

# Pushing Enterprise Security Down the Network Stack

Ankur Nayak, Alex Reimers, Russ Clark, Nick Feamster  
School of Computer Science, Georgia Tech

## ABSTRACT

Network security is typically reactive: Networks provide connectivity and subsequently alter this connectivity according to various security policies, as implemented in middleboxes, or at higher layers. This approach gives rise to complicated interactions between protocols and systems that can cause incorrect behavior and slow response to attacks. In this paper, we propose a proactive approach to securing networks, whereby security-related actions (*e.g.*, dropping or redirecting traffic) are embedded into the network fabric itself, leaving only a fixed set of actions to higher layers. We explore this approach in the context of network access control. Our design uses programmable switches to manipulate traffic at lower layers; these switches interact with policy and monitoring at higher layers. We apply our approach to Georgia Tech’s network access control system, show how the new design can both overcome the current shortcomings and provide new security functions, describe our proposed deployment, and discuss open research questions.

## 1. Introduction

Enterprise networks host many potentially untrusted devices. These devices are dynamic, increasingly heterogeneous, and vulnerable to compromise. The increasing complexity of communications networks posed by the growing number of devices, applications, middleboxes, and traffic types—coupled with users’ demands for secure, highly available networks—has made operating and maintaining networks extremely challenging.

Today’s network layer enables connectivity but does not provide any mechanisms for directly implementing security policies. As such, security is typically an afterthought, embodied by middleboxes, authentications, alert systems, and intrusion detection systems. This *post hoc* approach to security creates a plethora of independently (and often hard-to-manage) devices that may interact in unexpected ways resulting in weaker security or incorrect operation (*e.g.*, misconfiguration) [4]. For example, network admission control on the Georgia Tech campus network involves interaction between firewalls, dynamic address (DHCP) servers, virtual LANs, and intrusion detection systems, not to mention the switches and routers themselves [13]. As we will see, the interaction between these many “moving parts” creates a system that is brittle and unresponsive in the face of various security threats. This paper explores the following question: *Instead of deploying a collection of point solutions in the network, could security policies be integrated into the network fabric itself?*

Extending the metaphor of a network operating system [8] to the design of *secure* networks, we draw inspiration from the design of secure software and operating systems, whereby systems are built from, small, hardened, trusted components. Our design aims to imbue the network layer itself with the basic functions needed to implement security policies, as well as a control interface that allows the appropriate level of control to higher layers that wish to express those policies or explicitly control traffic. For example, consider the case of quarantining an infected host: This task currently requires network administrators to (1) install on-path firewalls that perform on-path inspection of traffic; and (2) manually update firewall rules when intrusion detection systems raise alarms. Instead, we advocate decoupling the tasks of monitoring and control, performing distributed inference and monitoring and detection using existing subsystems that can monitor traffic at higher layers and detect compromised hosts (*e.g.*, [7, 11]) and then push control resulting from this inference *into the network fabric itself* by altering how the network switches themselves forward traffic.

Several recent trends make this refactoring possible: First, programmable (and software-based) network devices [3, 10] allow *more direct, fine-grained control* over traffic in the network. At first blush, programmable network devices might seem to present yet another source of complexity, yet we believe that this programmability actually presents an opportunity to proactively secure the network layer. Second, distributed network monitoring algorithms have improved significantly in their ability to quickly and accurately correlate traffic from many distinct (and often distributed) sources to detect coordinated attacks (*e.g.*, for detecting botnets [7] and spammers [11]). Finally, recent network designs that centralize some aspects of network control [5, 6] allow us to make some aspects of security inherent to the network itself.

In this paper, we study this approach in the context of a specific—and initially modest—network security task: Network admission control and monitoring. Rather than applying security functions at higher layers, we control connectivity at the lowest possible layer, using policies installed in programmable switches by a controller. The controller interacts directly with the switches, thus simplifying complicated distributed configuration tasks and possibly even eliminating “security middleboxes” altogether. The switches interact with monitoring subsystems through a controller that allows an operator to quarantine hosts or subsets of traffic whenever a compromise is detected. To the best of our knowledge, none of the campus networks today allow dynamic configuration of network policies based on integration with monitoring systems. We expect that, by decoupling security policies

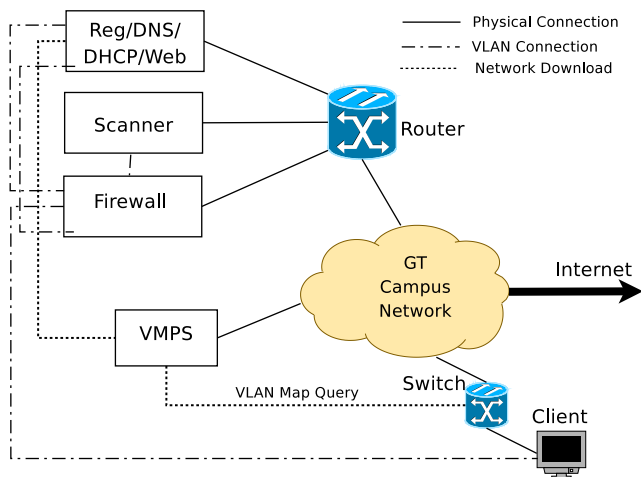


Figure 1: Current START Architecture.

from the implementation of these policies in the switches themselves, our design will not only simplify the implementation of network security policies, but also allow for more fine-grained security policies and a wider range of features.

Our proposed design presents many research challenges. First, the system must support a large number of users and traffic flows: The residential network alone on the Georgia Tech network has 16,000 active users. The system must provide flexibility and dynamic control, without storing a prohibitive amount of state at the switches themselves or introducing excessive delays on packet forwarding. Second, the controller and programmable switches must be integrated with real-time monitoring and alert systems; the controller must be able to quickly correlate and synthesize alerts and quickly send control messages to switches to affect traffic flows. Finally, the controller channel must be secure: the controller and switch interfaces must be robust to attack, and the control channel between the controller and the devices themselves must be available and secure.

The rest of the paper is organized as follows. Section 2 presents background on OpenFlow and an overview of the current authentication infrastructure on the Georgia Tech campus network. Section 3 describes the proposed design, and Section 4 describes our deployment plan. Section 5 presents related work, and Section 6 concludes with a summary and research agenda.

## 2. Background

We describe the network admission control problem and introduce OpenFlow [10], an interface for programmable switches on which we base our design.

### 2.1 Network Access Control and Monitoring

Campus and enterprise networks are often large, heterogeneous, and unmanageable. Thus, network administration can be both troublesome, manual, and error-prone. Network administrators encounter often situations where machines are infected or compromised. Today, the network operator must manually remove or quarantine the machine from the net-

work, which is tedious. A network admission control system should be able to offer flexible, fast control over network traffic while also scaling to a large number of users and traffic flows. To the extent possible, network management should also be automated, to ease the burden on the network administrators.

#### 2.1.1 Current system overview

Figure 1 shows the current START architecture [13], the authentication system deployed in the Georgia Tech Campus. It is currently based on virtual LANs (VLANs) and VLAN Management Policy Server (VMPS) [15] and provides for dynamic network assignment that allows users to be placed on a separate network for authentication, scanning, and access to software update services to correct any problems discovered during the scan. After a client is authenticated and passes these tests, the system migrates a client to the regular VLAN with full network access and gives the client a public IP address. The START system supports the following functions:

**Registration** The registration system provides the Web interface to the backend registration database, DHCP, DNS, authentication and updates for external systems. The Web interface guides users through the registration process. The DNS server for the network is a custom application written in perl. It returns the IP address for the registration server for all DNS queries, except for a list of domains needed for updating workstations (*e.g.*, windowsupdate.com). Two instances of the DHCP server are run: One for the unregistered VLAN, and one for the registered VLAN. Each instance has its own configuration files that are created automatically from data in the registration system’s database.

**Initial Scanning** During the registration process, systems are scanned for known vulnerabilities. If the scan reveals vulnerabilities, the user is presented with these vulnerabilities and given an opportunity to update the system. The firewall for the network allows traffic to get to the appropriate update servers for Microsoft and Apple.

**Firewall** The registration VLAN uses a firewall to block network traffic to unregistered desktops. However, the firewall allows Web and secure Web (*i.e.*, port 80 and 443) traffic to pass so that desktop machines can reach update sites.

Various routers and switches are employed to facilitate creating the VLANs necessary for the needed networks. The local switches determine which VLAN for each machine that joins the network. The switch will download VLAN maps periodically from a VMPS. Unknown MAC addresses are assigned to the unregistered VLAN and known MAC addresses are placed onto the appropriate subnet. VMPS periodically downloads the VLAN maps from the registration server. Network security is enforced by creating ARP tables that map each MAC address to its registered IP and pushing that table to each router.

#### 2.1.2 Problems with the current design

The current architecture has several shortcomings:

1. **Access control is too coarse-grained.** START deploys two different VLANs to separate infected/compromised machines from healthy machines. This segregation results in all compromised hosts residing on a single VLAN; such a configuration does not provide proper isolation, since these infected hosts are not isolated from each other. Additionally, relying on VLANs makes the system inflexible and less configurable, because VLANs typically map hosts to network segments according to MAC address, *not* according to individual flows.
2. **Hosts cannot be dynamically remapped to different portions of the network.** In the current configuration, when a machine is mapped to a different part of the network, it must be rebooted to ensure that it receives a public IP address, which is inconvenient because it relies on user intervention.
3. **Monitoring is not continuous.** Authentication and scanning only occur when a network device is initially introduced; if the device is subsequently compromised (or otherwise becomes the source of unwanted traffic), it cannot be dynamically remapped to the garden-walled portion of the network.

Many of the current shortcomings result from the fact that security functions have been added on top of the existing network infrastructure. This design was natural when switches needed to be treated as “black boxes”; however, switch vendors have begun to expose a standard interface, OpenFlow, whereby an external controller can affect the way a switch forwards traffic. This additional facility at the switches themselves offers the opportunity for placing more security functions at lower layers of the network stack. We briefly summarize the OpenFlow architecture below.

## 2.2 OpenFlow

OpenFlow-enabled switches expose an open protocol for programming the flow table and taking actions based on entries in these flow tables. The basic architecture consists of a *switch*, a centralized *controller*, and end-hosts. Typically, a single controller communicates with multiple switches. The switch and the controller communicate over a secure channel using the standardized OpenFlow control protocol, which can affect flow table entries on the switch. Currently, all OpenFlow switches support the following policies:

1. **Forward** this flow’s packets to a given port or ports. This function allows packets to be forwarded.
2. **Encapsulate** and forward this flow’s packets to a controller. In this case, the packet is delivered to a secure channel, where it is encapsulated and sent to a controller. This function may typically be used for the first packet in a new flow, so a controller can decide if the flow should be added to the flow table.
3. **Drop** this flow’s packets. This operation can be used to restrict access, to curb denial of service attacks, or to reduce spurious broadcast discovery traffic from hosts.

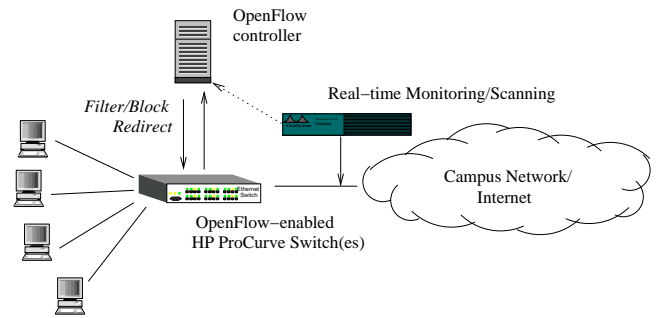


Figure 2: OpenFlow-based START Architecture.

In most switches, these functions should operate at line rate. Upon receiving the first packet from an unknown sender, the switch sends the packet back to the controller. The controller then decides how to forward the packet and communicates this information back to the switch. Thus, the OpenFlow interface provides dynamic remote control of the switches.

## 3. Design

In this section, we push security functions into network switches to improve network access control and monitoring. We describe our approach to integrating network-layer access control into switches themselves, including a revision to the current architecture based on OpenFlow-enabled switches and a centralized controller. We then explain how this new network layer can expose functions to higher layers that can implement more complex policies. We describe how this design enables fine-grained, dynamic control over traffic for implementing security policies, as well as how it simplifies network design by eliminating on-path middleboxes that implement security policies at higher layers.

### 3.1 Network-Layer Access Control

Figure 2 shows an OpenFlow-enabled network that can support security functions such as access control at the switches themselves. When a device first connects, it broadcasts a DHCP “discover” message. The DHCP server sends back a public IP address to the machine. To gain access to the wide-area Internet, the machine must authenticate itself via the START Web service; OpenFlow-enabled switches can redirect all HTTP requests from unauthenticated machines to the START Web site by default. Once a user authenticates the machine, the Web service saves the MAC address of the machine and updates flow table entries to allow access to a restricted set of destinations (*e.g.*, Microsoft Update). At this point, a scanner examines the device for potential infections. If the machine passes the scan, the START Web service sends a request to controller to permit traffic from this machine to be forwarded to any destination. The network performs continual scanning of hosts, using distributed inference techniques (some of which we describe in more detail in Section 3.2). If the machine is later found to be infected, then it is quarantined.

The controller maintains a state machine for every MAC address of machines connected to the network. Here, we

will describe what policies are required for each state. Every machine can reside in one of three states. Either it can be in registration state, or monitoring state, or quarantined state. In the registration state, the machine is in the process of authenticating itself to the web service. The monitoring state enables the monitoring softwares to pro-actively detect infected machines in the network and send updates to the controller. We plan to deploy BotMiner for Bot monitoring and SNARE/SpamTracker for spam. In the third state, which is the quarantined state, the machine is firewalled and denied access to the Internet.

**Registration state** When the switch receives a packet from a machine for which it has no flow table entry, it forwards the packet to the controller. The controller maintains a database of authenticated machines, as well as the flow-table entries associated with them. The following flow policies are then sent to switch:

1. Drop all packets other than HTTP, DHCP and ARP.
2. Broadcast all DHCP and ARP packets.
3. Allow HTTP traffic to update sites.
4. Forward other HTTP requests to the Web service.

**Operation State** In addition to the normal forwarding policies, the controller also receives updates from network monitors about the IP addresses of infected (or otherwise misbehaving) hosts. The following policies are implemented in this state:

1. Forward all packets using normal routing.
2. If the controller receives an update about an infected host, it moves that host to the quarantined or registration state and updates switch flow tables accordingly.

**Quarantine State** In the quarantined state, the host is essentially disconnected from the network:

1. Drop all packets coming from the machine based on the MAC address.
2. Upon HTTP request, return Web page to client informing the user of the infection.

The controller manages the state of each machine and updates the flow table entries in the switches accordingly.

### 3.2 Exposing Control to Higher Layers

The architecture implements security functions at lower layers on the switches themselves, exposing only a standard control interface to higher layers where more sophisticated policies can be expressed. Instead of manipulating traffic directly, higher layers and other components such as alert systems only control traffic via the standard, narrow OpenFlow-based switch interface.

This refactoring keeps on-path forwarding decisions simple, while still allowing complex policies to be implemented through a standard control interface. One of the areas where this separation of control facilitates more complex policies

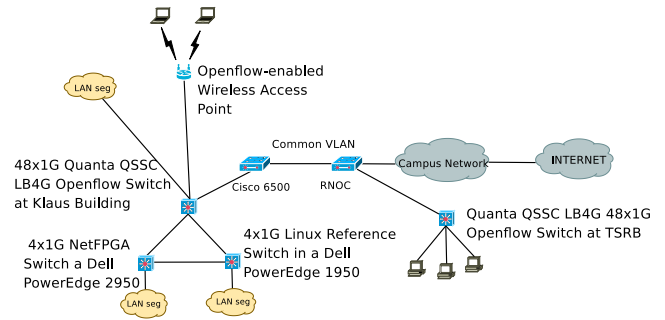


Figure 3: Research Testbed.

is *continuous monitoring*, and, in particular, distributed inference (particularly using information gleaned from protocols at higher layers). Whereas today’s networks completely decouple monitoring from lower-layer traffic control, the proposed approach allows switches to dynamically re-map clients based on input from higher layers (*e.g.*, alarms from distributed network monitoring systems, such as BotMiner [7] and SpamTracker [11]).

## 4. Planned Deployment and Evaluation

In this section, we describe the initial research testbed on which we plan to deploy a prototype implementation of the OpenFlow-based network access control system.

**Research Testbed** Figure 3 illustrates the initial deployment platform that we are already building to test the OpenFlow-based START architecture. The deployment is a dedicated network that is physically separate from the production network and yet has its own IP prefix and upstream connectivity. It consists of five OpenFlow-enabled switches: two Quanta QSSC LB4G 48x1GB switches connected on a common VLAN (one in the Klaus Computing building, and one in the Georgia Tech Technology Square Research Building), each with a 10GB copper uplink connecting machines from two research labs, as well as experimental network devices; one HP ProCurve 48x1GB switch deployed in the Georgia Tech Research NOC (RNOC) [12]; one OpenFlow-enabled 4x1GB NetFPGA switch, running in a Dell PowerEdge 2950; and one Linux reference implementation of OpenFlow, with a 4-port NIC running in a Dell PowerEdge 1950. This platform will allow us to deploy and test our new architecture before deploying it on the production network.

**Campus Deployment** The physical network currently supported by START was recently upgraded to include approximately 275 HP switches; these switches are capable of supporting the OpenFlow firmware once the architecture is ultimately deployed on the campus network. One of the more significant practical challenges in the campus deployment will be straining the scalability of the system on a production network without disrupting connectivity. For example, the proposed architecture may involve installing many flow table entries in the switches, which may either exhaust memory or slow lookup performance if entries are not stored efficiently, or if state is not offloaded to the controller whenever

possible. To address this concern, we will first stress-test the design on the research testbed and subsequently the architecture on a smaller number of production switches before completely rolling out the architecture on all 275 switches.

## 5. Related Work

Our architecture relies on flow-based operation, as advocated by OpenFlow [10], which has origins in the designs of earlier protocols, such as ATM [9, 14], and even early programmable switch architectures. We also draw inspiration from 4D [6] and Ethane [1], both of which advocate controlling network switches from a separate, logically centralized system. Although our approach is similar in spirit to these previous efforts that separate control from switches, we explore how this refactoring can help improve on and generalize existing security functions that already exist at higher layers and in middleboxes. For example, whereas an Ethane controller acts primarily as an intermediary for controlling traffic between hosts on an enterprise network, our approach explores how more complex security policies and actions (*e.g.*, actions based on alerts from distributed detection systems) could be more directly integrated into the network fabric and thus possibly allow significantly more dynamic, fine-grained, and flexible security policies than are feasible today.

NOX is a recently proposed “network operating system” that provides a uniform, centralized programmatic interface for a network [8]. Whereas NOX provides a platform for experimentation and research with new protocols, our proposed architecture should improve security by embedding security into the network itself. NOX could serve as a platform which we could use as the basis for our architecture.

Recent trends in packet forwarding architectures (*e.g.*, [2]) have tried to achieve a similar shift towards the lower layers by having the software part of the switch take the decisions of forwarding and pass it on to the hardware. Typically, the software and the hardware reside on the same machine. But, with such an architecture, its difficult to embed intelligent capabilities into the software, since there is no central aggregation of network-wide data. With our design, the central controller can do all the processing of the data collected from the monitors and send back the flow decisions to the switches through a fixed protocol.

## 6. Research Agenda

Today, we can only imagine very coarse-grained configuration changes like moving a user to the quarantine VLAN or a private VLAN. The proposed design that pushes security down the network protocol stack potentially simplifies many network security functions, and also allows dynamic, fine-grained security policies that may make it possible to eliminate dedicated intrusion prevention systems at higher layers altogether. We anticipate that as we build this system, we will encounter various research challenges. In our ongoing research, we are grappling with the following issues:

**Scale and Performance** When deploying the architecture on the campus network, we expect to encounter numerous

challenges involving scalability. For example, the Georgia Tech residential network must support approximately 16,000 users; the portion of the campus that runs START comprises more than 13,000 network ports, and future plans include expanding START to more than 40,000 active ports across academic buildings and merging START with the (separate) access control system currently used for the campus wireless network. A significant challenge will be implementing dynamic, fine-grained policies with flow-table entries, without exhausting switch memory or slowing forwarding. Recent proposals for optimizing customizable forwarding [2] may offer a useful starting point.

**Integration with Monitoring** As previously mentioned, the current network access control system scans hosts when they are first introduced into the network but cannot re-assign these hosts to different networks when they are deemed to be compromised. In our ongoing work, we will integrate alarms that arise from distributed monitoring and inference into mechanisms that can affect traffic flows more directly.

## REFERENCES

- [1] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker. Ethane : Taking control of the enterprise. In *SIGCOMM '07*, 2007.
- [2] M. Casado, T. Koponen, D. Moon, and S. Shenker. Rethinking packet forwarding hardware. In *Proc. Seventh ACM SIGCOMM HotNets Workshop*, Nov. 2008.
- [3] Cisco Application eXtension Platform Overview. [http://www.cisco.com/en/US/prod/collateral/routers/ps9701/white\\_paper\\_c11\\_459082.html](http://www.cisco.com/en/US/prod/collateral/routers/ps9701/white_paper_c11_459082.html).
- [4] N. Feamster and H. Balakrishnan. Detecting BGP Configuration Faults with Static Analysis. In *Proc. 2nd Symposium on Networked Systems Design and Implementation*, Boston, MA, May 2005.
- [5] N. Feamster, H. Balakrishnan, J. Rexford, A. Shaikh, and K. van der Merwe. The case for separating routing from routers. In *ACM SIGCOMM Workshop on Future Directions in Network Architecture*, Portland, OR, Sept. 2004.
- [6] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang. A clean slate 4D approach to network control and management. *ACM Computer Communications Review*, 35(5):41–54, 2005.
- [7] G. Gu, R. Perdisci, J. Zhang, and W. Lee. BotMiner: Clustering analysis of network traffic for protocol- and structure-independent botnet detection. In *17th USENIX Security Symposium*, 2008.
- [8] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker. NOX: towards an operating system for networks. *ACM SIGCOMM Computer Communication Review*, 38(3):105–110, 2008.
- [9] P. Newman, G. Minshall, and T. L. Lyon. IP Switching - ATM under IP. *IEEE/ACM Trans. Netw.*, 6(2):117–129, 1998.
- [10] OpenFlow Switch Consortium. <http://www.openflowswitch.org/>, 2008.
- [11] A. Ramachandran, N. Feamster, and S. Vempala. Filtering spam with behavioral blacklisting. In *Proc. 14th ACM Conference on Computer and Communications Security*, Alexandria, VA, Oct. 2007.
- [12] Georgia Tech Research Network Operations Center (GT-RNOC). <http://www.rnoc.gatech.edu/>.
- [13] Scanning Technology for Automated Registration, Repair and Response Tasks. <https://start.gatech.edu/>.
- [14] J. van der Merwe, S. Rooney, I. Leslie, and S. Crosby. The Tempest - A Practical Framework for Network Programmability. *IEEE Network*, 12(3):20–28, May 1998.
- [15] Configuring Dynamic Port VLAN Membership with VMPS. [http://www.cisco.com/univercd/cc/td/doc/product/lan/cat5000/rel\\_4\\_2/config/vmps.htm](http://www.cisco.com/univercd/cc/td/doc/product/lan/cat5000/rel_4_2/config/vmps.htm).