Commoditizing the "S" in SDN: BareMetal Switching Infrastructure

Theophilus Benson

Today's SDN environment provides insufficient tools and primitives for enabling the inclusion of third party software in a safe and reliable manner. Particularly, controller are architected in a monolithic fashion with insufficient fault containment domains, SDNs provide no test harnesses for detecting bugs and certifying proper application functionality, and SDNs provide no primitives for including third-party extensions to the data-plane; e.g. network functions. The implications of these limitations are that: (1) SDN-Apps are poorly tested and prone to bugs which will remain until SDN-App developers are able to test their code under representative conditions; (2) SDN-App crashes can take down the entire network due to the monolithic design of the controllers, and existing replication based techniques will not help; and (3) operators are unable to efficiently integrate new data-plane functionality a flexible manner, thus limiting the set of policies that can be implemented in the network. These deficiencies span both the control and data plane thus greatly impacting the viability of SDNs as a viable alternative for many enterprises.

Our solution to these problems consists of three research direction: (1) a re-design of the controller architecture [2] centering around a set of abstractions that enables us to safely run SDN-Apps with a best-effort model that overcomes SDN-App failures by detecting and transforming failure triggering events; (2) a re-design of the data-plane network functions [1] into more modular components with a novel programming language that provides native support for tighter coupling with the SDN controller, e.g with a control channel; and, (3) the development of a novel control plane substrate [3] that safely enables large scale testing on a production network

New Directions Much of my existing work, focuses on redesigning the SDN operating system and developing frameworks for the SDN operating system to support third party SDN-Apps. More recently, we have started to explore the introduction of novel switch primitives that enable delegation, or out-sourcing of functionality, without delegating control over the networking or compromising visibility. For example, we are introducing primitive to enable in-network transactional semantics thus distributing and reducing the efforts required to rollback network changes [2]. In addition, we are exploring switch primitives that enable recovery from controller failure by allowing the switches to pro-actively retransmits events to the new master controller. Orthogonally, we are exploring switch primitives that enables the switch to cache packets – given this cache, a switch can respond to ARP requests and LLDP requests without needing to burden the controller. This improves controller scalability and minimizes overheads – most importantly, it enables the controller to support a large range of applications and switches.

The growing heterogeneity of feature-set for hardware devices and supported SDN-apps introduces compatibility, inter-interoperability, and further complexities that create bugs. While existing work [2] enables efficient recovery from bugs and allows for proactive detection of bugs [3], the growing complexity coupled with the SDNApp store will create a need methodologies to certify SDN-Apps for specific types and class of networks.

TestBed Requirements Existing research infrastructures, e.g. NSFCloud and GENI, focus on providing researcher access to bare-metal hardware and to virtualized resources. These enable testing SDN operating systems and novel switch primitives using emulators, e.g. Mininet, and by using virtual switches. However, these virtual resources often obfuscate performance challenges and abstract practical challenges that arise from using physical switches. These issues limit the viability and ability to test primitives for switches. While NetFPGAs allow prototyping of hardware primitives – there is a huge and growing need to develop abstractions that run on existing whitebox or commodity switches. Further, there is a need to understand how novel SDN operating systems interact when exposed to the heterogeneity of actual physical switches – and more importantly to understand how the ability to modify the software switches can be used to improve the controller architecture. In this proposal, we argue for a research infrastructure to provides "baremetal switches" and enables testing of novel switch-OS, testing of novel SDN agents on established switch agents, and comparisons of hardware choices on the Switch-OS and/or switch agents. This research infrastructure itself introduces several challenges specific to virtualization and sharing of a switch's CPU and memory. Even more, there is a need for appropriate abstractions for managing and accounting for switch resource utilization in a similar manner as modern hypervisors. The challenges are distinct from those addressed by existing network hypervisors, e.g. FlowVisor and ONOS, which simply enable multiplexing of the TCAM by multiple entries. What we require involves more holistic virtualization of a switch and its resources: this further requires switch support.

References

[1] ANWER, B., BENSON, T., FEAMSTER, N., AND LEVIN, D. Programming slick network functions. SOSR '15.
[2] CHANDRASEKARAN, B., AND BENSON, T. Tolerating sdn application failures with legosdn. HotNets '14.
[3] SHELLY, N., TSCHAEN, B., FÖRSTER, K.-T., CHANG, M., BENSON, T., AND VANBEVER, L. Destroying networks for fun (and profit). HotNets '15.