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 applications. 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. 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. This focus on **full-stack optimizations** defines my niche as a systems researcher.

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. 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?

With further investigation, I find that the networking stack traversing dominates the kernel overhead. This motivates me to aggressively tailor unnecessary components of the stack for specific distributed protocols, trading slight genericity loss for performance boosts. To do so, 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. My Electrode [1] project implements such partitioning for the Multi-Paxos protocol, while my Dint [2] implements two transaction protocols (version-based and lock-based); both achieve remarkable performance boosts. For instance, Dint achieves up to 23× higher throughput than kernel networking stacks, and 2.6× higher than a recent DPDK-based kernel-bypass stack Caladan (as the eBPF offloads directly work on raw ethernet packets, bypassing any socket connections). 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).

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. 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 has a common limitation: a lack of fault tolerance, while failures in datacenters are pervasive.

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". Such stripping incurs excessive IO pressure on the networking stack, as each stripe of a span will require an RMA. In Carbink, I tailor the excessive network IOs by erasure coding a batch of spans (i.e., spanset), reducing the number of RMAs per swapping to only one on average. However, spanset-granularity erasure coding inevitably incurs memory fragmentation. To address this problem, I design a pauseless defragmentation mechanism running off the swapping critical path. 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).

CPU efficient load balancing for microsecond-scale RPCs. My Mew [3] project studied how to efficiently load balance microsecond-scale RPCs among multiple servers, under the emerging microservice architectures. Conventional Power-of-Two load balancing probes servers' load too often (i.e., probing before each RPC) and hurts application throughput. On the other hand, probing too infrequently will result in stale estimates of load, causing suboptimal load balancing 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; it performs probing statistically following an optimal probing frequency, obtained by running a gradient descent algorithm on the probing frequency vs. tail latency space. To efficiently implement load probing, especially for kernel networking stacks with high overhead, Mew leverages eBPF to directly return load signal values in the kernel, without going through the full kernel stack or user space.

Other research. My prior research helped Facebook develop their evolvable and change-aware network telemetry system PCAT [5], designed memory-efficient probabilistic data structures with high accuracy and speed [6, 7, 8], and proposed minimal changes to SmartNIC hardware to enable sharing in multi-tenant public cloud [9].

Future research. 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 example direction is to apply far memory techniques to LLM training and inference by swapping to CPU memory. For performance, I plan to codesign such 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.

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 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] **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.

[7] 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.

[8] **Yang Zhou**, Omid Alipourfard, Minlan Yu, and Tong Yang. Accelerating Network Measurement in Software. *ACM SIGCOMM Computer Communication Review*, 2018.

[9] **Yang Zhou**, Mark Wilkening, James Mickens, and Minlan Yu. SmartNIC Security Isolation in the Cloud with S-NIC. Under submission.