

Research Statement

Yang Zhou

I am a systems researcher spanning the areas of networking, operating systems, and distributed systems, focusing on datacenter environments. A datacenter centralizes hundreds of thousands of machines with high-speed networks, enables computations over huge amounts of data, and hosts popular applications (e.g., Google search, Netflix streaming, ChatGPT) that impact billions of people’s lives.

In the era of massive-scale data and computations, networking plays a critical role in supporting scale-out datacenter applications running across multiple machines. Ideally, the underlying datacenter infrastructure should be efficient to maintain steady cloud revenues while meeting high user expectations, and be evolvable to handle the increasingly diverse and performance-hungry applications as well as heterogeneous hardware. However, there is a growing **mismatch** between what networking stacks (involving NICs, kernels, transport layers, and threading) provide and what applications need, causing severe efficiency and evolvability problems. For example, the most widely used kernel networking stack prioritizes security and isolation with separated kernel and user contexts, incurring prohibitive CPU overheads; meanwhile, emerging in-memory applications demand ultra-low latency and high throughput, preferring coalescing different contexts but losing isolation. Even though the networking stacks keep evolving, e.g., the modern kernel-bypass RDMA stacks, applications tend to be network-unaware and take networking resources for granted and unlimited, easily causing resource depletion. Such mismatch gets largely exacerbated in large-scale datacenters where networking stacks and applications are usually developed and maintained by disjoint groups of engineers, i.e., network vs. application engineers (due to their growing complexities and industrial organizational structures). This fundamental mismatch causes less efficient use of datacenter resources and hinders the scaling-out of diverse datacenter applications.

My research has focused on bridging the mismatch by **codesigning** low-level networking stacks and high-level datacenter applications from a systems perspective. My codesign aims to realize high efficiency and agile evolvability for datacenter infrastructure, and it innovates in two directions: (1) application-aware networking by restructuring networking stacks based on application needs, and (2) network-aware applications by redesigning applications to be network-efficient. They have borne fruit for many important datacenter applications, including existing ones (e.g., consensus, distributed transactions) and emerging ones (e.g., far memory over networks, microsecond-scale RPCs). My Electrode [1], Dint [2], and Mew [3] safely inject Paxos, transactions, and RPC load balancing logics into the kernel networking stack respectively via eBPF. This not only achieves remarkable performance improvements (by avoiding kernel overheads) but also allows customizing and evolving the kernel stack based on application needs. My Carbink [4] enables network-aware fault tolerance for far memory with high network and memory efficiency, making it practically usable in datacenters with failures being the norm; it also results in a joint patent with Google. Specific to evolvability, my PCAT [5] helps Facebook design an evolvable telemetry system to handle frequent changes in production networks.

My research methodology has been empiricism-guided *measuring, tailoring, and fitting* to analyze, optimize, and implement real-world systems—just like how tailors made clothes in the old times. First, I thoroughly measure to reason through the performance characteristics of various networking stack primitives and complex applications; I also draw on my two-year experiences in Google’s networking and system teams to uncover critical feature requirements in production systems. Second, I aggressively tailor unnecessary or overlapping operations in networking stacks and applications to optimize for high efficiency. Third, I strategically partition and fit applications to the right networking stack primitives to efficiently implement the entire system. This focus on **full-stack optimizations** defines my niche as a systems researcher.

Previous Work

CPU efficient distributed protocols with evolvable kernel networking via eBPF. In-memory distributed protocols such as consensus and distributed transactions are important building blocks for datacenter applications. They require intensive network IOs, while the widely-used kernel networking stack gives low IO performance due to high per-IO CPU overhead. Such mismatch has fostered a popular belief that kernel-bypass is the necessary key to high performance for these protocols. However, kernel-bypass is not a panacea: it essentially trades security, isolation, protection, maintainability, and debuggability for performance; it also burns one or more CPU cores for busy-polling even at low loads, which is usually hard to adopt in public cloud deployments due to per-core pricing [6]. As such,

I revisit the above popular belief and ask: is the current kernel networking stack really ill-suited for CPU-efficient distributed protocols, especially given many kernel advancements over decades?

I first measure the source of the high overhead for kernel networking stacks. When running a prior well-designed transaction protocol under a recent Linux kernel networking stack, I find that networking stack traversing dominates the overhead (64% vs. 16% on context switching and 12% on interrupt handling). This motivates me to aggressively tailor unnecessary components of the stack for specific distributed protocols, trading slight genericity loss for performance boosts. For example, the reliable transport along with complex queue disciplines, which incurs costly `sk_buff` maintenance and packet copies, could be cut; this is because (1) distributed protocols themselves can recover from packet loss with application-level timeouts, and (2) packet loss happens rarely within today’s well-engineered datacenter networks. To realize such tailoring, I leverage eBPF to *safely* offload protocol-specific request processing logic into the early stages of the kernel stack; this avoids going through the full stack and user space, removing most of the kernel overheads.

However, offloading complex distributed protocols into the kernel is challenging, because eBPF has a constrained programming model for kernel safety and liveness. To address this challenge, I strategically partition the distributed protocols to fit frequent critical paths into the kernel for high performance while complex rare paths into the user space for full functionalities. Take the classic Multi-Paxos protocol as an example. Electrode [1] offloads failure-free Multi-Paxos operations of broadcasting, acknowledging, and waiting-on-quorums into the kernel via eBPF; when failure happens, it runs complex failure-handling operations in the user space. I implement such partitioning for Multi-Paxos and two transaction protocols (version-based and lock-based) atop unmodified Linux kernels, and achieve remarkable performance boosts. For instance, Dint [2] for transaction offloading achieves up to $23\times$ higher throughput than kernel networking stacks, and $2.6\times$ higher than a recent DPDK-based kernel-bypass stack [7] (as the eBPF offloads directly work on raw ethernet packets, bypassing any socket connections). Owing to the kernel-friendliness and high performance, my eBPF offloading work has sparked interest in both industry (e.g., Meta, Intel) and academia (e.g., University of Washington, University of Michigan, NYU).

Looking further out, future kernel networking stacks should be evolvable in order to efficiently tackle increasingly diverse applications and heterogeneous hardware. My Electrode and Dint projects already demonstrate that eBPF can provide significant evolvability to kernel networking stacks for specific applications. I am now working on an evolvable generic RPC framework by implementing a reliable RPC transport in eBPF; it leverages efficient `AF_XDP` sockets to direct RPC requests to user-space applications for processing. The evolvability of this RPC framework manifests into three aspects: (1) customizing network transport protocols based on application types (e.g., video), (2) customizing the locations of transport layer offloads ranging from host kernels to SmartNICs (many SmartNICs directly support eBPF), and (3) application-informed request load balancing among CPU cores.

Network and memory efficient fault-tolerant far memory. In a datacenter, matching a particular application to just enough memory and CPUs is hard. A commodity server tightly couples memory and compute, hosting a fixed number of CPUs and RAM modules that are unlikely to exactly match the computational requirements of any particular application. Even if a datacenter contains a heterogeneous mix of server configurations, the load on each server (and thus the amount of available resources for a new application) changes dynamically as old applications exit and new applications arrive. Thus, even state-of-the-art cluster schedulers struggle to efficiently bin-pack a datacenter’s aggregate collection of CPUs and RAM. For example, Google [8] and Alibaba [9] report that the average server has only 60% memory utilization, with substantial variance across machines.

Disaggregated datacenter memory is a promising solution. It pairs a CPU with an arbitrary set of possibly-remote RAM modules, with a fast network interconnect keeping access latencies to far memory small. Much of the prior work in this space [10, 11] has a common limitation: a lack of fault tolerance. Unfortunately, in a datacenter containing hundreds of thousands of machines, faults are pervasive. Without fault tolerance, the failure rate of an application using far memory will be much higher than the failure rate of an application that only uses local memory; the reason is that the use of far memory increases the set of machines whose failure can impact an application.

Achieving both network and memory efficient fault-tolerant far memory is challenging. Conventional memory-efficient fault tolerance scheme applies erasure coding, and stripes a single memory page across multiple remote nodes with RMA-based swapping. For brevity, I use *span* to denote “memory page”. Assuming Reed-Solomon code with 4 data chunks and 2 parity chunks, the conventional scheme requires 6 RMAs per span swap-out and 4 RMAs per swap-in, incurring excessive network IO pressure on the networking stack. In Carbink, I tailor the excessive network IOs by eschewing the span-granularity erasure coding, and instead erasure code at the spanset granularity. A spanset consists of multiple spans with the same size, i.e., 4 data spans and 2 parity spans in our example, and gets swapped out together in a batch. This only requires averagely $(4 + 2)/4 = 1.5$ RMAs per span swap-out and a single RMA per swap-in, significantly improving network efficiency.

However, spanset-granularity erasure coding inevitably incurs memory fragmentation. This is because each span lives in exactly one place (either local memory or far memory), and swapping a span inside a spanset from far memory to local memory creates dead space (and thus fragmentation) in far memory. To address this problem, I design a pauseless defragmentation mechanism running off the swapping critical path, asynchronously reclaiming dead space for later swap-outs in the background. In contrast to the simple span swapping via RMA, this background defragmentation has complex two-phase commit procedures to guarantee crash consistency; therefore, I choose to implement it using more expressive RPCs. Carbink is implemented and evaluated atop Google’s datacenter infrastructure. Compared to a state-of-the-art fault-tolerant design that uses span-granularity erasure coding, Carbink has 29% lower tail latency and 48% higher application performance, with at most 35% higher far memory usage (due to asynchronous memory defragmentation). Carbink also results in a joint patent with Google.

CPU efficient load balancing for microsecond-scale RPCs. Datacenter applications are evolving into microservice architectures, with many small services connected via RPCs to serve user requests. To ensure responsiveness, these services require high throughput and low tail latency, reaching millions of operations/sec per server and microsecond-scale latency respectively. This creates a mismatch between existing RPC frameworks and application demands, in terms of efficiently load balancing microsecond-scale RPCs. Conventional Power-of-Two load balancing probes servers’ load too often (i.e., probing before each RPC) and hurts application throughput, as a load probing consumes comparable server CPUs as a microsecond-scale RPC. My measurement shows that it reduces the goodput (i.e., maximum throughput under tail latency SLO) by half compared to naively dispatching RPCs at random. On the other hand, probing too infrequently will result in stale estimates of load, resulting in suboptimal load balancing, the emergence of hot spots, and violated SLOs. To break this dilemma, Mew [3] tailors unnecessary load probings to just fulfill the staleness requirement that does not degrade tail latency. To do so, Mew performs probing statistically following an optimal probing frequency, obtained by running a gradient descent algorithm on the probing frequency vs. tail latency space.

However, there are more challenges in how to efficiently fit RPC load balancing into RPC frameworks. The first is what load signal to use that is general enough to capture different load levels of servers, and is strongly correlated to future RPC’s tail latency. Instead of using the conventional signal of CPU utilization, I use the low-level thread and packet queueing delay, because the former cannot differentiate between the ideal case of exactly-saturated CPUs and the bad case of overloaded CPUs. The second challenge is how to efficiently implement load probing, especially for kernel networking stacks with high overhead. My solution is leveraging eBPF to directly return load signal values in the kernel, without going through the full kernel stack or user space. With all the above designs, Mew is able to reduce RPC tail latency by $2\times$, while achieving $1.7\times$ higher goodput, over a state-of-the-art solution.

Other datacenter infrastructure research:

Evolvable and memory efficient network telemetry. As modern datacenter networks get larger and more complex, operators must rely on network telemetry systems for continuous monitoring, alerting, failure troubleshooting, etc. However, changes happen frequently in production networks (e.g., modifications to monitoring intent, advances of device APIs), impacting the reliability of network telemetry systems. To handle various changes, I helped Facebook develop their evolvable network telemetry system PCAT [5]. PCAT proposes to use a change cube abstraction to systematically track changes, and an intent-based layering design to confine and track changes. The overall result of PCAT is a change-aware network telemetry system that supports fast-evolving datacenter networks at Facebook.

Network telemetry also requires high efficiency for memory. Telemetry data must be stored in memory, at least temporarily, but memory is a precious resource. Network devices (e.g., NICs, switches) often have less than 100MB of memory; server memory is more plentiful, but should be mostly devoted to applications. My Cold Filter [12], Elastic Sketch [13], Pyramid Sketch [14], and more [15, 16, 17] design memory-efficient probabilistic data structures that can be updated at line rate, have low memory footprints, and high accuracy. At the time of this writing, Elastic Sketch is cited over 400 times by follow-up work across many academic research groups (e.g., CMU, Princeton, University of Pennsylvania, Technion, KTH). Some of them try to further optimize its memory usage, speed, or accuracy; some re-purpose its design for more telemetry tasks; and some leverage its implementation for P4 compiler research.

Secure hardware architecture for SmartNICs. Cloud providers are deploying various SmartNICs with wimpy-yet-power-efficient RISC cores to offload simple network functions such as network virtualization and traffic scheduling. Unfortunately, vast cloud tenants are barred from the efficiency benefits of SmartNICs, because they are not allowed to run their own customized functions on SmartNICs. The root cause is that modern SmartNICs provide little isolation between the network functions belonging to different tenants; these NICs also do not protect network functions from the datacenter-provided management OS running on the NIC. My S-NIC [18] project proposes minimal changes to

SmartNIC hardware, so that datacenters can provide offloaded functions with strong isolation, while preserving most of the total-cost-of-ownership benefits with minimal performance degradations. S-NIC’s designs target various commodity multi-core SmartNICs, and explicitly isolate their IO subsystems and on-NIC accelerators.

Future Research

Building on my past experiences in networking, memory management, OS kernels, and datacenter applications, I am excited to apply my full-stack optimization approach with cross-layer codesign to the following problems.

Deployment-friendly approaches to memory efficiency via malloc queueing. Previous work on increasing memory efficiency is mostly *not* deployment-friendly, requiring modifying either OS kernels [10] or application code and third-party libraries [11]. In search of deployment-friendly approaches, I have a preliminary insight around separating the provisioning of average memory usage and bursty usage: application’s peak memory usage is usually dominated by bursty, large memory allocations (e.g., temporarily loading a large file into memory); if one can time-interleave such allocations from different applications to avoid their memory peaks coinciding with each other, the overall memory provisioning can be reduced, thus improving memory efficiency. One way to implement time-interleaving is overwriting the `Malloc()` function to strategically delay memory allocations, which I believe is far more deployment-friendly than previous work. I call this approach malloc queueing, and it would mostly target batch processing applications whose performance is not sensitive to the incurred memory allocation delays.

eBPF for accelerators and more. eBPF programming language features verified safety and liveness, and has been widely applied to packet processing in kernels and SmartNICs. I intend to extend eBPF to manage heterogeneous hardware accelerators, and build a generic and easy-to-use programming interface between accelerators and application developers. Example accelerators include GPU and FPGA for massively parallel computing, and U2F (Universal 2nd Factor) keys for security. Through verification, this interface would enable strong safety and liveness guarantees for computations running on these accelerators. Besides hardware, I believe eBPF can shed light on more software applications. I intend to explore the following ones: (1) fast task scheduling (e.g., work stealing) for distributed computation framework like Ray [19], and (2) generic shared logs to support various distributed data structures [20]. Both applications would benefit from the efficient network IOs via kernel offloads, and require addressing challenges from the constrained programming model in eBPF.

Resource efficient machine learning. Machine learning (ML) workloads such as the training and inference of Large Language Models (LLMs) are extremely resource-hungry, requiring expensive accelerators like GPUs. I intend to take a full-stack approach to improve the resource efficiency of ML workloads, covering GPU memory efficiency and compute efficiency. One direction is applying far memory techniques to LLM training and inference by swapping to CPU memory. For performance, I plan to codesign far memory swapping with the memory access patterns of LLM weights and key-value cache, e.g., different access frequencies for different weights due to the attention mechanism in LLMs. Another direction is developing a unified GPU memory abstraction that allows easily accessing remote GPU memory over high-speed networks such as NVLink; this kind of GPU memory pooling would help reduce memory stranding and fragmentation caused by dynamic memory allocations in ML workloads. For performance, I plan to codesign such memory pooling with ML workload characteristics, e.g., allowing relaxed consistency. Finally, I am interested in fine-grained GPU kernel scheduling at the microsecond scale possibly with preemption; the goal is to efficiently multiplex GPU compute resources among multiple jobs without losing performance.

Datacenter-scale distributed runtime. A long-term goal of my research is to build a datacenter-scale distributed runtime to not only simplify application development but also increase the whole datacenter efficiency and evolvability. This distributed runtime sits between applications and datacenter resources: (1) for applications, it provides generic and stable interfaces to use compute, memory, storage, and accelerators, and customizable fault tolerance and recovery schemes based on application needs; (2) for resources, it eschews the conventional reservation-based provisioning strategy, and instead provisions resources in a best-effort manner to achieve high resource efficiency.

Today’s datacenters have already provisioned network resources in a best-effort manner, and I plan to expand this strategy to cover more resources like compute, memory, storage, and accelerators. For these new best-effort resources, many networking techniques like congestion control can be applied to enable efficient fair sharing. However, unlike the network resources that are delay-tolerable for applications, other resources especially the memory are not (think of out-of-memory errors). To address this challenge, I intend to leverage techniques like far memory and malloc queueing to create a delay-tolerable memory abstraction, at the cost of lower resource utility than normal memory.

References

- [1] **Yang Zhou**, Zezhou Wang, Sowmya Dharanipragada, and Minlan Yu. Electrode: Accelerating Distributed Protocols with eBPF. In *Proceedings of USENIX NSDI*, 2023.
- [2] **Yang Zhou**, Xingyu Xiang, Matthew Kiley, Sowmya Dharanipragada, and Minlan Yu. DINT: Fast In-Kernel Distributed Transactions with eBPF. Under submission.
- [3] **Yang Zhou**, Hassan Wassel, James Mickens, Minlan Yu, and Amin Vahdat. Mew: Efficient Inter-Server Load Balancing for Microsecond-Scale RPCs. Under submission.
- [4] **Yang Zhou**, Hassan Wassel, Sihang Liu, Jiaqi Gao, James Mickens, Minlan Yu, Chris Kennelly, Paul Turner, David Culler, Hank Levy, and Amin Vahdat. Carbink: Fault-Tolerant Far Memory. In *Proceedings of USENIX OSDI*, 2022.
- [5] **Yang Zhou**, Ying Zhang, Minlan Yu, Guangyu Wang, Dexter Cao, Eric Sung, and Starsky Wong. Evolvable Network Telemetry at Facebook. In *Proceedings of USENIX NSDI*, 2022.
- [6] William Tu, Yi-Hung Wei, Gianni Antichi, and Ben Pfaff. Revisiting the Open vSwitch Dataplane Ten Years Later. In *Proceedings of ACM SIGCOMM*, 2021.
- [7] Joshua Fried, Zhenyuan Ruan, Amy Ousterhout, and Adam Belay. Caladan: Mitigating Interference at Microsecond Timescales. In *Proceedings of USENIX OSDI*, 2020.
- [8] Muhammad Tirmazi, Adam Barker, Nan Deng, Md E Haque, Zhijing Gene Qin, Steven Hand, Mor Harchol-Balter, and John Wilkes. Borg: the Next Generation. In *Proceedings of ACM EuroSys*, 2020.
- [9] Chengzhi Lu, Kejiang Ye, Guoyao Xu, Cheng-Zhong Xu, and Tongxin Bai. Imbalance in the Cloud: An Analysis on Alibaba Cluster Trace. In *IEEE International Conference on Big Data (Big Data)*, 2017.
- [10] Emmanuel Amaro, Christopher Branner-Augmon, Zhihong Luo, Amy Ousterhout, Marcos K Aguilera, Aurojit Panda, Sylvia Ratnasamy, and Scott Shenker. Can Far Memory Improve Job Throughput? In *Proceedings of ACM EuroSys*, 2020.
- [11] Zhenyuan Ruan, Malte Schwarzkopf, Marcos K Aguilera, and Adam Belay. AIFM: High-Performance, Application-Integrated Far Memory. In *Proceedings of USENIX OSDI*, 2020.
- [12] **Yang Zhou**, Tong Yang, Jie Jiang, Bin Cui, Minlan Yu, Xiaoming Li, and Steve Uhlig. Cold Filter: A Meta-Framework for Faster and More Accurate Stream Processing. In *Proceedings of ACM SIGMOD*, 2018.
- [13] Tong Yang, Jie Jiang, Peng Liu, Qun Huang, Junzhi Gong, **Yang Zhou**, Rui Miao, Xiaoming Li, and Steve Uhlig. Elastic Sketch: Adaptive and Fast Network-Wide Measurements. In *Proceedings of ACM SIGCOMM*, 2018.
- [14] Tong Yang, **Yang Zhou**, Hao Jin, Shigang Chen, and Xiaoming Li. Pyramid Sketch: A Sketch Framework for Frequency Estimation of Data Streams. *Proceedings of the VLDB Endowment*, 2017.
- [15] **Yang Zhou**, Omid Alipourfard, Minlan Yu, and Tong Yang. Accelerating Network Measurement in Software. *ACM SIGCOMM Computer Communication Review*, 2018.
- [16] Omid Alipourfard, Masoud Moshref, **Yang Zhou**, Tong Yang, and Minlan Yu. A Comparison of Performance and Accuracy of Measurement Algorithms in Software. In *Proceedings of ACM Symposium on SDN Research (SOSR)*, 2018.
- [17] Zhuochen Fan, Gang Wen, Zhipeng Huang, **Yang Zhou**, Qiaobin Fu, Tong Yang, Alex X Liu, and Bin Cui. On the Evolutionary of Bloom Filter False Positives - An Information Theoretical Approach to Optimizing Bloom Filter Parameters. *IEEE Transactions on Knowledge & Data Engineering*, 2022.
- [18] **Yang Zhou**, Mark Wilkening, James Mickens, and Minlan Yu. SmartNIC Security Isolation in the Cloud with S-NIC. Under submission.
- [19] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I. Jordan, and Ion Stoica. Ray: A Distributed Framework for Emerging AI Applications. In *Proceedings of USENIX OSDI*, 2018.
- [20] Mahesh Balakrishnan, Dahlia Malkhi, Ted Wobber, Ming Wu, Vijayan Prabhakaran, Michael Wei, John D Davis, Sriram Rao, Tao Zou, and Aviad Zuck. Tango: Distributed Data Structures Over a Shared Log. In *Proceedings of ACM SOSP*, 2013.

[This page intentionally left blank.]

Teaching Statement

Yang Zhou

I greatly enjoy the rewards of teaching and mentoring students. For me, the rewards consist of two significant parts: (1) the pride and fulfillment when my teaching helps students carry out their studies smoothly and when my mentored students grow into independent researchers, and (2) the interesting future research directions inspired or confirmed during teaching and mentoring. Driven by these rewards, I have taught as a teaching assistant and as a small-group “supervisor”, and mentored four undergraduates and five junior PhD students in their research. Based on my research background, I am qualified to teach undergraduate courses of computer networks, operating systems, distributed systems, and algorithms and data structures, and graduate courses of data center networking and dataplane operating systems (detailed later).

Mentoring Experience and Methodology

I have mentored four undergraduates on system research, and informally mentored five junior PhDs on their research ideas, internship applications, and study experience at Harvard. Among the four undergraduates, one (Zezhou Wang) published an NSDI’23 paper with me and went to University of Washington (UW) as a system PhD; two of them (Xingyu Xiang and Matt Kiley) co-authored an NSDI’24 submission with me, and are about to apply for system PhDs as well as the rest one. Such mentoring brings me enormous pride, e.g., seeing Zezhou gets into the UW PhD program. It inspires my future research—working with Zezhou on eBPF sparks two follow-up projects: one has become the NSDI’24 submission, and another is showing promising results. Below I summarize my mentoring methodology:

- *Building students’ confidence.* It is well-known that confidence is crucial for students, but how to build their confidence is challenging. One way I find helpful is respecting students’ thoughts by giving them enough freedom to try their thoughts while keeping an eye on the big agendas and goals. Another way is connecting them to experts upon entering a new field, avoiding the steep learning curves overwhelming or destroying their confidence. The experts, who could be the mentors themselves, would point out the proper materials or steps for quick ramp-ups.
- *Encouraging students to form their own opinions and tastes.* I encourage and anticipate students to form their own opinions about systems, develop their own tastes on promising research problems, and stick with them. I do not worry too much about if students’ opinions/tastes are wrong, as once they go deep into specific directions they believe, they will learn extensive experiences and insights to refine their previous opinions/tastes.
- *Collaborating widely.* Wide collaboration across industry and academia is especially beneficial for practical system research, and mentors should play the important role in connecting students with proper researchers in the wild. For example, my fault-tolerant far memory project Carbink was collaborated with Google via my co-advisor’s connections, and then inspired by Google’s desire for high availability. However, collaborating with industry usually requires teasing out real research challenges, while not being misled by massive engineering details; advisors should leverage their experience to help students (especially junior PhDs) navigate efficiently in this space. For another example, my eBPF-for-Paxos project Electrode would not be possible without the collaboration with Sowmya Dharanipragada who is a distributed system PhD at Cornell. Going forward, I would like to expand collaborations to theory, machine learning, architecture, programming languages, etc.

Teaching Experience and Philosophy

System course teaching: I was the teaching assistant (TA) for a computer system course, the Harvard CS145 Networking at Scale, along with an undergraduate TA. This course features eight P4-switch related projects, three of which are designed and developed by me including detailed guides and skeleton code. I held three one-hour sections covering network programming, background knowledge for projects, and handy tools for developing and debugging. Other duties include holding weekly office hours, answering students’ questions on forums, and grading projects. In addition to TA, I also had a guest lecture experience at UC Berkeley on far memory techniques in data centers, mainly facing junior graduate students from architecture areas. I started from common and accessible facts like resource utilization and

DRAM prices, then explained why data center operators have an interest in far memory, and finally discussed my work in this space.

Algorithm course teaching: I was the small-group supervisor for the Algorithm Design and Analysis course at Peking University as an undergraduate. This role requires supervising around 14 students in small classes, giving recitations, teaching advanced algorithms and data structures, preparing new problem sets and quizzes, and grading, all on a weekly basis. I extensively introduced non-textbook topics related to my undergraduate research of probabilistic data structures. Although time-consuming, being such a supervisor is truly gratifying, especially when students understand my research and try various optimizations as their final course projects. One student (Yicheng Jin) in my small class is now pursuing a computer science PhD at Duke University.

Introductory teaching: I taught non-CS audiences about the Internet from a computer science perspective during the English Language Program at Harvard. It was a slightly difficult yet fun experience especially when I told the audience that Internet data is transmitted in small packets: they were shocked and immediately asked why, and then I gave them detailed yet understandable explanations until they grasped the design philosophy behind it. This experience gave me a good sense of how to teach introductory courses in the future. Below I summarize my teaching philosophy:

- *Building safe and inclusive environments.* Students in the same class usually have different prior knowledge; thus it is important to create safe and inclusive environments to make students feel they are welcome to ask both the simplest questions and challenging ones. I got such first-hand experiences when I took my co-advisor James Mickens' CS263 System Security course: it has the most open class environment I have ever seen because of James' unique humor, and students ask so many interesting questions during the class. As a result, I personally learned so much security knowledge, though my research is on networked systems.
- *Focusing on hands-on experiences.* I believe the best way to learn computer systems is through reading, running, debugging, and hacking well-written codebases in a hands-on manner. My personal experience in learning dataplane operating systems exactly follows this pattern: after reading relevant papers, I could not understand how specific designs get implemented and contributed to the final performance; then I decided to read the codebase of a dataplane OS called Caladan [1], and run and debug it; finally, I built my own research prototype atop it. After the process, my understanding of dataplane OSes became much clearer, and I gradually began appreciating the merits of various designs in this space. For future system courses I teach, I would like to incorporate well-written teaching systems, such as the WeensyOS [2], into my agenda to help students gain hands-on experiences.
- *Promoting critical thinking on the pros and cons of techniques.* I learned this from the Harvard CS260r Projects and Close Readings in Software Systems—Serverless Computing by Eddie Kohler, where he discussed serverless computing research from a traditional system research perspective. He showed impressive critical thinking on the pros and cons of serverless computing, and helped us grasp the real novel components of this paradigm without deifying any new terms. I plan to apply a similar philosophy to my teaching, encouraging students to critically think about new techniques around us, such as the emerging LLM techniques.

Course plans: In addition to the aforementioned undergraduate courses based on textbook knowledge, I would like to hold two advanced graduate courses and a seminar course based on my research:

- *Data center networking:* I will discuss how modern data centers design and build high-performance network fabrics including topology, routing, congestion control, fault tolerance, load balancing, etc.
- *Dataplane operating systems:* I will discuss how the OS evolves to keep up with the fast hardware in data centers, including user-space networking, efficient threading, light-weight isolation, etc.
- *System seminar course:* I will invite a broad set of system researchers from both academia and industry to give talks on various system research topics, and foster potential collaborations with students.

References

- [1] The Caladan authors. Caladan opensource. <https://github.com/shenango/caladan>.
- [2] Eddie Kohler. Harvard CS61 Systems Programming and Machine Organization (2023): WeensyOS. <https://cs61.seas.harvard.edu/site/2023/WeensyOS/>.

Diversity Statement

Yang Zhou

I view DEI as the basic soil for growing humanity and excellence in society, including the academic community; it is about the daily respect for people regardless of their self-identifications, and self-introspection on “whether I want to be treated like what I treat others”. Everyone has the duty to foster DEI in her/his surroundings, because that eventually determines how the society will treat them in one day. Here, I would like to sample my and my family’s experiences of being underrepresented due to educational background, language, political affiliation, and ethnic origin, to motivate how I grow awareness of the challenges faced by underrepresented populations and the importance of DEI, and possible ways to foster DEI—some I have adopted and some I plan to do.

I am a first-generation college student, so my parents could hardly give me advice on how to succeed in college and in my PhD studies. However, I was lucky to receive tremendous emotional support from them. I was also fortunate to receive academic mentorship from a variety of professors and student peers. Thus, I am proud to be a faculty job applicant today, and I look forward to creating a sharing and inclusive environment in the classroom and in my research group.

As a first-generation immigrant to the US, one of the first challenges that I faced was mastering the English language. At Harvard, I greatly benefited from the university’s English Language Program (ELP), which offered weekly lectures by experienced English teachers, and recruited native English speakers from the university to serve as language partners. The ELP experience showed me how community building is a critical aspect of helping students integrate into challenging environments. As a professor, I hope to make students aware of programs like the ELP that target specific barriers to students’ success (e.g., language issues, or a lack of adequate high school preparation for college-level classes).

Fifty years ago, my uncle was denied admission to his dream civil aviation university, despite his excellent academic performance and physical fitness. He was rejected because his father (my grandfather) was a combat medic for the Chinese Nationalist Party—the party who had fought with the Communist Party of China that founded the People’s Republic of China. Such political discrimination prevented a whole generation of my uncles from participating in activities that were even slightly related to military service. I was told this experience at a very young age; thus, I have always known that the political environment of the past can influence personal outcomes in the present.

My mother and her family are Hui Chinese, one of the ethnic minorities that comprises 0.79% of the total Chinese population. Being an ethnic minority in China often results in discrimination by the majority Han population. For example, a popular stereotype is that Hui Chinese are thieves. Fortunately, my parents always taught me to not treat people by their ethnicity, race, or religion. As a result, I am always conscious of potential biases that may impact my interactions with others, and I hope to support DEI principles as a professor.

My Past Contributions to Advancing DEI

I have participated in various activities that supported DEI via mentoring and teaching.

Mentoring: During the summers of 2022 and 2023, I mentored four undergraduate students for research internships at Harvard: three came from non-US schools, with two being in the US for the first time. To help the students get familiar with systems research (and life in the US), I held weekly meetings with each student, talking about not only research but also various cultural acclimation challenges that I had experienced during my own PhD. At the time of this writing, one of them has co-authored a paper with me that was published at a premier system conference. This student was also accepted to the University of Washington as a computer science PhD student. The other three students have also decided to apply to systems PhD programs, including one that was hesitating for a long time before working with me. I also consistently (monthly) shared my research and internship experiences with five junior PhD students over the past two years. All of them are non-native English speakers and are non-white.

Occasionally, I received email inquiries from PhDs who are in other research areas or from underrepresented minorities; I often scheduled one-to-one meetings to learn about their difficulties or puzzles. For example, Jessica Quaye, originally from the Republic of Ghana in West Africa, was interested in system research though she is in an architecture research group. I had long meetings with her both in person and online, and introduced her to my co-advisor Minlan Yu to identify potential opportunities for collaboration and advising.

Besides one-to-one mentoring, I also participate in one-to-many panels to share my research experience with junior system PhDs. For example, I was a panelist for the “Getting started with systems research” panel [1] organized by Students@Systems in 2022. The video recording for the panel is freely accessible online to help systems PhD students regardless of their university or physical location.

Talking: Academic networking (e.g., talking to peer researchers at conferences) is crucial to the success of a PhD student. However, junior graduate students are often afraid of professional networking, e.g., due to fears about having little experience or being from less prodigious schools. However, I vividly remember how, at a conference, James Mickens (one of my co-advisers) stood in front of the door of a breakout room and publicly said “I am James, a Professor at Harvard, and you are welcome to talk to me!” This event inspired me to proactively interact with junior students during conferences, to talk about mutual research interests and identify potential collaboration opportunities. I also like to encourage poster presenters for their research, especially when there are no people who are currently engaging with their posters.

I also talk to undergraduates and high school students regarding computer science research. For example, in October 2022, I gave a research talk at a Harvard AM/CS/EE PhD recruitment event (accessible to all US universities) which targeted students “that hold membership in an underrepresented and/or historically minoritized group in STEM.” In 2022, I also gave talks at the Harvard SEAS Undergraduate Research Open House and the SEAS Research Showcase, targeting Harvard freshman and sophomore undergraduates. These talks were well-received, with several undergraduates in the audience later contacting my research lab to learn more about participation opportunities; I still mentor one of these undergraduates. Going back to the time when I was an undergraduate, I had the privilege to talk to juniors in my alma mater high school on why a computer science major is a good college major. Some of these students still contact me for advice.

Teaching: I make an explicit effort to help students with little prior exposure to computer science, and I try to promote inclusiveness during teaching. When I was the small-group “supervisor” for the Algorithm Design and Analysis course at Peking University, I realized that some students lacked high school experience with programming contests; these students often found it hard to catch up with peers who did have this experience. To help them, I wrote step-by-step, thorough explanations for the algorithms discussed in class, and I handed out these explanations after class. When TA’ing a course at Harvard University, I answered all questions that appeared in the Ed forum, no matter whether the questions were anonymous or not, to keep everyone’s learning progress on track.

My Future Plans for Fostering DEI

Going forward, as a faculty member, I plan to take the following actions:

- *Advising:* Actively recruiting underrepresented students, being attentive to any anti-DEI atmosphere in my research group, and explicitly adopting counter-measures to foster DEI with affirmative actions.
- *Connecting:* Reducing the barriers of students finding research opportunities by organizing mutual-connecting programs like UCB DARE [2]—matching students with faculty members for research.
- *Teaching:* Being attentive to any students with weaker prior knowledge in my classes, and helping them build confidence with support on a case-by-case basis.
- *Daily life:* Being kind to people I meet, no matter their age, color, disability, gender, ethnicity, politics, religion, education, language, and more. I believe “kindness is the ultimate nobility” [3].

References

- [1] Student@Systems. A panel on “Getting started with systems research”. <https://students-at-systems.org/pages/events/getting-started-with-systems-research.html>.
- [2] UC Berkeley. DARE: Diversifying Access to Research in Engineering. <https://dare.berkeley.edu/>.
- [3] Amin Vahdat. SIGCOMM Lifetime Achievement Award 2020 Keynote (48m44s): kindness is the ultimate nobility. https://youtu.be/Am_itCzkaE0?t=2924.