowing processing to proceed in parallel, minimizing cache misses, and making efficient use of the PCIe interconnect. To enable efficient processing on multiple CPU cores, NICs use multiple send and receive queues and offer steering mechanisms, such as receive-side scaling, to assign incoming packets to receive queues.

Even given all these optimizations, modern NIC capabilities are of limited benefit for the small and frequent RPC interactions typical of most data center communication patterns. This leads many practical applications and services to spend much of their CPU time on kernel processing. While more functionality can be pushed to the NIC, we also need agility: data center services evolve fast enough that any fixed hardware is likely to be out of date before it can be deployed. Instead, I propose a new flexible NIC hardware model, FlexNIC, that exposes a programming interface to NIC processing. FlexNIC allows software to install packet processing, memory transfer, and rate limit rules into the NIC. This tailors NIC operation to handle common case packet processing traditionally done in software, reducing memory and processing overheads. Operating systems and applications alike can use FlexNIC to improve packet processing performance.

Integrated high-performance TCP stack. Packet processing for the TCP protocol in particular is notoriously CPU intensive. When used for small RPCs typical of many services, there are frequent transitions between application and the kernel. Even bypassing the kernel, and thereby sacrificing essential correctness and isolation guarantees, applications still lose thousands of cycles in application TCP handling code.

I design and implement FlexTCP, a new flexible protocol stack that implements full TCP semantics and stronger isolation between connections than found in Linux. TCP functionality is split between the application library, kernel, and FlexNIC. Applications send and receive data directly through FlexNIC, where processing rules implement reliable TCP data transfers and assist the kernel in enforcing congestion control. Kernel software handles infrequent operations, such as opening new connections, and digests congestion feedback out-of-band to adjust NIC rate limits. Using an emulation-based methodology, I show that FlexTCP can increase per-core packet throughput by $8.2\times$ compared to the Linux kernel TCP implementation and $3.2\times$ compared to kernel bypass that ignores resource isolation requirements. FlexTCP, instead, enforces resource isolation and provides tighter performance bounds under load relative to Linux, improving fairness and tail latency by orders of magnitude.

Accelerating applications with customized network processing. Once we accelerate kernel protocol processing, application packet processing can still be a bottleneck. Integrated NIC-software processing can also address this issue. We can offload application-level protocol processing, steer packets to match application locality, and customize the NIC-software interface to streamline the remaining software request processing. In three application case studies, integrated processing reduces application request processing time, improves scalability, and improves cache utilization. When compared to a high-performance user-level network stack, our prototype implementations achieve $2.3\times$ better throughput for a real-time analytics platform modeled after Apache Storm, 60% better throughput for an intrusion detection system, and 60% better latency for a key-value store.

Other Work

In addition to my thesis work, I collaborated with colleagues on a number of additional research directions, including data center network resource allocation, file system crash consistency, and a network stack for taking full advantage of modern NICs.

Network resource allocation with reconfigurable switches. Reconfigurability is not only interesting for end host processing; it is also useful inside the data center network core. Traditional data center switches implement protocol processing using fixed hardware functions. This limits innovation; switch vendors can take years to adopt new protocols before data center operators can evaluate and deploy them. Recent reconfigurable network switches finally make it feasible to short-circuit that delay. Software can configure switches to parse and process custom packet headers using reconfigurable match-action tables in order to exercise control over how packets are processed and routed.

However, flexible switches have limited state, support limited types of operations, and limit per-packet computation in order to be able to operate at line rate. This poses a challenge for implementing existing protocols, which target general hardware or software implementations and assume access to complex computation and data structures. Our work [13] provides a set of general building blocks that mask these limitations using a set of novel approximation techniques. We use these building blocks to tackle the network resource allocation problem within data centers. We show that flexible switches can implement approximate variants of a number of congestion control and load balancing protocols. We show these approximations are both accurate and do not exceed the hardware resource limits of commercially available switches. In particular, we implement RCP on a production switch and show that, as expected, it is significantly faster and has lower-variance flow completion times compared to TCP.

File system crash consistency. Applications depend on persistent storage to recover state after system crashes. However, and perhaps surprisingly, the POSIX file system interface does not define the possible outcomes of a crash; crashes are rare and the wide range of behaviors in existing systems complicates standardization. As a result, it is difficult for application writers to correctly understand the ordering of and dependencies between file system operations, which can lead to corrupt application state and, in the worst case, catastrophic data loss.

In this project, we present crash-consistency models, analogous to memory consistency models, to specify the behavior of a file system across crashes [2]. Crash-consistency models include litmus tests, which demonstrate allowed and forbidden behaviors, and axiomatic and operational specifications. We present a formal framework for developing crash-consistency models, and a toolkit, called FERRITE, for validating those models against real file system implementations. We develop a crash-consistency model for ext4 and use FERRITE to demonstrate counter-intuitive crash behaviors of the ext4 implementation. To demonstrate the utility of crash-consistency models to application writers, we use our models to prototype proof-of-concept verification and synthesis tools, as well as new library interfaces for crash-safe applications.

Flexible network stack for modern network interface cards. In work I started before my thesis work, I (together with other researchers) investigated operating system support for modern NICs. As I mentioned, these NICs offer a wide variety of offload features and configuration parameters. For many of these configuration parameters, however, the optimal setting depends on the specific use-case. They often vary from NIC vendor to NIC vendor, and sometimes from NIC model to NIC model. Existing operating systems fail to efficiently exploit, and to effectively manage, the considerable hardware resources of modern network interface cards.

To fully take advantage of today's and tomorrow's NICs, we develop Dragonet [15], a new network stack design based on explicit descriptions of NIC capabilities. Dragonet represents both the physical capabilities of the network hardware and the current protocol state of the machine as data-flow graphs. We describe NIC capabilities in our new domain-specific language Unicorn [14]. Using the two data-flow graphs, Dragonet embeds the current protocol state into the NIC capability graph, instantiates the remaining functionality in software, and finally uses a search algorithm [11] to find optimal configuration parameter settings for a specified cost function. We show Dragonet can leverage multiple NIC queues and packet steering to significantly improve performance isolation compared to Linux.

Future Work

Network interface hardware. In my thesis work, I have investigated abstractions for a reconfigurable NIC from a software point of view. Emulation results have shown these abstractions reduce software overheads and can improve throughput by up to $8.3\times$. For cloud customers, this improvement in performance and efficiency has the potential to significantly reduce operating costs and to enable new applications, with communication overheads that would otherwise be prohibitive. However, achieving these performance improvements in the data center requires an efficient and cost-effective implementation of a reconfigurable NIC.

Data center operators have started to deploy programmable NICs based on FPGAs or network processors — NPUs, NICs augmented with CPU cores. Results from recent reconfigurable data center switches [3] indicate packet processing on NPUs or FPGAs is an order of magnitude slower and more expensive than using custom ASICs for the same functionality. The general trend towards specialized hardware in data centers [12] further supports this. While I based FlexNIC on hardware constraints in recent reconfigurable data center switches, building a NIC hardware implementation will allow us to quantify the tradeoffs between application performance and hardware complexity.

Programming languages for packet processing. My thesis work advocates splitting processing across the NIC, the OS kernel, and the application. This poses implementation challenges; changes involving more than one component require time-consuming modifications in multiple places and using multiple interfaces. Programming language support would significantly simplify development and improve reliability by enabling program analysis across components. For our case of split processing, the compiler would be responsible to turn a single implementation into code for the NIC, the OS kernel, and the application, based on programmer annotations and compiler optimization passes.

More generally, network protocol implementations are complex, balancing high performance, maintainability, and the need to be careful to design against security vulnerabilities that can be exploited remotely. In my view, a domain-specific language for protocols could bring protocol implementations closer to how protocol designers think about their implementation. For example, as a designer, I often think in terms of protocol scenarios, such as connection establishment, in-order packet reception, and re-transmission of lost packets. A programming language framework that enables correct and secure high-performance protocol implementations could prevent crashes and vulnerabilities, both inside and outside of the data center — especially considering the skyrocketing number of IoT devices connected to the internet.

Data center scale TCP. Data center management, especially for cloud computing, has grown increasingly finer, from hours to deploy on individual machines, to minutes on virtual machines, through seconds with containers, now down to milliseconds with serverless computing [10, 1, 4, 6]. This transition has also increased application density in the data center, in turn, increasing sharing and network load. Individual physical machines need to handle 10s to 100s of thousands of network connections.

This increased density combined with growing network bandwidth not only put pressure on software network processing but also on underlying network protocols. Data center congestion control, for example, has to operate across a wider spectrum: a single link could be shared and fully utilized by only a handful of connections or the same link could be used by 100,000s of connections. Existing congestion control protocols are not designed for and do not behave well over this range of operating conditions. We have observed that classical TCP triggers instability [5] under these conditions in our testbed cluster. As new application requirements and advances in data center infrastructure improve performance further, this and other problems with existing network protocols will become more pronounced. A promising avenue is to observe that existing network resource algorithms are constrained by an assumption of computation in the data path handling code; once we relax that assumption with FlexNIC, a broader range of options becomes available. Understanding and addressing these challenges is essential to support continuing data center evolution and with it the whole cloud ecosystem.

References

[1] Amazon Web Services. AWS Lambda – serverless compute. `https://aws.amazon.com/lambda/`.

[2] James Bornholt, Antoine Kaufmann, Jialin Li, Arvind Krishnamurthy, Emina Torlak, and Xi Wang. Specifying and checking file system crash-consistency models. In *21st International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS, 2016.

[3] Pat Bosshart, Glen Gibb, Hun-Seok Kim, George Varghese, Nick McKeown, Martin Izzard, Fernando Mujica, and Mark Horowitz. Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN. In *2013 ACM Conference on SIGCOMM*, SIGCOMM, 2013.

[4] Google Cloud Platform. Cloud Functions – serverless environment to build and connect cloud services. `https://cloud.google.com/functions/`.

[5] Sergey Gorinsky and Harrick Vin. Additive increase appears inferior. Technical Report TR2000-18, Department of Computer Sciences, University of Texas at Austin, May 2000.

[6] IBM Corporation. Cloud Functions – overview. `https://www.ibm.com/cloud/functions`.

[7] Antoine Kaufmann, Simon Peter, Thomas Anderson, and Arvind Krishnamurthy. FlexNIC: Rethinking network DMA. In *15th Workshop on Hot Topics in Operating Systems*, HOTOS, 2015.

[8] Antoine Kaufmann, Simon Peter, Naveen Kr. Sharma, Thomas Anderson, and Arvind Krishnamurthy. High performance packet processing with FlexNIC. In *21st International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS, 2016.

[9] Antoine Kaufmann, Tim Stamler, Simon Peter, Naveen Kr. Sharma, Thomas Anderson, and Arvind Krishnamurthy. Lightweight date center TCP packet processing. In *Under Submission*, 2017.

[10] Ricardo Koller and Dan Williams. Will serverless end the dominance of Linux in the cloud? In *16th Workshop on Hot Topics in Operating Systems*, HOTOS, 2017.

[11] Kornilios Kourtis, Pravin Shinde, Antoine Kaufmann, and Timothy Roscoe. Intelligent NIC queue management in the Dragonet network stack. In *3rd Conference of Timely Results in Operating Systems*, TRIOS, 2015.

[12] Ikuo Magaki, Moein Khazraee, Luis Vega Gutierrez, and Michael Bedford Taylor. Asic clouds: Specializing the datacenter. In *43rd Annual International Symposium on Computer Architecture*, ISCA, 2016.

[13] Naveen Kr. Sharma, Antoine Kaufmann, Thomas Anderson, Arvind Krishnamurthy, Jacob Nelson, and Simon Peter. Evaluating the power of flexible packet processing for network resource allocation. In *14th USENIX Symposium on Networked Systems Design and Implementation*, NSDI, 2017.

[14] Pravin Shinde, Antoine Kaufmann, Kornilios Kourtis, and Timothy Roscoe. Modeling NICs with Unicorn. In *7th Workshop on Programming Languages and Operating Systems*, PLOS, 2013.

[15] Pravin Shinde, Antoine Kaufmann, Timothy Roscoe, and Stefan Kaestle. We need to talk about NICs. In *14th Workshop on Hot Topics in Operating Systems*, HOTOS, 2013.