Research Statement

Shawn Shuoshuo Chen

Numerous indispensable applications, from Internet services to scientific computing, run in data centers and HPC clusters. These applications require the network to provide all host computers with static, uniform bandwidth to meet their performance expectations. Unfortunately, due to the rising data volume of applications like AI and video streaming, building high-speed networks with uniform bandwidth now incurs prohibitive costs and power consumption. **Reconfigurable networks** overcome this challenge by providing time-varying, nonuniform bandwidth between hosts. This paradigm shift stems from the insight that application bandwidth demands fluctuate over time, yet conventional networks are provisioned for peak demands. Enabled by optical circuit switching technologies, reconfigurable networks can dynamically adjust bandwidth between hosts at fine time granularities, according to real-time application demands. With reconfigurability, networks can deliver the promised performance without guaranteeing bandwidth uniformity at all times. However, the current network protocol stack is not ready to take advantage of reconfigurability.

My PhD research focuses on redesigning the network stack to achieve higher scalability and efficiency through reconfigurable networks. As my work demonstrates, different parts of the current network stack face inefficiencies in reconfigurable networks. The reason is that this dynamic infrastructure challenges the long-held assumptions of bandwidth stability and uniformity on which the stack is built. I have devoted systematic efforts to redesigning three core components of the stack: transport, routing, application. My approach coordinates between hosts and the network, software and hardware, and different network layers.

- **Transport.** Existing transport protocols operate suboptimally in reconfigurable networks because they are not designed to handle the rapidly changing network bandwidth. My work [1] has addressed this issue by coordinating between the hosts and the network.
- **Routing.** Routing protocols cause link congestion and packet loss in reconfigurable networks because they route traffic inefficiently under bandwidth volatility. My work [2] has mitigated this problem by coordinating between the routing software and hardware.
- **Application.** Reconfigurable networks make applications' performance more sensitive to their placement in the network due to bandwidth nonuniformity. My work (in submission) [3] places applications in the network with good performance by coordinating between the application and the physical layer.

My research has produced open source tools and systems [4, 5, 6] that are being used by several groups at UT Austin, Max Planck Institute for Software Systems, Cornell and Purdue. I also closely collaborate with both academia and the industry, including researchers and practitioners at UC San Diego, MIT, Harvard, Google, and Microsoft. In addition, I have been awarded a Lambda research grant, an Argonne Leadership Computing Facility Director's Discretionary award, and a Google Cloud research grant.

TDTCP: Efficient Transport in Reconfigurable Networks

Transport protocols, such as TCP, operate under a common belief that a network path has a stable maximum bandwidth. They are designed to discover this maximum bandwidth and transfer data at a matching rate. Reconfigurable networks invalidate this belief. Bandwidth changes drastically between hosts in very short times—much faster than the protocols can adjust their sending rates. As a result, these protocols underutilize the varying path bandwidth.

My SIGCOMM 2022 paper [1] has addressed this inefficiency with a new Time-Division TCP (TDTCP) protocol. TDTCP achieves efficiency gains by coordinating between the host and the network. A key insight is that the network—specifically, the optical circuit switches (OCSes) responsible for implementing bandwidth changes—receives a configuration in advance that specifies the future bandwidth per host and the exact times of bandwidth transition. By exposing the configuration to all hosts, TDTCP stays informed of the upcoming changes. Furthermore, TDTCP's optimized internal state machine can maintain two sending rates: one for the current bandwidth and one for the future. Once the future bandwidth becomes active, TDTCP instantly starts sending at the future rate. This mechanism avoids the slow and reactive bandwidth discovery process. I have fully implemented TDTCP in the Linux operating system. It improves the throughput of data transfer over three widely used transport protocols, i.e., DCTCP, CUBIC, and Multipath TCP, by 24–41%.

PreciseTE: Efficient Routing in Reconfigurable Networks

Routing protocols balance the load on different network paths by distributing data packets across these paths with a set of desired probability ratios. Existing routers approximate this distribution by replicating routing entries for each path in a small SRAM memory according to the ratios. Each packet then chooses a path from the replicated entries uniformly at random. However, in reconfigurable networks, this approach approximates target traffic distributions with poor precision, leading to congestion and packet loss. The root cause is that volatile bandwidth in reconfigurable networks results in ratios that are too diverse and skewed for the limited router memory to store.

My work [2, 7] is the first to quantify the significant impact of imprecise traffic distribution in large-scale reconfigurable networks. My NSDI 2024 paper [2] has proposed an algorithm to tackle memory-efficient weighted routing by coordinating between the routing software and hardware. This algorithm rounds the ratios intelligently to avoid excess replication while minimizing the divergence between the resulting and desired traffic distributions. Specifically, this is achieved by identifying and rounding multiple similar ratios to a single shared ratio, which only needs to be stored once. The algorithm also adaptively controls the aggressiveness of rounding and ratio merging based on the available memory space reported by the router hardware. Awareness of the hardware status allows the algorithm to reclaim more precision once the memory frees up. My algorithm has been demonstrated to improve traffic precision by 10× over the current approach and has been deployed in Google's production networks.

RFold: Efficient Application Placement in Reconfigurable Networks

ML jobs need contention-free communication to avoid delaying their bulk synchronous computation. Current static networks substantially overprovision their bandwidth to avoid contention. Reconfigurable torus networks (e.g., TPUv4 [8]) instead specifically optimize the topology for the near-neighbor ML communication pattern. Nonetheless, compared to static networks, a job's performance is more sensitive to its placement in reconfigurable networks. Jobs placed on non-neighboring hosts must route through intermediate hosts to communicate, subjecting them to contention with other traffic and hence unpredictable slowdown.

My work, RFold (in submission) [3], shows that applications can take advantage of reconfigurability to achieve contention-free placement in this specialized network. Specifically, RFold leverages cross-stack coordination between the application and the physical network layer. On the application layer, RFold represents a job's communication pattern as a graph, where communication between hosts corresponds to edges in the graph. To place a job, RFold first generates isomorphic transformations of the graph. It then searches for a contiguous mapping from any of the graphs to the underlying topology. On the physical layer, when no contiguous placement is found, RFold identifies a set of fragmented available hosts and shuffles/reconfigures their positions in the torus to form a contiguous available region that can accommodate the application-layer graph. Evaluation results show that RFold improves cluster utilization by 57% and reduces job completion time by 11× relative to existing methods.

Future Research

My vision is to realize unprecedented scalability and efficiency across a broad range of systems by leveraging reconfigurable connectivity. I plan to integrate reconfigurability into the end hosts, and co-design hosts and the data center/HPC network to achieve end-to-end reconfigurability. Below, I discuss my future directions.

Redesigning the operating system for reconfigurable I/O in a host. For decades, different resources in a host computer, including CPU, GPU, NIC, and storage, have been interconnected by the PCIe bus. We now face a challenge similar to that in data center/HPC networks: the static, uniform bandwidth of PCIe is no longer sufficient to support the I/O demands between these resources. Recent advances in silicon photonics [9] have integrated minuscule optical circuit switches (OCSes) and transceivers into a host. This integration creates reconfigurable I/O in a host where bandwidth can be dynamically allocated between different resources to facilitate data movement. To harness the power of reconfigurability, the host operating system stack—spanning device drivers, memory management, and the communication library—needs to be redesigned. For instance, when a process reads/writes a device memory, it first needs to call the communication library, which then programs the OCS to ensure an I/O path with sufficient bandwidth is created. Lastly, the memory management module and the device driver must coordinate to move data using the created I/O path.

Enabling end-to-end reconfigurability via host-network co-design. When a resource on one

host communicates with another resource on a remote host (e.g., remote memory access), communication bottlenecks can occur at any of the following locations: the inter-host network path or the I/O paths inside the two host computers. My work on reconfigurable networks has offered a solution to handle bottlenecks on the network path. Reconfigurable I/O will be able to handle the host bottlenecks. However, these two solutions work independently, making it challenging to completely eliminate bottlenecks. To this end, I plan to co-design the reconfigurable network with reconfigurable host I/O so that we can establish a bottleneck-free end-to-end data path between resources. There are plenty of interesting problems to be addressed. For example, the hosts and the data center/HPC network have different control loops to implement reconfiguration, which must be synchronized to support end-to-end bandwidth reservation.

ML-augmented design for reconfigurable systems. System design requires human developers to analyze workload traces, make assumptions and simplifications, and create heuristics and algorithms. This human-driven approach makes developing reconfigurable systems a significant undertaking. Even worse, it is untenable to keep up with the evolving workloads and growing hardware heterogeneity in today's reconfigurable systems. I plan to use ML to augment the current design approach. Here are two examples.

Example 1. Synthetic data generation is a technique that trains generative ML models to produce realistic yet hypothetical traces. It enables us to study many what-if questions prior to finalizing design decisions, e.g., how would TCP behave if all data packets are transferred in highly synchronized microsecond bursts over an end-to-end reconfigurable path? Drawing on my experience with traces at different system layers, I plan to build synthetic trace generators to drive system design and optimization.

Example 2. Modern AI tools can generate algorithms that match and sometimes outperform human designs. These powerful tools can be used to autonomously design different components of the network and system stack, e.g., the job placement algorithm. More importantly, they can be invoked frequently—whenever a job with new communication pattern is introduced or the network changes to another topology.

References

[1] Shawn Shuoshuo Chen, Weiyang Wang, Christopher Canel, Srinivasan Seshan, Alex C. Snoeren, and Peter Steenkiste. Time-division tcp for reconfigurable data center networks. In *Proceedings of the ACM SIGCOMM 2022 Conference (SIGCOMM 22)*, Amsterdam, Netherlands, 2022. Association for Computing Machinery.

[2] Shawn Shuoshuo Chen, Keqiang He, Rui Wang, Srinivasan Seshan, and Peter Steenkiste. Precise data center traffic engineering with constrained hardware resources. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, Santa Clara, CA, 2024. USENIX Association.

[3] Shawn Shuoshuo Chen, Daiyaan Arfeen, Minlan Yu, Peter Steenkiste, and Srinivasan Seshan. Toward co-adapting machine learning job shape and cluster topology. https://arxiv.org/abs/2510.03891, 2025. arXiv:2510.03891.

[4] Shawn Shuoshuo Chen. TDTCP. https://github.com/shuoshuc/TDTCP, 2025.

[5] Shawn Shuoshuo Chen. FabricEval. https://github.com/shuoshuc/FabricEval, 2025.

[6] Shawn Shuoshuo Chen. RFold. https://github.com/shuoshuc/RFold, 2025.

[7] Andrew D. Ferguson, Steve Gribble, Chi-Yao Hong, Charles Killian, Waqar Mohsin, Henrik Muehe, Joon Ong, Leon Poutievski, Arjun Singh, Lorenzo Vicisano, Richard Alimi, Shawn Shuoshuo Chen, Mike Conley, Subhasree Mandal, Karthik Nagaraj, Kondapa Naidu Bollineni, Amr Sabaa, Shidong Zhang, Min Zhu, and Amin Vahdat. Orion: Google's Software-Defined networking control plane. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. USENIX Association, 2021.

[8] Norm Jouppi, George Kurian, Sheng Li, Peter Ma, Rahul Nagarajan, Lifeng Nai, Nishant Patil, Suvinay Subramanian, Andy Swing, Brian Towles, Clifford Young, Xiang Zhou, Zongwei Zhou, and David A Patterson. Tpu v4: An optically reconfigurable supercomputer for machine learning with hardware support for embeddings. In *Proceedings of the 50th Annual International Symposium on Computer Architecture (ISCA 23)*, Orlando, FL, USA, 2023. Association for Computing Machinery.

[9] Darius Bunandar. Passage M1000 : A 3D Photonic Interposer for AI . In *2025 IEEE Hot Chips 37 Symposium (HCS)*, pages 1–34, Los Alamitos, CA, USA, August 2025. IEEE Computer Society.