Making Network Specialization Accessible

Akshay Narayan

While for multiple decades we have taught students that computer networks provide a best-effort packet delivery abstraction, modern networks are departing from this model to provide specialized features which improve applications' expressivity and performance. For example, an ecosystem of communication libraries now mediates a modern application's communication functionality, including new communication patterns (e.g., publish-subscribe and allreduce), application networking features such as serialization (e.g., an RPC interface) and encryption, and transport functionality such as congestion control. Meanwhile, the network's canonical socket interface has remained static, except when the aforementioned libraries provide alternate ad-hoc APIs. As a result, achieving the best performance requires both implementation efforts to specialize applications to the runtime environment (e.g., hardware acceleration or custom congestion control algorithms) as well as careful out-of-band configuration of an application's runtime environment. While larger organizations can afford the resulting development and maintenance costs, smaller organizations must use generic functionality with performance overheads, inefficient resource use, and resulting higher operational costs. With my research, I argue that we should make this specialization accessible to more application developers and that changing the interfaces and abstractions applications use to access network functionality can enable this transformation.

Approach. My approach to research is goal-driven. Given my goal of making network specialization accessible, I focus on the interfaces and abstractions that need to change to accomplish it. For example, within the transport layer, I observed that the assumption that congestion control algorithms (CCAs) must act on a per-packet basis was limiting both implementation flexibility as well as use in new datapaths. I thus built CCP [2] on the two insights that delaying congestion control decisions is acceptable and that there is a small set of signals most CCAs use. Similarly, at the application layer, I observed that a lack of a narrow interface has been limiting composition and specialization at the application layer. I built this narrow interface, which I call a Chunnel, and used this abstraction in Bertha [1, 5], a new networking stack that allows runtime reconfiguration of applications' communication libraries to best fit both the network runtime environment and communication peers' capabilities.

I believe it is important to invest time in producing and maintaining research artifacts that support both reproducibility and external users. Indeed, CCP has seen real-world adoption in congestion control experimentation platforms including Facebook's QUIC networking stack and in a production CDN network, and further adoption in at least a dozen (known to me) external publications proposing CCAs built on CCP. Further, Bundler [3], built in part using CCP, was awarded "Best Artifact" at EuroSys 2021.

Bertha: A Runtime Re-configurable Network Stack

Applications today use advanced communication features and services the traditional socket abstraction does not capture. For example, modern data analytics applications might use a microservice composition in which app elements communicate using hosted publish-subscribe services or application-layer networking features (e.g., encryption, authentication, load balancing, and monitoring) implemented in a service proxy, or use hardware-offloaded application functionality. Today, using these features requires developers to commit to specific implementations and their APIs, which ties an application to a specific runtime environment. Environment-specific optimizations require engineering resources; thus, many organizations with fewer resources instead use high-level virtualized or hosted implementations that tie their application to a specific environment or sacrifice performance. Instead, I argue that just as compilers today provide architecture-specific optimizations, the application's network stack should select the communication feature implementations that are best for the network environment at runtime.

To realize this vision, I built Bertha [1, 5], a runtime re-configurable network stack. Bertha's core abstraction, called Chunnels, specifies the use of a communication feature such as sharding without committing to a specific implementation until runtime. Bertha's insight is that the narrow Chunnel interface–a data type, send and receive operations, and a simple functionality specification–suffices to express communication libraries. Application developers can compose Chunnels into stacks with branches representing implementation options. Bertha then decides at runtime which implementations of each network feature to use. This runtime re-configuration enables network-specific optimizations; e.g., an application using publish-subscribe can switch Chunnel implementations depending on the number of connection participants to safely unlock better performance. The challenge when implementing Bertha was to ensure compatibility: if the network stack can change implementations at runtime, how can it know that the implementations are data-type compatible as well as compatible with the implementations network peers choose? I solved this challenge by designing a negotiation protocol used during reconfiguration which can abstractly represent implementation options and ensure endpoints choose compatible options.

CCP: Restructuring Endpoint Congestion Control

Another aspect of network heterogeneity is new transport layer stacks, which have changed in two key ways. First, new special-purpose transport stacks have gained popularity, e.g., using QUIC, kernel-bypass, or hardware processing. Second, developers have employed new CCAs to more efficiently utilize available network bandwidth. Many such proposals use sophisticated techniques including machine learning or signal processing that are difficult, if not impossible, to implement in a tightly-constrained datapath environment where performance is crucial. These two trends combine to create a problem for both datapath developers and congestion control researchers: on one hand, incorporating new CCAs into a datapath requires knowledge and effort per CCA, and on the other hand congestion control researchers must expend per-datapath effort to implement their algorithms so applications can use them. To address this challenge, I built Congestion Control Plane (CCP) [2], a CCA implementation framework that enables (via an in-datapath DSL) flexible congestion signal measurement while simultaneously supporting asynchronous CCA implementations. CCP is designed around two key insights: first, that moving congestion control off the datapath unlocks CCA expressivity and datapath portability, and second that while this abstraction requires slight delays in congestion control processing, this delay is acceptable for modern CCAs.

Impact. CCP has had impact in both industry and academia. At least 12 publications describing new CCAs have used CCP to implement and evaluate their ideas. Within my own work, CCP enabled the Nimbus [4] congestion control project as its implementation platform, the Park [7] project as a way to demonstrate challenges in applying reinforcement learning approaches to systems domains, and the Bundler [3] middlebox project. With Bundler, we observed that traditional deployments of datapath scheduling algorithms (e.g., fair queueing or traffic class prioritization) have a drawback: the operator at the edge knows the best scheduling policy, but cannot enforce it where congestion occurs in the middle of the network. We thus designed and implemented Bundler, which uses CCP-provided congestion control mechanisms to implement an edge-queued network abstraction. For example, we showed that using Bundler to enforce fair queueing can reduce the median flow completion time of the bundled component flows. Bundler allows smaller organizations without the resources to deploy private WANs to access traffic control optimizations. Without CCP, implementing Bundler would have been challenging; the ability to easily experiment with different CCAs to shift queues to edge domains was crucial for its development.

In industry, operators have deployed CCP as an experimental framework for new CCAs in `mvfst`, Facebook's production QUIC implementation, and a large commercial CDN network. With CCP, operators can ensure isolation between new experimental algorithm implementations and the rest of the datapath, which is important for reliability in their large-scale production environment.

Future Directions

In pursuit of my goal accessible specialization in the response to modern network heterogeneity, I foresee the following three directions for future work spanning across the network stack.

Fault Localization. Modern applications use functionality across multiple domains (e.g., Internet ASes) and multiple layers (e.g., service meshes, transport, and the Internet), and can share functionality (e.g., storage services or hosted communication primitives) with other applications. Thus, when a fault happens, it can be difficult to know what went wrong. Unfortunately, current telemetry interfaces isolate information within domains and layers. For example, a November 2023 Cloudflare outage was exacerbated by a lack of an application's visibility into its datacenter's electical power system. Large organizations can invest resources in tightly controlled network environments with rich metrics; how can we unlock similar tools for smaller ones? To this end, I will design an interface that enables applications to *introspect* their network runtime environment when failures occur to query network elements for their health status. Since applications may not know ahead of time what elements their runtime environment contains, this method can both reveal failures that the operator might otherwise overlook as well as focus the operator's attention on those components that the application used and indicated anomalies.

CCA Contention. A significant impediment to the deployment of new congestion control algorithms is the multi-decade-strong conventional wisdom that their fairness properties when contending for bandwidth are important factors in flows' bandwidth allocations, and applications' resulting network performance. This is one among multiple factors that make designing special-purpose CCAs out of reach for most organizations today. In a recent position paper, I argued [6] that CCA Contention is no longer a typical determinant of flows' bandwidth allocations and proposed a measurement study to resolve this hypothesis. Confirming this hypothesis would allow CCA developers to design against a simpler network abstraction. For example, they could more easily build CCAs adapted to specific network environments, since they would not have to worry about tricky contention dynamics. Further, if I find that CCA contention does not typically determine bandwidth allocations, it will be important to understand what mechanism does, and whether that mechanism is an equitable one. Then, drawing on my experience with Bundler [3], I will design an equitable, understandable, and deployable bandwidth allocation abstraction for the Internet.

Reasoning About Performance. Modern application developers can struggle to reason about how a change to their apps' structure might affect performance. One reason is the use of monolithic communication libraries, which make it difficult to decompose performance into constituent parts. Rather, a more modular abstraction (indeed, such as Chunnels) could allow for more in-depth reasoning about the performance of applications' network stacks. For example, this reasoning could help developers reason about which functionality would most benefit from hardware offloads. Similarly, at a larger scale, how should a network architect decide which of a number of proposed systems, each with a unique set of performance characteristics and deployment costs, to use in their network? Indeed, reasoning about deploying systems across a network today offers no abstraction at all; the architect must carefully study each system and its potential interactions with all other systems. Instead, I propose to develop an abstraction that captures the resources a system uses and the benefits it provides. With this abstraction, a network architect could use a reasoning framework to keep track of their deployment and expected performance.

References

[1] **Akshay Narayan**, A. Panda, M. Alizadeh, H. Balakrishnan, A. Krishnamurthy, and S. Shenker. Bringing Reconfigurability to the Network Stack. https://arxiv.org/abs/2311.07753 (under submission).

[2] **Akshay Narayan**, F. Cangialosi, D. Raghavan, P. Goyal, S. Narayana, R. Mittal, M. Alizadeh, and H. Balakrishnan. Restructing Endpoint Congestion Control. SIGCOMM, 2018.

[3] F. Cangialosi*, **Akshay Narayan***, P. Goyal, R. Mittal, M. Alizadeh, and H. Balakrishnan. Site-to-Site Internet Traffic Control. EuroSys, 2021.

[4] P. Goyal, **Akshay Narayan**, F. Cangialosi, S. Narayana, M. Alizadeh, and H. Balakrishnan. Elasticity Detection: A Building Block for Delay-Sensitive Congestion Control. SIGCOMM, 2022.

[5] **Akshay Narayan**, A. Panda, M. Alizadeh, H. Balakrishnan, A. Krishnamurthy, and S. Shenker. Bertha: Tunneling through the Network API. HotNets, 2020.

[6] Lloyd Brown, Y. Kothari, **Akshay Narayan**, A. Krishnamurthy, A. Panda, J. Sherry, and S. Shenker. How I Learned to Stop Worrying About CCA Contention. HotNets, 2023.

[7] H. Mao, P. Negi, **Akshay Narayan**, H. Wang, J. Yang, H. Wang, R. Marcus, R. Addanki, M. K. Shirkoohi, S. He, V. Nathan, F. Cangialosi, S. Venkatakrishnan, W.-H. Weng, S. Han, T. Kraska, and M. Alizadeh. Park: An Open Platform for Learning-Augmented Computer Systems. NeurIPS, 2019.