Taming Model Incongruities in Networked Systems

Nirav Atre, *Carnegie Mellon University*

1 Overview

We reason about computer systems via models of their behavior: whether implicit, as in *mental* models; or explicit, as in *mathematical* models. These models enable us to, *e.g.*, predict a system's throughput, or make claims about its availability. My dissertation research broadly focuses on a problem I call **model incongruity**: circumstances where our model does not match the actual behavior of the underlying system. As my work shows, model incongruities can not only result in performance issues and correctness bugs, but also expose our systems to crippling adversarial attacks. For example, consider the following scenarios:

(1) Alice is a manager at an e-commerce company. Knowing that revenue is strongly correlated with user experience,[1] she decides to deploy a new caching algorithm to improve page load times. Based on simulations, she predicts a 5% improvement in cache hit-rate by switching to the new algorithm. In deployment, however, Alice finds that although hit-rate did improve with the new algorithm, latency unexpectedly got *worse*, costing the company hundreds of thousands of dollars in lost sales. My SIGCOMM 2020 paper [3] explains that the problem Alice is experiencing arises from *delayed hits*, a phenomenon that subverts the textbook caching principle that improving hit-rate always improves latency. Had Alice deployed my caching algorithm, MAD (§1.1), she could optimize latency directly by accounting for delayed hits in her caching model.

(2) Bob is an engineer at a file hosting company. He notices that customer download speeds are bottlenecked by a firewall that runs deterministically at 40 Gbps, and decides to deploy a different firewall that provides 10× higher (but non-deterministic) throughput. This results in significant end-to-end performance improvements, and customers are thrilled; however, one day, throughput unexpectedly degrades to the point where several customers cannot access their data, costing the company millions of dollars in service-level agreement (SLA) violations. My SIGCOMM 2022 paper [1] shows that such performance degradation can be effected by *algorithmic complexity attacks* (ACAs) resulting from incongruity between Bob's design choice (optimizing common-case performance) and a transient change in workload (from purely stochastic to containing tiny amounts of adversarial traffic). Had Bob leveraged my drop-in scheduling framework, SurgeProtector (§1.2), he could have had the best of both worlds: provable robustness guarantees against ACAs without sacrificing common-case performance.

(3) Charlie is a network engineer at a hospital equipped with robotic surgery and telemedicine (*e.g.*, video conferencing). He is tasked with deploying a hardware packet scheduler to ensure that surgery-related traffic is always prioritized over video traffic. In deployment, a network connection comprised of latency-critical surgery traffic unexpectedly gets dropped, resulting in a botched medical procedure and an expensive lawsuit. My NSDI 2024 paper [2] explains that this failure was the result of incongruity between the number of concurrent connections we see in today's networks, and the maximum number of connections that Charlie's scheduler could possibly handle (due to a *fundamental scalability barrier* imposed by traditional, comparison-based priority queues underlying the scheduler). Had Charlie used my hardware priority queue architecture, BBQ (§1.3), he could have circumvented this scalability barrier, allowing him to provision his scheduler for worst-case connection counts.

Overall, **my dissertation research presents techniques to address a broad range of model incongruities in networked systems**. Some of these incongruities are artifacts of hitherto-unanticipated network effects (*e.g.*, delayed hits); others are caused by transient mismatch between dynamic workloads and Pareto-optimal design decisions (*e.g.*, ACA vulnerabilities); and still others are butterfly effects magnifying small suboptimalities (*e.g.*, scheduling errors). Regardless of the context, the resulting techniques have enabled me to build systems that embody performance [3, 7, 10], scalability [2], and robustness [1]. For my research, I have been awarded a CyLab Presidential Fellowship, named a French-American Doctoral Exchange (FADEx) Laureate in Cybersecurity, and received a Best Paper Award at OSDI 2023. My research artifacts are all open-source, resulting in significant community engagement (with a cumulative 780+ stars and 90+ forks on GitHub), and a Distinguished Artifact Award at OSDI 2023.

[1] In 2008, Amazon reported that every 100 ms of latency cost them 1% in sales. In 2017, Akamai pegged this same estimate at 7%.

My interest in addressing model incongruence is wrought from first-hand experience building (and painstakingly debugging) high-performance systems. To wit, the starting point for my dissertation research was Pigasus [9, 10], a state-of-the-art, FPGA-based intrusion prevention system (IPS) that I helped design and implement during the first two years of my Ph.D. In fact, many of the tools I describe in the examples above are integrated into Pigasus, accompanied by new system models, and proofs of various theoretical guarantees for these models. However, as the examples also demonstrate, model incongruities have serious implications for a *much broader* range of systems. For this reason, I have met with developers at Akamai to discuss how MAD might improve their content distribution systems, worked with Microsoft Azure's host networking team to develop new performance models for their SmartNIC datapath, and met with engineers at NVIDIA, Samsung, Microsoft, and Futurewei to discuss how BBQ could serve as a scalable scheduling primitive for the next generation of their hardware.

In tackling systems problems, my approach involves: (i) a continuous refinement loop of developing a mathematical framework that is tractable to analysis, yet sophisticated enough to capture real-world nuances;[2] (ii) formulating the problem as a precise question within this evolving framework, then designing an idealized oracle to answer it; and (iii) building a practical system (in software, hardware, or both) based on insights and patterns hidden in the oracle's solutions. I am strongly inclined towards this style of research because in addition to culminating in *real systems*, it also enables me to provide *useful theoretical guarantees* for the systems that I build.

I believe deeply in the value of exploring research hand-in-hand with training junior scholars, and I have been immensely fortunate to mentor several remarkably talented students over the years. Benjamin Carleton was a summer research intern at CMU in 2021, and he explored the impact of delayed hits in multi-tier caches; he presented a poster [4] at ACM SOSP 2021, for which he was awarded the first prize in the ACM SIGOPS Student Research Competition (SRC). Erica Chiang was an undergraduate student at CMU, and she worked on characterizing the robustness of job size heuristics used in SurgeProtector; she was a co-author on the SIGCOMM 2022 paper [1], presented a poster [5] on her research at the same conference, won the second (runner-up) prize in the undergraduate SRC, and was subsequently awarded the NSF Graduate Research Fellowship (GRFP). Both Benjamin and Erica are now Ph.D. students at Cornell University. For the past 3 years, I have been mentoring Miguel Ferreira, a Ph.D. student at CMU Portugal. Our work, published at PODC 2024 [6], highlights a severe model incongruity in datacenter networks where a mismatch between topological assumptions and operating conditions can cut *in half* the realized bisection bandwidth compared to what datacenter operators might reasonably expect.

The ever-increasing complexity and heterogeneity of networked systems will no doubt engender even more sophisticated and severe model incongruities, and I am excited about the prospect of applying the systems-plus-theory toolkit I have developed towards building better systems in the face of such uncertainty. In what follows, I describe my dissertation research (§1.1 through §1.3), briefly discuss other networking research to which I have made significant research contributions (§2), and conclude by sketching some future research directions (§3).

1.1 MAD: Taming Delayed Hits

Problem Statement. Caches are at the heart of latency-sensitive systems, and they appear in seemingly every computing context (*e.g.*, microprocessors, distributed file systems, CDN proxies, and software switches). Textbooks tell us that there are exactly two possible outcomes when an object is requested: a low-latency cache *hit*, or a higher latency cache *miss*. In reality, there is a third potential outcome that was rarely acknowledged, and virtually never modeled: a *delayed hit*. Delayed hits occur in high-throughput systems when multiple requests for the same object occur before the object is fetched from the backing store. Left unaccounted, delayed hits subvert our performance expectations in a variety of ways. For instance, I found that they can cause simulators to underestimate average request latencies by up to 63%, lead practitioners to inadvertently deploy the wrong caching algorithm, and, perhaps most importantly, leave significant latency improvements on the table.

Solution. My research [3], which appeared in SIGCOMM 2020, first and foremost highlighted the importance of accounting for delayed hits in modern systems. This work developed a novel theoretical framework to analyze delayed hits, and proved that the seemingly related objectives of hit-rate maximization and latency minimization are *fundamentally different problems* when some hits are delayed; as a consequence, Belady's algorithm, the optimal offline strategy for hit-rate maximization, does *not* guarantee minimal latency in the presence of delayed hits. I designed a new offline caching algorithm, called Belatedly, which computes empirically tight bounds on the optimal latency in polynomial time via a network flow formulation; across a broad range of network settings, Belatedly delivered 9-37% lower average latencies than Belady. Using insights derived from Belatedly's solutions, I then

[2]Finding this sweet spot is perhaps the most challenging part of the process, because incongruities arise precisely when these pieces diverge.

designed Minimum-AggregateDelay (MAD), a simple augmentation to make *online* caching algorithms aware of delayed hits. In simulation, the MAD-augmented versions of these caching algorithms achieved up to 35% lower average latencies compared to their baselines in the highest-throughput settings. Finally, I prototyped a CDN caching node to validate these findings in the context of a real system, demonstrating a 12-18% relative latency improvement using MAD.

Since Publication. More than anything, this work opened a Pandora's box by highlighting how little we understand in the face of systemic complexity, even in a domain as old as caching. Since 2020, there has been a concerted effort to model and account for delayed hits in high-performance systems ranging in scale from a single server (*e.g.*, CXL interconnect) to the Internet (*e.g.*, cross-WAN traffic).

1.2 SurgeProtector: Taming Algorithmic Complexity Attacks (ACAs)

Problem Statement. ACAs are a class of denial-of-service (DoS) attacks targeting variance in the run-time complexity of algorithms and data-structures underlying networked systems. Using a *small amount of carefully-crafted adversarial traffic*, they allow attackers to induce a *large amount of work in the target system*, pushing it into overload and causing it to drop packets from innocent users. ACAs are capable of producing harm that is inordinately higher than the attacker's own resource investment, making them far more potent than run-of-the-mill volumetric attacks. For instance, existing work has shown that an adversary could cripple a 10 Gbps DNS resolver using just 5 Kbps of attack bandwidth by targeting an open ACA vulnerability in DNSSEC (2,000,000× harm). Unfortunately, there was no *general* fix to ACAs; every patch would have to be designed and engineered on a case-by-case basis, entailing intrusive dataplane changes which in turn would require significant amounts of time and effort to implement. Worse, these patches were no free lunch; they required sacrificing some system property or another (*e.g.*, memory efficiency, or average-case performance) in exchange for ACA resilience. This left operators with two equally unappealing alternatives: risk the debilitating effects of ACAs, or spend resources only to end up with a system that has inferior common-case characteristics.

Solution. SurgeProtector [1], which appeared in SIGCOMM 2022, is a drop-in scheduler that *provably* protects network dataplanes against ACAs. It does so by leveraging a simple insight: ACAs are caused by a small sliver of traffic that is extremely expensive to serve; if we strategically permute the order in which incoming packets reach the dataplane, we can selectively prevent attack traffic from ever receiving service. To that end, SurgeProtector interposes a strict-priority packet scheduler between the ingress link and the dataplane, and uses Weighted Shortest Job First (WSJF) to assign packet priorities (i.e., service order). I developed a novel adversarial analysis framework for scheduling algorithms in the single-server setting, and showed that many algorithms that perform well in the context of stochastic workloads (*e.g.*, first-come first-served, fair queueing) could result in *unbounded* harm in the adversarial setting. Conversely, WSJF, the scheduling algorithm at the heart of SurgeProtector, imposes a strict theoretical upper-bound on the worst-case harm an attacker could achieve via ACAs; put simply, *for every 1 bps of innocent traffic the attacker wishes to displace, they must invest at least 1 bps of their own bandwidth into the attack*, significantly reducing the potency of ACAs (from 2,000,000× down to 1×). However, the scheduling algorithm was only part of the story, and I encountered two major challenges in deploying SurgeProtector. First, WSJF requires *a priori* knowledge of a packet's service time, which is generally impractical; to that end, I showed that *estimates* of service time are sufficient (even in an adversarial setting), and it is possible to construct simple, constant-time heuristics to provide such estimates in real systems. And second, the scheduler itself presents a novel attack surface for the adversary; I designed and implemented hardened prototypes of SurgeProtector in both software and hardware, demonstrating that it is feasible to build a scheduler that is itself impervious to attacks. Overall, SurgeProtector represents a systematic, general-purpose approach to ACA mitigation: it comes with theoretical robustness guarantees, is work-conserving (i.e., *all* traffic is served so long as service cycles are available), and entails no changes to the dataplane (i.e., obviates trading-off desirable system properties in exchange for ACA resilience).

Since Publication. SurgeProtector was integrated into Pigasus 2.0 [9], a widely-used open-source IPS. Recent work has also proposed implementing SurgeProtector as a mitigation mechanism for ACAs targeting DNSSEC, a research direction I intend to explore in the future.

1.3 BBQ: Taming Scheduling Errors

Problem Statement. Packet scheduling is the problem of deciding the order and/or time network packets ought to be served or transmitted, and today we have a rich repertoire of scheduling algorithms achieving, *e.g.*, fairness, starvation avoidance, or optimal flow completion time. At the heart of these schedulers is a *priority queue*, which allows the scheduler to map packets' relative order to unique priorities, sort them, and subsequently extract the highest-priority item (i.e., the next packet to schedule) from the queue. However, existing hardware priority queue designs used comparison-based sorting, which entail a fundamental trade-off between performance and scalability. Thus, no existing solution could be practically *deployed* on modern switches and SmartNICs because they either did not scale to realistic flow counts or failed to deliver good throughput, thus requiring an impractical number of replicas. Instead, operators often resort to *approximate* versions of these scheduling algorithms, but these also result in priority inversions (i.e., scheduling errors) and thereby model incongruity for downstream network services.

Solution. BBQ [2], which appeared in NSDI 2024, presents a highly scalable and performant hardware priority queue architecture which enables, for the first time, line-rate packet scheduling with realistic flow counts. BBQ leverages integer priority queueing (IPQ), a sorting technique that circumvents the complexity barrier imposed by comparison-based sorting, thereby breaking the trade-off between scalability and performance. The starting point for BBQ is a Hierarchical Find-First Set (HFFS) queue, an IPQ (used previously in software) which guarantees constant worst-case time complexity for standard heap operations. The key insight of this work is that HFFS queueing is amenable to a *highly efficient, fully-pipelined hardware implementation.* Although efficient pipeline parallelism is key to BBQ's high performance, incorporating this parallelism introduced new challenges in avoiding data hazards (parallel reads and writes to the same data) and ensuring correctness. BBQ uses a variety of architectural techniques (*e.g.*, write forwarding, speculation, and operation coloring) to disentangle parallel access to shared data without sacrificing performance. I implemented BBQ in hardware, demonstrating that it supports 100K+ flows and 32K+ priorities at 100 Gbps line rate (148.8 Mpps, or 3× the packet rate of existing hardware designs) on FPGAs, and 1 Tbps (1,488 Mpps) line rate on ASICs.

Since Publication. BBQ is open-source, and I am exploring the possibility of integrating it into schedulers used in SmartNICs (*e.g.*, at Microsoft), switches (*e.g.*, at NVIDIA), and storage controllers (*e.g.*, Samsung).

2 Other Research

Pigasus (OSDI '20) [10, 9]. Intrusion detection and prevention systems (IDS/IPS) are among the most demanding stateful network functions, and a recurring theme in that literature has been the gap between the capabilities of existing systems and the workloads they need to support. In this work, we designed and implemented Pigasus, a state-of-the-art, FPGA-based IDS/IPS pipeline that achieves 100 Gbps line rate with hundreds of thousands of concurrent flows, all within the footprint of a single server. The key to achieving this is Pigasus's *FPGA-first* architecture. Inverted from the classic offload paradigm, the majority of packet processing in Pigasus is performed via a highly-parallel datapath aboard the FPGA, while the CPU is treated as a secondary compute engine for more complex processing (regular expression search) on a tiny fraction of input traffic (5% on average). Pigasus is open-source, has served as the basis for several other research projects at CMU and beyond, and has been featured on popular technology blogs such as Dark Reading[3] and The Morning Paper[4].

Ensō (OSDI '23, Best Paper Award) [7, 8]. The communication interface between the CPU and NIC has remained unchanged since 1994: exchanging fixed-sized packet buffers. However, both NICs and networked software have evolved considerably since then, leading to a mismatch between this so-called *packetized* interface and its clients. This work proposed Ensō, a new streaming NIC-CPU interface designed to better support how NICs and software interact today. At its core, Enso eschews fixed-size buffers, and instead structures communication as a stream that can be used to send opaque data of arbitrary size. This change alleviates software overheads, reduces PCIe bandwidth requirements, and leads to fewer cache misses. Ensō yields substantial improvements across the board, bolstering throughput for high-performance network applications by 1.5–6×, and reducing latency by up to 43%.

Impossibility Results for Data-Center Routing (PODC '24) [6] Modern DCNs are built atop Clos topologies, which are habitually thought to emulate a *macro-switch*: a single, infinite-capacity, non-blocking interconnect between sources and destinations. However, many of the assumptions underpinning Clos networks (*e.g.*, admission control) are not true in DCNs. In this work, we provide the first mathematical bounds characterizing the gap in throughput and flow demand allocations between practical DCNs and the macro-switch abstraction for these networks. Our results show that, under ordinary (max-min) fairness constraints, the bisection bandwidth provided by the datacenter network can be as low as *half* of what is predicted by the macro-switch abstraction. Additionally, making routing decisions that *explicitly optimize for max-min fairness* can nonetheless result in starvation for one

[3]https://www.darkreading.com/cyberattacks-data-breaches/researchers-say-they-ve-developed-fastest-open-source-ids-ips
[4]https://blog.acolyer.org/2020/11/16/pigasus/

or more connections, a disconcerting thought for any DCN operator. Our findings cast a spotlight on the trade-offs between fairness and throughput, and call into question some common assumptions about the design and evaluation of data-center routing and scheduling.

3 Future Work

My dissertation work opens a broad range of theoretical and practical questions, and so far I have only addressed some of them. Here, I outline some promising research directions that merit further investigation.

Automatically Extracting Adversary-Proof Heuristics. In SurgeProtector [1], I demonstrated that *manually-designed* heuristic functions can be used to estimate packet processing times, even in adversarial settings; however, there are several reasons I believe this process must be *automated* in order to see widespread deployment. First, the space of all possible heuristics is far too rich to search, let alone evaluate, manually [5]. Second, comparing candidate heuristics is compute-intensive.[5] Finally, manually-designed heuristics don't capture implementation details (*e.g.*, inflated runtime due to I/O behavior), which can only be uncovered by automated tools. To that end, an exciting avenue for future work is to automatically extract adversary-proof heuristics for a given program using a combination of domain knowledge, program analysis, and data-driven search (*e.g.*, reinforcement learning).

Workload-Adaptive Dataplanes. Today, the usual process of building systems entails committing upfront to a *single* design that is highly tailored to a *particular* objective (*e.g.*, memory efficiency, or worst-case performance) and is targeted towards a *specific* workload. Unfortunately, this one-size-fits-all strategy (i) leaves significant improvements on the table, whether in terms of performance or attack resilience, and (ii) results in model incongruities when workload characteristics change over time. Run-time reconfigurable hardware architectures (*e.g.*, FPGAs or ASIC overlays) unlock the possibility of segueing between Pareto-optimal designs in response to workload changes. In principle, this would allow designers to optimize for *multiple* disjoint objectives in a workload-dependent manner (*e.g.*, "Optimize X when workload looks like Y"). There are a number of interesting research challenges that must be addressed to make this feasible, but solving them paves the way to building systems that work well in *any* context.

Hardware Packet Scheduling using DRAM. Today, we appear to be the end of the scalability roadmap for hardware packet schedulers: since every queue entry must be stored *somewhere*, we are ultimately bottlenecked by the available hardware memory. For the sake of performance, today's designs exclusively use SRAM, which offers deterministic, single-cycle memory access. Unfortunately, SRAM is a scarce resource, and even highly scalable designs such as BBQ [2] would have an impractically high SRAM footprint for 200K+ connections. However, it is not far fetched to believe that schedulers will some day need priority queueing for 1M+ connections. A natural question then is: *how do we get there?* I believe the key to this lies in offloading queue entries to DRAM, which provides much slower (and non-deterministic) access latencies compared to SRAM, but is a far more abundant memory resource. The clean decoupling between BBQ's priority index structure and its queue memory makes it *feasible* to offload queue memory to DRAM, but there are several challenges that need to be addressed along the way.

Despite my penchant for model incongruities, my ideal world is one where they *don't* exist. While such a panacea is almost certainly beyond reach, I strongly believe that the pursuit of this goal (by systematically identifying, understanding, and mitigating incongruities) will yield significant, tangible benefits for future generations of systems. For instance, *performance SLAs* are virtually non-existent today, ostensibly due to the best-effort model of the Internet. However, this is not fundamental; over the past few decades, network bandwidth and latency have improved remarkably (not just in *absolute* terms, but also in terms of *predictability*), even across the wide area network. Nonetheless, as we have seen, the frequency and severity of model incongruities that appear in practice today would rightly make any operator wary about tendering performance claims. If, instead, model incongruities related to these metrics (a) were exceedingly rare, and (b) had negligible impact, we could finally achieve the long-elusive goal of having Internet services backed by strong Quality-of-Service guarantees. Overall, I am excited about the prospect of building models and systems that bring us closer to this vision.

Finally, while my dissertation research has focused on networked systems and packet processing, model incongruities do not end there; rather, they appear virtually *everywhere* we look. Going forward, I would like to apply my research vision to a broader range of system settings; I am particularly interested in exploring incongruities in the context of cyber-physical systems (*e.g.*, autonomous vehicles), where even tiny discrepancies in model assumptions can have significant implications for real-world safety.

[5]Ultimately, the right choice of heuristic depends on the operating conditions (expected range of traffic load, expected job size distribution for innocent traffic, *etc.*). This in turn entails solving systems of integrals, requiring time and effort only computers can afford.

References

[1] Nirav Atre, Hugo Sadok, Erica Chiang, Weina Wang, and Justine Sherry. SurgeProtector: Mitigating Temporal Algorithmic Complexity Attacks using Adversarial Scheduling. In *Proceedings of the 2022 Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*, New York, NY, USA, 2022. Association for Computing Machinery.

[2] Nirav Atre, Hugo Sadok, and Justine Sherry. BBQ: A Fast and Scalable Integer Priority Queue for Hardware Packet Scheduling. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 455–475, Santa Clara, CA, April 2024. USENIX Association.

[3] Nirav Atre, Justine Sherry, Weina Wang, and Daniel S. Berger. Caching with Delayed Hits. In *Proceedings of the 2020 Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*, SIGCOMM '20, page 495–513, New York, NY, USA, 2020. Association for Computing Machinery.

[4] Benjamin Carleton and Nirav Atre. Delayed Hits in Multi-Level Caches. In *Symposium on Operating Systems Principles (SOSP) (Poster Session)*, 2021.

[5] Erica Chiang, Nirav Atre, Hugo Sadok, Weina Wang, and Justine Sherry. Robust Heuristics: Attacks and Defenses on Job Size Estimation for WSJF Systems. In *Proceedings of the 2022 Conference of the ACM Special Interest Group on Data Communication (SIGCOMM) (Poster Session)*, 2022.

[6] Miguel Alves Ferreira, Nirav Atre, Justine Sherry, and João Luís Sobrinho. Impossibility Results for Data-Center Routing with Congestion Control and Unsplittable Flows. In *Proceedings of the 2024 ACM Symposium on Principles of Distributed Computing*, 2024.

[7] Hugo Sadok, Nirav Atre, Zhipeng Zhao, Daniel S. Berger, James C. Hoe, Aurojit Panda, Justine Sherry, and Ren Wang. Ensō: A Streaming Interface for NIC-Application Communication. In *17th USENIX Symposium on Operating Systems Design and Implementation*, OSDI '23. USENIX Association, July 2023. **Best Paper Award**, and **Distinguished Artifact Award**.

[8] Hugo Sadok, Zhipeng Zhao, Valerie Choung, Nirav Atre, Daniel S. Berger, James C. Hoe, Aurojit Panda, and Justine Sherry. We Need Kernel Interposition over the Network Dataplane. In *Proceedings of the Workshop on Hot Topics in Operating Systems*, HotOS '21, pages 152–158, New York, NY, USA, May 2021. Association for Computing Machinery.

[9] Zhipeng Zhao, Nirav Atre, Hugo Sadok, Siddharth Sahay, Shashank Obla, James C. Hoe, and Justine Sherry. Pigasus 2.0: Making the Pigasus IDS Robust to Attacks and Different Workloads. In *Proceedings of the 2022 Conference of the ACM Special Interest Group on Data Communication (SIGCOMM) (Poster Session)*, 2022.

[10] Zhipeng Zhao, Hugo Sadok, Nirav Atre, James Hoe, Vyas Sekar, and Justine Sherry. Achieving 100Gbps Intrusion Prevention on a Single Server. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, Banff, Alberta, November 2020. USENIX Association.