A Northbound API for Sharing SDNs

Andrew Ferguson*, Arjun Guha†, Chen Liang*, Rodrigo Fonseca*, Shriram Krishnamurthi*
*Brown University, † Cornell University

Managing and reasoning about today's enterprise and datacenter networks is a herculean task. These networks are made of many kinds of switches and middleboxes that are configured in isolation using proprietary configuration languages. Software-defined networking has the potential to alleviate these problems: protocols such as OpenFlow define a standard interface to network devices that controllers can use for uniform configuration. This enables SDN controllers to provide high-level abstractions for network control programs, akin to the abstractions early operating systems introduced for heterogeneous hardware resources.

If SDNs introduce a network operating system, then current stacks lack the equivalent of a "user-level" API to allow multiple, less-privileged principals to affect network operations. In a network, such principals may be end-users or, more commonly, end-hosts and applications acting on users' behalf. Despite potential advantages, such untrusted principals cannot affect network operations in today's networks. For example, if a video-conferencing application could request a bandwidth reservation, it could guarantee a high-quality call within an enterprise network. If a host which detects malicious traffic could notify the network control-plane, the network could block malicious traffic closer to its source. Or, a MapReduce-style application running on several end-hosts could improve performance by requesting maximally disjoint paths during its shuffle phase.

For shared configuration to be practical, a network must limit the privileges of end users, and provide a simple API for requesting high-level, abstract resources. Furthermore, it is infeasible to give untrusted principals direct control of a software-defined network. Without a carefully-crafted security policy, a malicious or misbehaving principal could hog bandwidth, drop others' packets, or worse. Therefore, for several principals to safely participate in network configuration, we need powerful and flexible conflict-resolution mechanisms. At the same time, an application should be able to request a bandwidth reservation, without worrying about queue configuration on switches; a distributed framework should be able to request disjoint paths, without precisely calculating these paths itself; an end-host firewall should be able to block malicious traffic, without determining the switch where such traffic enters the network. We call a network that safely provides these high-level abstractions for several principals a *participatory network* [2].

In this poster, we present the design and evaluation of PANE, our prototype participatory networking controller. PANE is built as a controller for an OpenFlow-based SDN. We use it to control a heterogenous network of software and hardware switches supporting Linux servers and mobile devices. PANE's design addresses two key challenges of any Northbound API: how to safely decompose control and visibility of the network, and how to resolve conflicts between participants and across requests. Our solutions to these challenges were developed through formal reasoning, and by porting real-world applications to solve existing use cases.

Our experiments demonstrate that PANE can improve application performance, and our experience suggests that modifying applications to use its API is easy. With only minor modifications, we enhanced four third-party applications to exploit PANE: Ekiga, SSHGuard, ZooKeeper, and Hadoop. The PANE prototype and these modified applications are all publicly available on Github.[1]

Network Abstractions PANE provides a simple text-based protocol that principals can use to *request* network resources, *query* the state of the network, or provide *hints* about future traffic patterns (Table 1). Requests are for resources (*e.g.*, bandwidth or access control), and imply an action to be taken by the controller. Queries read properties of the network's state (*e.g.*, switch capabilities, or available bandwidth). Hints inform PANE about current or future traffic characteristics; the controller may choose to use hints to improve service. Requests and hints can optionally include start and end times, allowing the controller to plan for the future.

Authorization and Conflict Resolution PANE delegates privileges to issue requests, hints, and queries in a hierarchical fashion from the network's administrator to its principals. In PANE, *shares* determine *who* can say *what* about *which* flows. Shares form a hierarchy; principals can create new shares under a share they own, as well as add other principals. Child shares cannot have more privileges, or refer to more flow space, than their parent share.

PANE principals issue requests, hints, and queries in the context of a share. PANE first evaluates whether the requested resources fit within the share, taking into account static rules and previously committed resources. If the request is feasible, PANE checks for any conflicts with previously granted requests. This evaluation happens hierarchically as well, with flexible conflict resolution operators along the tree of resources [1].

[1] `http://github.com/brownsys`

Share	S	$\in \{P\} \times \{F\} \times \{Priv\}$	A share gives principals some privileges to affect a set of flows.
Principal	P	$::= (user, host, app)$	A triple consisting of an *application*, running on a *host* by a *user*.
Flow	F	$::= \langle$**srcIP**$=n_1$, **dstIP**$=n_2$, **proto**$=n_3$, **srcPort**$=n_4$, **dstPort**$=n_5\rangle$	A set of packets with shared properties: source and destination IP address, transport protocol, and source and destination transport ports.
Privilege	*Priv*	::= **CanDeny** n | **CanAllow** n	The privileges to allow or deny traffic for up to n seconds (optional).
		| **CanReserve** n | **CanRateLimit** n	The privileges to reserve bandwidth or set rate-limits, up to n MB.
		| **CanWaypoint** $\{IP\}$ | **CanAvoid** $\{IP\}$	The privileges to direct traffic through or around particular IP addresses.
Message	*Msg*	::= $P : \{F\} : S \rightarrow$ (*Req Tspec* | *Hint Tspec* | *Query*)	A message from a principal with a request, hint, or query using a share.
Time Spec	*Tspec*	::= **from** t_1 **until** t_2	An optional specification from time t_1 until t_2.
Request	*Req*	::= **Allow** | **Deny**	Request to allow/deny traffic.
		| **Reserve** n | **RateLimit** n	Request to reserve n MB or rate-limit to n MB.
		| **Waypoint** *IP* | **Avoid** *IP*	Waypoint/avoid traffic through a middlebox with the given IP address.
Query	*Query*	::= **TrafficBetween** *srcIP dstIP* | ...	Query the total traffic between two hosts.
Hint	*Hint*	::= **Duration** t | ...	Hint that the flow's duration is t.

Table 1: PANE definitions (top) and end-user API (bottom). There is also a simple API to create shares and delegate privileges.

Implementation and Evaluation PANE is implemented as a Haskell program. It primarily uses OpenFlow 1.0 to configure switches; it also uses Open vSwitch commands, and OpenFlow slicing extensions to configure queues.

Ekiga Ekiga is an open source video conferencing application which we modified to ask the user for the anticipated duration of video calls. Ekiga uses this information to schedule a bandwidth reservation between the caller's host and either the network gateway or the recipient's host. If a desired reservation cannot be scheduled, the user is notified that the call quality may be sub-optimal.

SSHGuard SSHGuard detects brute-force attacks by monitoring logs and installing local firewall rules (*e.g.*, via `iptables`). We modified SSHGuard to send **Deny** messages to the PANE controller. Because PANE controls the local network, it can block malicious traffic at the edge. Not only does this offload work from the end-host's network stack, it also protects internal network traffic that may have suffered by sharing a link along which a denial-of-service attack is taking place.

ZooKeeper ZooKeeper provides consistent, available, shared state among a quorum of replicated servers. For resiliency in the face of network failures, ZooKeeper servers may be distributed throughout a datacenter; therefore, quorum messages may be delayed by heavy traffic on shared links. Because ZooKeeper's role is to provide coordination for other services, such negative effects are undesirable. We modified ZooKeeper to make bandwidth reservations using PANE. In our benchmark, we found that competing traffic dramatically reduced ZooKeeper's performance: average latency quadrupled from 1.55ms to 6.46ms. When, upon startup, each member of the ensemble made a PANE reservation for 10 Mbps of guaranteed minimum bandwidth for messages with other ZooKeeper servers, average latency dropped down to 2.02ms.

Hadoop We augmented a Hadoop 2.0.3 pre-release to use PANE. By using PANE, our version of Hadoop is able to reserve guaranteed bandwidth for its operations. The first set of reservations occurs during the shuffle: each reducer reserves bandwidth for transferring data from the mappers. The second set reserves bandwidth when writing the final output back to HDFS. These few reservations protect the majority of network transfers which occur during the lifetime of a Hadoop job.

We executed three 40 GB sort jobs in parallel on a network of 22 machines (20 slaves and two masters) connected by a Pronto 3290 switch controlled by PANE. Hadoop currently has the ability to prioritize or weight jobs using the scheduler, but this control does not extend to the network. In our benchmark, the first two jobs were provided with 25% of the cluster's memory resources, and the third, acting as the "high priority" job, was provided with 50%. The benchmark was run in two configurations: in the first, Hadoop made no requests using PANE; in the second, our modified Hadoop requested guaranteed bandwidth for each large flow.

Averaged across three runs, the high priority job's completion time decreased by 19% when its bandwidth was guaranteed. Because it completed more quickly, the lower priority jobs' runtime also decreased, by an average of 9%, since Hadoop's work-conserving scheduler re-allocates freed memory resources to remaining jobs.

References

[1] Andrew D. Ferguson, Arjun Guha, Chen Liang, Rodrigo Fonseca, and Shriram Krishnamurthi. Hierarchical Policies for Software Defined Networks. In *HotSDN '12*.

[2] Andrew D. Ferguson, Arjun Guha, Jordan Place, Rodrigo Fonseca, and Shriram Krishnamurthi. Participatory Networking. In *Hot-ICE '12*.