

Networks increasingly use software in both the control and the data plane. Software Defined Networking (SDN) use software controllers running on remote servers to make control plane decisions (*e.g.*, select policy compliant paths); while Network Function Virtualization (NFV) allows software to process packets on the data plane. The software used by a network must meet the network’s performance and reliability requirements. These requirements are more stringent than those imposed on other systems – *e.g.*, network functions must process 10-100 million packets per second, and controllers must be reliable and responsive even when distributed over large geographic extents. Developing and running software to address these requirements remains challenging. My work addresses this challenge, in particular addressing questions around how such software should be built, debugged, executed and verified. I discuss this work next, and then describe my plans for future research.

1 Network Function Virtualization

Networks used to focus only on only forwarding packets (which they did using specialized ASICs), but have increasingly shifted to more general forms of packet processing (which are done using commodity processors). This paradigm shift, called Network Function Virtualization (NFV), typically involves deploying new functionality as software network functions (NFs) that run on commodity servers. To avoid being a bottleneck, these network functions must process packets at line rate without adding significant latency. As a result I/O dominates processing in these NFs, and makes them different from other commonly virtualized server applications (*e.g.*, web servers). How should we build and compose these network functions, and how can we extend network verification to account for these more complex functions.

1.1 How to Build Network Functions?

High-level programming abstractions enable rapid-development of new applications, reduce the number of bugs in these application, and improve maintainability. However, conventional wisdom holds that these abstractions come at a significant performance penalty for packet-processing applications. As a result, NFs today are largely written in C or C++ using low-level packet processing libraries such as DPDK. Some work (*e.g.*, Snabb) allows NF logic to be expressed in higher level languages (*e.g.*, Lua), but do not raise the level of abstraction. For these reasons, most NFs today are written by large teams, and can take years to develop.

To disprove conventional wisdom, we developed a high-level programming framework – NetBricks [13] – that can achieve and often exceed the performance achievable by conventional NFs. NetBricks NFs are expressed as dataflow graphs composed of a small set of framework-provided abstractions. The behavior of each abstraction can however be customized through the use of user-defined functions (UDFs). In this model, framework-provides abstractions are responsible for implementing general packet processing functionality, while UDFs are used to implement domain specific functionality for each NF. The key contribution of NetBricks is to identify a small set of abstractions that can be used to express a wide range of functionality, and that can be optimized by the framework. To evaluate NetBricks’ generality and performance we reimplemented Maglev [4], a load balancer used within Google, in approximately 100 lines of code and could achieve performance that was better than what was previously reported.

1.2 How to Compose Network Functions?

NFV deployments typically run several NFs on a single server, relying on processes, containers or virtual machines – which we collectively refer to as processes – to provide isolation. These NFs perform frequent interprocess communication (IPC) to multiplex shared network links (through a virtual switch) and compose multiple NFs. IPCs requires crossing process isolation boundaries which impose performance overhead and can result in as much as an order-of-magnitude slowdown in packet processing rates.

To mitigate this isolation overhead, we developed a zero-copy software-based isolation technique called ZCSI in NetBricks [13]. ZCSI performs most isolation checks at compile time, and therefore induces minimal runtime overheads. This technique extends work done for extensible microkernel systems such as SPIN and Singularity, but in our case is implemented in usermode and specialized for NFV applications. Furthermore, NetBricks also uses unique types (or linear types) to provide safe, zero-copy IPC. As a result NetBricks provides the same isolation as processes without the associated performance penalty.

1.3 How to prevent NF misconfiguration?

Misconfigured network functions can result in network policies being violated, *e.g.*, allowing untrusted hosts to access confidential data. Network verification tools, *e.g.*, Veriflow and Header-Space Analysis (HSA), are increasingly used to avoid misconfiguration. These tools assume that the network dataplane is stateless and implements a limited set of actions. Neither assumption holds for network functions and as a result these tools cannot be extended to verify correctness in the presence of network functions. Furthermore, since NFs are general applications, many verification tasks are in fact undecidable in their presence [12].

To address this we developed VMN [15], a verification framework for networks containing NFs. VMN focuses on verifying reachability and isolation, and operates on networks containing NF models. The use of NF models allow us to abstract away most of an NFs code, so that our concrete specification only contains an NFs forwarding model. As a result we can ensure that verifying isolation and reachability in the presence of NFs remains decidable, and avoid false-positives. Despite the use of models, VMN’s decision process does not scale to large networks, and we rely on compositional verification – where in, assuming certain conditions hold for NFs, we can verify invariants on small sub-parts of the network rather than having to consider the whole network, this allows us to verify invariants on arbitrarily large networks.

1.4 Beyond Building and Verifying Network Functions

In NSS [14], we proposed interfaces that allowed third-party developers to safely and quickly deploy NFs, safely without any manual intervention. In FTMB [19], we presented techniques that provide fault tolerance for NFs, without affecting performance. In E2 [10], we addressed challenges in cluster scheduling in NFV applications. Finally, I contributed to BESS (also know as SoftNIC) [6], a minimal, modular virtual switch designed for NFV applications.

2 Software Defined Networking

Networks are increasingly adopting the SDN paradigm, in which the network control plane – which computes forwarding state – is implemented separately from the data plane – which forwards packets based on this forwarding state. This separation allows the control plane to be logically centralized (residing in one or more controllers), making it easy to implement a wide variety of network management functionality (from routing to isolation to traffic engineering to network virtualization). This separation, however, introduces new failure modes; in particular, the control plane can fail independently of the data plane, so the control plane and data plane can have different reachability properties. Thus, one must address several classic distributed systems questions in the context of SDN control planes, including:

2.1 What policies are enforceable in the presence of control network failures?

SDNs commonly use out-of-band control networks, *i.e.*, they rely on a separate network for control plane communications. In such networks, partitions can disconnect the control plane even when the data plane remains connected. In CAP for Networks [11] we asked about the limitations on network correctness in the presence of control plane partitions, finding classes of policies where the network must either violate the policies or explicitly partition the data plane.

2.2 What is the right consistency model for distributed controllers?

SDN controllers are typically replicated across servers for fault tolerance. Currently, most replicated controllers – *e.g.*, Onix and ONOS – are designed as replicated-state machines (RSMs) and rely on consensus protocols such as Paxos to ensure the control plane is serializable. The use of consensus protocols reduces control plane resilience – a majority of controllers need to be able to communicate for the control plane to make progress – and increases latency when responding to network events. However, serializability – or stronger consistency guarantees – are considered by the community as essential for achieving correctness in SDNs, mandating the use of consensus protocols or similar mechanisms.

In SCL [16] we challenge this notion, and show – through theoretical proofs and empirical results – that serializability is not required for correctness, and that weaker consistency requirements allow SCL to provide lower latency and higher availability.

2.3 How to debug distributed controllers?

SDN control applications are distributed applications, which are hard to debug due to concurrency. Current debugging techniques require developers to read through and reconcile logs from different components of the system. Along with Colin Scott, I was involved in the development of STS [17], a tool for finding minimal causal sequences (MCS); that is, the smallest set of events that could trigger the bug in question. In DEMi [18], we extended dynamic partial-order reduction (DPOR), an algorithm used for model checking distributed systems, to find such MCSs for both faulty SDN controllers and more general distributed systems (such as Apache Spark and Raft implementations).

3 Other Research

Beyond SDN and NFV, my other research in networking has looked at inter-domain and intra-domain routing. My work in this area includes developing Data Driven Connectivity [7], a practical algorithm that achieves perfect resilience – *i.e.*, ensures that routes are available as long as the network is connected – and requires only 2 bits in the packet header. DDC can be implemented in modern programmable dataplanes such as P4. In follow-up work [3], we have also investigated theoretical bounds on the resilience that can be achieved without changing forwarding tables. In another work, Route Bazaar [2], we developed an alternate inter-domain routing scheme that uses reputation mechanisms and tamper-proof logs (built upon mechanisms derived from cryptocurrency blockchains) to provide greater flexibility when routing traffic on the internet.

Beyond networking, my research has also looked at problems in large scale data analytics, where I have collaborated on the following projects: BlinkDB [1], an approximate query processing engine that can provide approximate results with bounded latency; Cayman [20], a scheduler that provides explicit support for cluster compute jobs that use sampling (*e.g.*, BlinkDB queries, machine learning jobs, etc.); Drizzle [21], a low latency, fault tolerant framework for streaming applications; and Tiny Tasks [8], a proposal for using very small (millisecond length) tasks in cluster computing frameworks to improve cluster utilization. I was also involved in building and testing Ivy [9], a system for interactively discovering inductive invariants and verifying distributed systems.

4 Future Research

My approach to research has revolved around understanding techniques in areas such as programming languages (NetBricks, VMN, DEMi), operating systems (NetBricks), distributed systems (SCL, DDC, Ivy), cryptography (RouteBazaar) and other areas, and extending these techniques to solve problems in networking and systems. The problems I am interested in addressing include:

- **How to verify and debug applications built with microservices?** Many complex applications are now built by composing “microservices” which communicate with each other through a virtual network. An individual application can be composed of many 100s of microservices (*e.g.*, Uber claims to use a 1000 microservices for its application). Despite their growing popularity and complexity, existing tools do not support verifying application-wide invariants nor do they support debugging issues resulting from the interaction between microservices. General tools for doing this must target the virtual network, which is the only component with visibility across microservices, however it lacks semantic information. Can we use message information from the virtual network, in conjunction with simplified models of individual microservices, to verify application-wide invariants and provide better debugging support?

- **How to scale programs in non-uniform memory access architectures?** Increasingly, the latency for accessing a particular memory address varies by core; this is referred to as non-uniform memory access (NUMA). The number of NUMA domains has been steadily increasing in commodity servers. The current best practice for scaling across these domains relies on share-nothing parallelism or requires treating each domain as a separate machine. Both choices require developers to build more complex synchronization primitives. Can we instead provide programming models and data structures that allow applications with shared memory to more easily scale across NUMA nodes, while still achieving high performance?

- **Can we allow centralized computation of inter-domain paths without sacrificing AS confidentiality?** In prior work [5] we have shown that centralized route computation can both improve convergence time

and reduce path instability in the Internet. However, the current distributed approach allows ASes to ensure that their policies remain secret. Our previous attempt to allow centralization while preserving anonymity built on the use of secure multi-party computation (SMPC), which had high performance overheads. Can recent hardware mechanisms for confidentiality, *e.g.*, Intel SGX and ARM TrustZone, be used to build systems that allow inter-domain path computation to be centralized, while preserving confidentiality for ASes, and not requiring them to trust any provider?

References

- [1] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica. BlinkDB: Queries with Bounded Errors and Bounded Response Times on Very Large Data. In *EuroSys*, 2013.
- [2] I. Castro, A. Panda, B. Raghavan, S. Shenker, and S. Gorinsky. Route Bazaar: Automatic Intedomain Contract Negotiation. In *HotOS*, 2015.
- [3] M. Chiesa, I. Nikolaevskiy, S. Mitrovic, A. Panda, A. Gurtov, A. Madry, M. Schapira, and S. Shenker. The Quest for Resilient (Static) Forwarding Tables. In *INFOCOM*, 2016.
- [4] D. E. Eisenbud, C. Yi, C. Contavalli, C. Smith, R. Kononov, E. Mann-Hielscher, A. Cilingiroglu, B. Cheyney, W. Shang, and J. D. Hosein. Maglev: A Fast and Reliable Software Network Load Balancer. In *NSDI*, 2016.
- [5] D. Gupta, A. Segal, A. Panda, G. Segev, M. Schapira, J. Feigenbaum, J. Rexford, and S. Shenker. A New Approach to Interdomain Routing Based on Secure Multi-Party Computation. In *HotNets*, 2012.
- [6] S. Han, K. Jang, A. Panda, S. Palkar, D. Han, and S. Ratnasamy. Softnic: A software nic to augment hardware. In *Technical Report UCB/EECS-2015-155*. EECS Department, University of California, Berkeley, 2015.
- [7] J. Liu, A. Panda, A. Singla, B. Godfrey, M. Schapira, and S. Shenker. Ensuring Connectivity via Data Plane Mechanisms. In *NSDI*, 2013.
- [8] K. Ousterhout, A. Panda, J. Rosen, S. Venkataraman, R. Xin, S. Ratnasamy, S. Shenker, and I. Stoica. The Case for Tiny Tasks in Compute Clusters. In *HotOS*, 2013.
- [9] O. Padon, K. McMillan, A. Panda, M. Sagiv, and S. Shoham. Ivy: Interactive Verification of Parametrized Systems via Effectively Propositional Reasoning. In *PLDI*, 2016.
- [10] S. Palkar, C. Lan, S. Han, A. Panda, K. Jang, S. Ratnasamy, L. Rizzo, and S. Shenker. E2: A Framework for Network Function Virtualization. In *SOSP*, 2015.
- [11] A. Panda, C. Scott, A. Ghodsi, T. Koponen, and S. Shenker. CAP for Networks. In *HotSDN*, 2013.
- [12] A. Panda, K. Argyraki, M. Sagiv, M. Schapira, and S. Shenker. New Directions for Network Verification. In *SNAPL*, 2015.
- [13] A. Panda, S. Han, K. Jang, M. Walls, S. Ratnasamy, and S. Shenker. NetBricks: Taking the V out of NFV. In *OSDI*, 2016.
- [14] A. Panda, J. M. McCauley, A. Tootoonchian, J. Sherry, T. Koponen, S. Ratnasamy, and S. Shenker. Open Network Interfaces for Carrier Networks. *SIGCOMM Computer Communication Review*, 46(1):5–11, 2016.
- [15] A. Panda, O. Lahav, K. Argyraki, M. Sagiv, and S. Shenker. Verifying Reachability in Networks with Mutable Datapaths. In *NSDI*, 2017.

-
- [16] A. Panda, W. Zheng, X. Hu, A. Krishnamurthy, and S. Shenker. SCL: Simplifying Distributed SDN Control Planes. In *NSDI*, 2017.
 - [17] C. Scott, A. Wundsam, B. Raghavan, A. Panda, A. Or, J. Lai, E. Huang, Z. Liu, A. El-Hassany, S. Whitlock, H. Acharya, K. Zarifis, and S. Shenker. Troubleshooting Blackbox SDN Control Software with Minimal Causal Sequences. In *SIGCOMM*, 2014.
 - [18] C. Scott, A. Panda, V. Brajkovic, G. Necula, A. Krishnamurthy, and S. Shenker. Minimizing Faulty Executions of Distributed Systems. In *NSDI*, 2016.
 - [19] J. Sherry, P. X. Gao, S. Basu, A. Panda, A. Krishnamurthy, C. Maciocco, M. Manesh, J. Martins, S. Ratnasamy, L. Rizzo, and S. Shenker. Rollback Recovery for Middleboxes. In *SIGCOMM*, 2015.
 - [20] S. Venkatraman, A. Panda, G. Ananthanarayanan, M. Franklin, and I. Stoica. The Power of Choice in Data-Aware Cluster Scheduling. In *OSDI*, 2014.
 - [21] S. Venkatraman, A. Panda, K. Ousterhout, A. Ghodsi, M. J. Franklin, B. Recht, and I. Stoica. Drizzle: Fast and adaptable stream processing at scale, 2016.