The Stanford OpenRoads Deployment

Kok-Kiong Yap*, Masayoshi Kobayashi◇, David Underhill*,
Srinivasan Seetharaman†, Peyman Kazemian*, and Nick McKeown*
*Stanford University, ◇ NEC System Platforms Labs, † Deutsche Telekom R&D Lab
yapkke@stanford.edu, m-kobayashi@eo.jp.nec.com, dgu@cs.stanford.edu,
srini.seetharaman@telekom.com, kazemian@stanford.edu, nickm@stanford.edu

ABSTRACT

We have built and deployed OpenRoads [11], a testbed that allows multiple network experiments to be conducted concurrently in a production network. For example, multiple routing protocols, mobility managers and network access controllers can run simultaneously in the same network. In this paper, we describe and discuss our deployment of the testbed at Stanford University. We focus on the challenges we faced deploying in a production network, and the tools we built to overcome these challenges. Our goal is to gain enough experience for other groups to deploy OpenRoads in their campus network.

Categories and Subject Descriptors

C.2.0 [**Computer Systems Organization**]: Computer-Communication Networks—*General*

General Terms

Management, Measurement, Experimentation

Keywords

Wireless Testbed, OpenRoads, OpenFlow

1. INTRODUCTION

In a recent poster we described OpenRoads [11], a way to create and run many network experiments at the same time in a *production* wireless network. In the spirit of GENI [6], OpenRoads allows both the datapath and control of a wireless network to be *sliced*, with each experiment running in its own isolated slice. A user's experiment might be a new routing protocol, a mobility manager, a network access controller, or an entire network management system. We have created and tested a number of experiments, and even run a class in which students each created and deployed their own mobility manager, all running simultaneously in the same network to manage a different set of mobile devices.

WiNTECH'09, September 21, 2009, Beijing, China.

In this paper, we describe our first deployment of OpenRoads in the Gates Computer Science Building at Stanford. While it is too early to draw broad conclusions about our operational experience with OpenRoads,[1] we can describe how it works, and the building blocks we have created to make the network easier to deploy and manage. All of our tools are (or shortly will be) available as open-source utilities for the community to use, build-on or modify. Our goal is to create a solid enough testbed at Stanford so that others may easily deploy OpenRoads networks on their campus. Our hope is for this to be possible by the end of 2009.

OpenRoads is built on several key technologies:

- *A sliceable datapath.* In order to isolate one experiment from another, OpenRoads is built on OpenFlow [12]. Each experiment is allocated its own "flow-space" (a range of header values), and may route and re-route packets within its slice.
- *A means to isolate slices.* Isolation in the sliceable datapath is enforced by FlowVisor [1, 18]. FlowVisor provides a means to virtualize an OpenFlow network by ensuring that each slice is isolated in its own flow-space.
- *A means to deploy experiments in a production network.* Our network consists of 30 WiFi access points (APs) and a WiMAX base station, all running OpenFlow. We deployed OpenRoads in the Gates Computer Science building at Stanford University, and many users use it as their production network.

Paper outline: In the next section we describe the OpenRoads testbed, and provide a basic introduction to OpenFlow. § 3 describes how we manage the testbed; in particular, how we combine multiple WiFi networks in different parts of campus to act as "one" network; how we slice the WiFi network; and how we manage IP addresses inside a production network where we don't control the production DHCP server. In § 4 we describe how we monitor the network. We describe related work in § 5 before concluding in § 6.

[1]We plan a follow-up paper to describe our experiences.

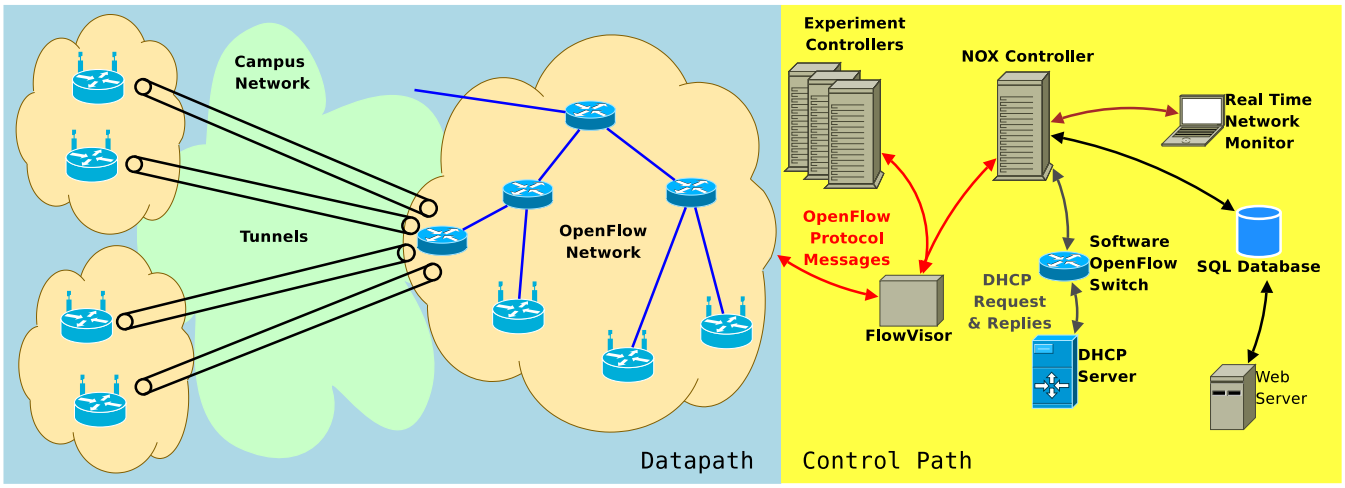


Figure 1: Testbed setup overview indicating the various management and monitoring components

2. THE OPENROADS WIRELESS TESTBED AT STANFORD

OpenFlow Primer: OpenFlow is a feature that is added to Ethernet switches, routers, WiFi APs and WiMax basestations. OpenFlow provides a standard, open interface to control the "flow-table" that is present in almost all modern switches and routers. The flow-table consists of multiple flow-entries, each matching on a user-defined set of packets (e.g. all packets with a given IP prefix, a given MAC address, a TCP port number, or some combination). If a packet matches a flow-entry, then it is processed according to a specific action: e.g., forward, drop, or modify the packet. OpenFlow is described in [12], and several reference implementations are available at [14].

OpenRoads uses OpenFlow as a way to control the APs and switches in the network datapath. For example, packets are routed by setting up flow-entries in a sequence of switches; flows can be re-routed (e.g. by a mobility manager) by changing the flow-entries using the OpenFlow API. Typically (though not always) flow-entries are established reactively: When the first packet in a flow arrives to the first switch or AP, it "misses" in the flow-table and is sent to the controller, which decides whether to accept the flow and if so, picks a route and sets up flow-entries in the switches along the path.[2] Once a flow has been established, all future packets in the flow are forwarded in hardware.

An OpenFlow network needs a controller. In principle, a researcher can use any controller that speaks the OpenFlow protocol, including creating their own controller. In practice, it is often easier to use an existing controller; in our network we typically use NOX [7]. When a researcher creates an experiment, they create a new module in NOX to decide how flows are to be processed. An open-source version of NOX is available at [13].

If we want to run multiple experiments at the same time, we need a way to run multiple controllers, and allow each controller to run different parts of the network. OpenRoads uses FlowVisor [18] to slice up the OpenFlow network, and give each experiment its topology and "flow-space" (i.e. range of header values) for their experiments. FlowVisor connects a controller to a particular slice of the network (a sub-topology and flow-space), and runs a policy to isolate the controllers from each other. The reader is encouraged to read [18] to learn more about FlowVisor. In brief, FlowVisor connects to a network of OpenFlow switches using the OpenFlow protocol; all of the switches think they are controlled by the FlowVisor. All the experimenters' controllers connect to the FlowVisor, but believe they are connecting to their own private network of OpenFlow switches. The FlowVisor manages the multiplexing and demultiplexing of OpenFlow control messages to create the illusion of multiple independent OpenFlow networks.

In our testbed, we deployed five 48-port 1GE OpenFlow Ethernet switches, 30 WiFi APs, and 1 WiMAX basestation. The testbed includes switches from NEC (IP8800) and HP (ProCurve 5406ZL); both are OpenFlow-enabled through a prototype firmware upgrade. The WiMAX basestation is built by NEC. The WiFi APs are based on the ALIX PCEngine boxes with dual 802.11g interfaces. The APs run the Linux-based software reference switch from the OpenFlow website, and are powered by passive power over Ethernet to reduce the cabling needed for our deployment. Figure 2 shows the location of our these APs throughout the Gates Computer Science building.

Figure 1 illustrates the various management and monitoring components of our deployment at Stanford University's Gates building. We will elaborate on each component later in the paper.

3. MANAGEMENT

3.1 Building an OpenRoads network across campus

Figure 1 shows our OpenFlow-enabled WiFi access points spread across our campus network, in many different locations. Ideally, we would like to place APs anywhere in the network, and have them automatically "connect" to the OpenRoads testbed. This means two things have to happen: The AP needs to become part of the OpenRoads datapath, by tunneling flows to/from the rest of the OpenRoads network; and it needs to associate with, and be controlled by, the FlowVisor (and experimenters' controllers behind it).

[2]Flow-entries can also be setup proactively in advance.

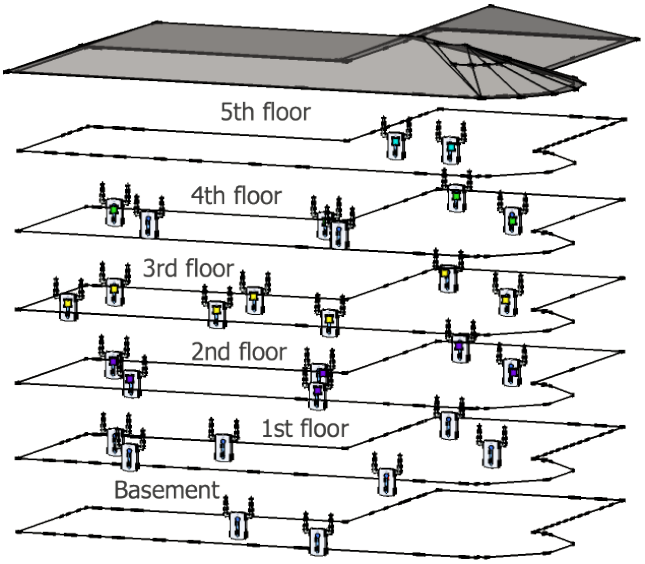


Figure 2: Location of WiFi APs in Stanford Gates building

Wireless APs connect to the OpenRoads datapath using a lightweight tunneling tool called *Capsulator* [5]. As illustrated in Figure 1, the Capsulator tunnels packets between the OpenFlow network and wireless APs across the campus network. Its operation is summarized in Figure 3.

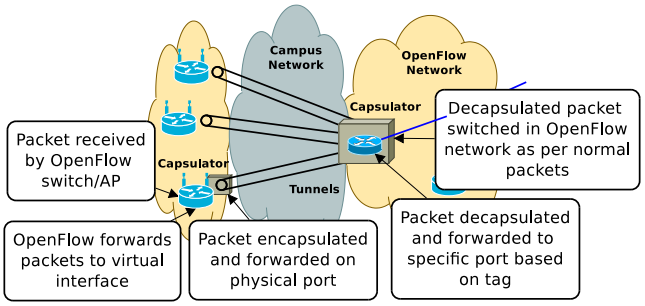


Figure 3: Example showing workings of Capsulator

The Capsulator runs on every OpenRoads AP, and creates a tunnel for packets to/from our testbed. Capsulator is a user-space program that encapsulates Ethernet frames in IP packets (protocol number = 245), and adds a 4-byte value to identify the AP the packet came from (and hence the experiment it belongs to).

Because our wireless APs only have one physical Ethernet port, the AP needs to distinguish the tunneled traffic from regular production traffic (in its location), and so we create virtual network interfaces using TUN / TAP.

3.1.1 Cost of Tunneling

The Capsulator adds an overhead of 38 bytes per packet (4 byte tag, 20 byte IP header, and 14 byte Ethernet header). If the size of an encapsulated packet exceeds the maximum transmission unit (MTU), then the packet is fragmented, resulting in another 34 bytes of overhead. To characterize the effect of this overhead in our traffic, we collected an 8 hour trace which consisted of more than 13.9 million packets. Only 0.31% of these packets were fragmented.

We also quantify Capsulator's impact on both delay (using ping) and throughput (using iperf). Both results are summarized in Table 1. Round-trip time (RTT) is increased by 5.4% or 0.08 milliseconds. TCP throughput is decreased by 3.4% or 0.9 *Mbps*. This performance degradation is acceptable for our deployment.

Table 1: Capsulator overhead

	Performance (± std dev)	
	w/o Capsulator	with Capsulator
RTT (ms)	1.39 ± 1.55	1.47 ± 1.27
Xput (Mbps)	26.4 ± 3.33	25.5 ± 3.31

3.2 Slicing an AP using SSIDs

OpenRoads allows an end-user to opt-in to (one or more) experiments. We do this by assigning a different SSID to each experiment, which requires each AP to support multiple SSIDs.

An experiment runs inside its own "slice" of resources - we create a slice from a combination of multiple SSIDs and virtual interfaces. When a slice is created, we create a virtual WiFi interface on all of the APs in the slice's topology, and assign a unique SSID to the interface. Since each experiment can be assigned a distinct SSID, users may opt-in to an experiment by simply choosing an SSID. Detailed instructions are provided in [2].

Using virtual interfaces is easy on our APs because they run Linux. Although more expensive than the lowest-cost commodity APs (our box is about $120; a low-cost WiFi today is about $40), our APs cost less than a typical enterprise AP. And the same idea could be applied to a low-cost AP running OpenWRT.

Using a separate SSID for each experiment also means each SSID can use different encryption and authentication settings. However, all the virtual interfaces are limited to use the same wireless channel and power settings.

Each SSID (i.e. slice) is part of a different experiment, and is therefore attached to a different controller (created by the experimenter). FlowVisor is responsible for connecting each slice to its own controller.

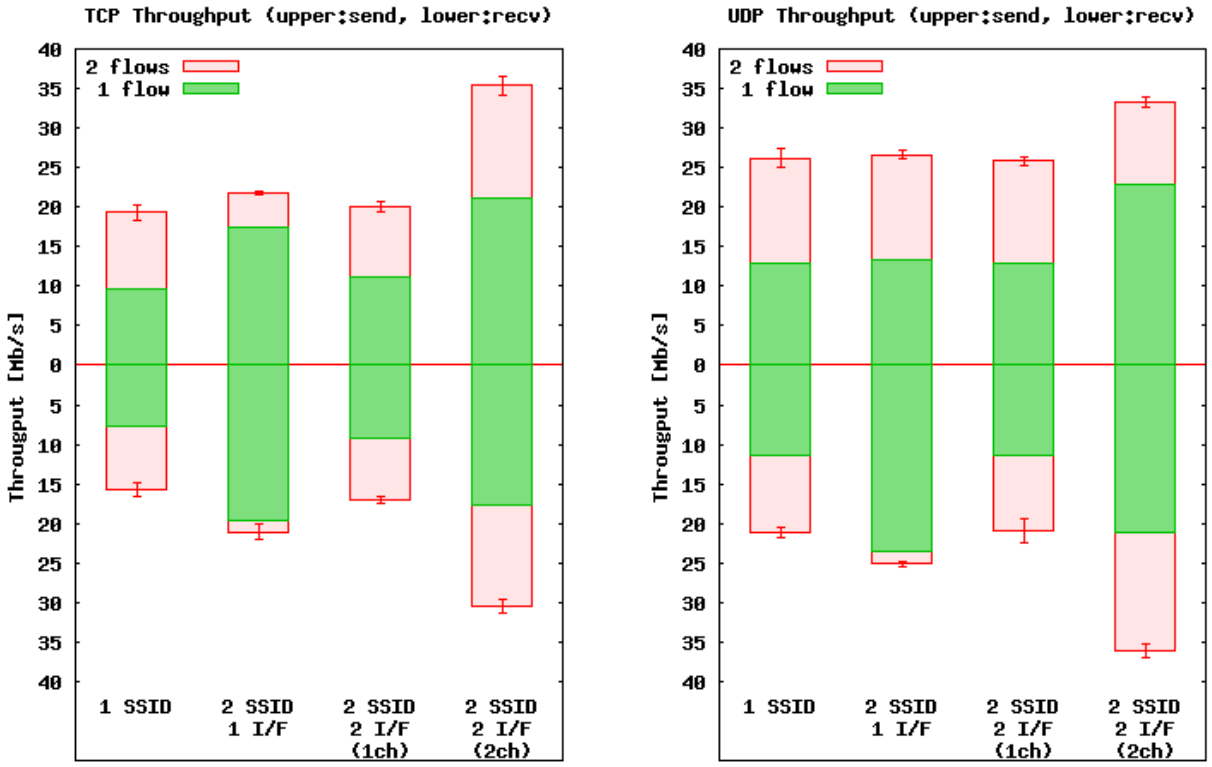
3.2.1 Performance with Multiple SSIDs

We were curious to know how multiple SSIDs and virtual interfaces would affect the performance of the wireless network (and hence degrade the performance "isolation" between slices): To evaluate the performance impact of having multiple SSIDs on a single wireless interface, we compared the following cases with iperf [9]:

1 SSID 2 iperf sessions over a single WiFi interface which have 1 SSID.

2 SSID 1 iperf session over each of 2 virtual WiFi interfaces (on a single physical WiFi interface). Each virtual interface has a different SSID.

2 SSID 2 I/F (1ch) An iperf session over each of 2 physical WiFi interfaces. Both interfaces are tuned to the same channel.



(a) TCP

(b) UDP

Figure 4: Comparison of throughput in different multiplex methods

2 SSID 2 I/F (2ch) An iperf session over each of 2 physical WiFi interfaces. The interfaces are tuned to non-overlapping wireless channels (channel 1 and channel 11).

We ran 10 iperf sessions (each lasting 30 seconds) and calculated the average throughput and its standard deviation. Figures 4(a) and 4(b) show the results of TCP and UDP measurements respectively. Each figure shows both the total throughput and standard deviation of the 2 iperf sessions as well as the throughput of 1 iperf session. The upper half of each graph corresponds to data being sent from the AP to the wireless clients. The lower half of each graph corresponds to data being sent from wireless clients to the AP.

Surprisingly, regardless of which protocol (TCP/UDP) is used or the direction traffic is sent, the 2 SSID case outperforms the 1 SSID case. This is likely due to the lack of coordination between the iperf sessions. With 2 SSIDs, the AP allows 1 flow to maximize its throughput. Similarly, the 2 SSID case always outperforms 2 SSID 2 I/F (1ch). In the 2 SSID 2 I/F (1ch) case, the 2 physical wireless interfaces are working independently and interfere with each other as a result. The reason 2 SSID 2 I/F (2ch) always works best is simple – 2 wireless channels can operate with minimal interference. This result shows that using multiple SSIDs is a feasible way of multiplexing the network if fairness is not an issue. Otherwise, it might be better to use multiple interfaces. Both methodologies can be supported in our platform, since we have built them instead of using commodity APs. Such a choice will have to be made for each deployment of OpenRoads.

3.3 Allocating IP addresses to mobile clients

Because OpenRoads is built on OpenFlow, it doesn't require all experiments to use IP – so long as an experiment sends/receives legal WiFi packets, it is free to modify the layers above, so long as an OpenFlow match can correctly match on the headers. However, for the time being, we expect most experiments to use IP, and therefore we need to allocate an IP address to the users who opt-in to an experiment; and make sure they are allowed to opt-in.

It is likely that an OpenRoads network will be created as part of, but not entirely replace, a campus production network. It will therefore be allocated a set of IP addresses to manage independently of the production network. For example, in our network at Stanford, we were allocated a block of IP addresses in the middle of a subnet for allocation to our wireless clients, and the task of serving these addresses was delegated to us. We needed to verify that the requesting client is registered with the campus, and is eligible to opt-in to an experiment. This cannot be easily achieved in a conventional network because one cannot use multiple DHCP servers in a single broadcast domain.

We solved this using OpenFlow's control of the datapath. By redirecting DHCP requests we allow two or more DHCP servers in the same broadcast domain. Our approach is shown in Figure 1.

Using OpenFlow, we identify all DHCP requests that are from one of our wireless APs. If not, then the request is broadcast (as usual) and will be handled by the campus-wide DHCP server. Otherwise, we check the client's registration via a *whois* server or a list of MAC addresses registered with us (other forms of authentication could easily be implemented). If the client is registered, we forward the request to our own DHCP server and an IP address is allocated. Replies are unicast to the client if possible, and otherwise they are broadcast to all clients which made a recent request.

In our deployment, the DHCP runs on the same physical server as the NOX controller, and is connected to the network via a software Linux-based OpenFlow switch on the same machine. By deploying our own DHCP server we have been able to customize the authentication process to fit our needs.

3.3.1 Efficiency

To test the efficiency of our approach, we measured the response time of DHCP requests and compare it to the campus-wide DHCP server. The result, based on 100 samples for each network, is presented in Table 2. Here, each DHCP request is initiated after association with a new wireless AP.

Table 2: DHCP response time

	Response Time (s)		
	Min	Average / Std Dev	Max
OpenFlow	1.08	3.52 / 3.09	22.1
Campus	11.16	14.0 / 1.59	19.6

We found that our DHCP service typically provides a faster response time than the campus-wide DHCP server. While this result may be specific to our deployment, it is encouraging that OpenFlow can be used efficiently for a critical network application.

4. MONITORING

4.1 Logging

Typical measurements of traffic statistics today require "chokepoint(s)" through which all network traffic must pass. This creates two obvious issues. First, all traffic passing through the chokepoint(s) will be logged, making opt-in more difficult. Further, chokepoint(s) can be hard to implement in a large network, especially if traffic is load-balanced over multiple routers.

In contrast, we can collect data and statistics from every single switch in an OpenFlow network — there is no need for chokepoint(s). For example, a flow expiration message is sent to the controller whenever a flow has expired in the switch. This message contains the duration of the flow as well as how many packets and bytes were forwarded as part of the flow. We can use this to determine the average rate of the flow and what flows are in the network at a given time. Also, our system logs information available via SNMP. For example, we log the association of clients to WiFi APs. This information can be used to reconstruct user movement in the network.

Further, an OpenFlow network allows users to opt-in for data collection. In fact, OpenFlow provides fine-grained control over not just whose traffic can be logged, but also what traffic can be logged. Using the FlowVisor, we can filter the flow expiration messages seen by the controller of a given researcher. This ensures that each researcher only sees the permitted traffic of test subjects which have opted-in. For example, Alice can allow an experimenter to solely access her HTTP traffic statistics without revealing other traffic statistics such as SSH. This cannot be achieved via the common SNMP available in switches.

In our system, we store information using the SQLite database provided in NOX. This data is periodically transferred into a MySQL database. This is depicted in Figure 1. All logs are timestamped. This allows temporal aspects of data to be explored and correlated. An application for which we develop the graphing tool described in §4.2 for.

4.1.1 Use Case: Traffic Analysis

One way we use our logs is to analyze our network's traffic. Table 3 shows the breakdown of traffic by type. This data was collected over a one week period from May 31 to June 6, 2009. During this time, 11 million flows transferred 263 million packets carrying 41GB of data. The average flow size was about 22 packets and 159 bytes.

Table 3: Traffic breakdown of our network

Traffic	Percent of Total		
Type	Flow Entries	Packets	Bytes
HTTP	31.07	41.57	71.74
SSH	0.35	20.13	13.83
ARP	23.87	17.39	6.55
L2 Broadcast	4.90	1.37	1.90
NTP	0.37	1.45	1.51
DNS	29.35	1.36	0.96
Others	3.74	0.84	0.31
IMAP	1.15	0.05	0.02
OpenFlow	0.05	11.28	0.01
Not IP/ARP	0.01	0.00	0.00

The dominance of HTTP and SSH traffic in our wireless network is expected since two common activities of our users include surfing the web and remotely accessing machines via SSH. The high level of ARP activity is due to an ongoing experiment. The devices involved in the experiment use a primitive networking stack which does not cache ARP replies.

Also, the APs use in-band control to communicate with their OpenFlow controller. Therefore, the control traffic of the 30 APs is carried by the backbone itself. This constitutes 11.3% of all packets, though these packets are generally small (they are only 0.01% of all traffic by volume) and use only 0.05% of the available flow entries. Thus OpenFlow has not added much overhead in terms of traffic volume or flow entries in our deployment.

4.2 Data Graphing Tool

Distilling useful information from the large volume of data generated from our monitoring operations (from §4.1 and other sources) is difficult. Furthermore, it is difficult to see the temporal variation and cross-correlation between different flow characteristics. This is an important requirement for accounting and debugging purposes. One way to look at this is that it is a data visualization problem. Being able to plot various metrics for visual inspection, along similar axes,

would allow an administrator to deduce the various activities and issues in the network. It is also critical to observe that it is not clear out front which metrics are interesting.

In response to this data visualization problem, we built a custom network data graphing tool (downloadable from [3]). This web-based tool extracts and plots information from our MySQL database (as shown in Figure 1). This graphing tool allows a network administrator to compose custom queries through a web interface, lookup the MySQL database, and plot results on the returned webpage. The query constraints can be any combination of the following: a) Time range of the logs, b) Exact value match on certain fields, c) Additional SQL-like queries. Furthermore, the graphing tool produces three different types of plots and we expect more to be added in the future. Such interactivity is crucial since what the interesting metrics are is not immediately clear, and therefore administrators need to mix and match metrics on the fly for visual inspection.

Figure 5 illustrates several plots which is generated through this tool's web-based interface. For example, one may compare the number of flows and the numbers of users in the network to see if a correlation exists at any point in time. The cumulative distribution function of flow duration and packet size can also be plotted to determine the median. Further, the interactive web interface allows logged data to be efficiently analyzed and enhances our ability to diagnose issues in the network.

4.3 Real-Time Visualization

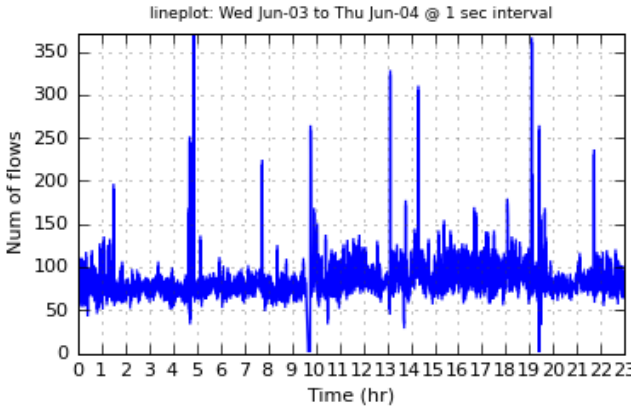
In addition to a tool for graphing logged data (§4.2), we also developed an open-source, real-time network monitoring tool [21] on top of the ENVI[20] network visualization framework. This tool can visualize data supplied by the testbed's OpenFlow controller (§ 2). This includes the network topology, link utilization, switch processor utilization, user distribution, and even individual flows if desired. This allows us to quickly visualize the current state of the network.

Figure 6 shows a screenshot of this tool running in our testbed. The blue circles represent core switches, while green circles represent wireless access points. The figure illustrates that a large number of access points can be connected to a single core switch — many of these are tunneled from remote locations as previously discussed in the tunneling section (§ 3.1). When the mouse hovers over a node, additional information about that node is shown. In the figure, the OpenFlow datapath ID, switch vendor, IP address, hostname, and switch version are shown for a particular access point. Links are traffic-light color-coded based on utilization — most links in this screen capture are un-utilized (black).

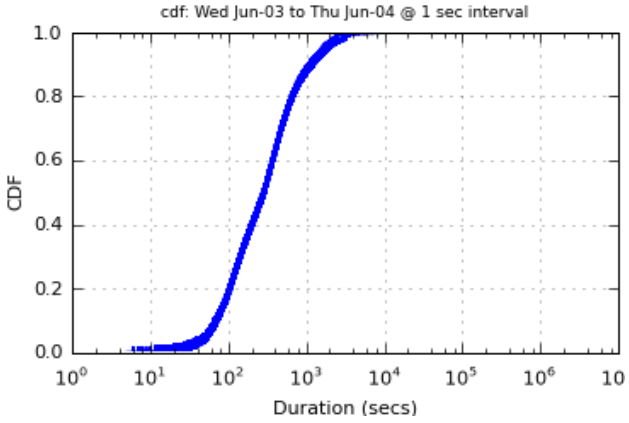
This tool has proven to be particularly useful for diagnosing network connectivity problems. The alternative to using this tool is to manually ping across each link in the network — a tedious process in a large network.

5. RELATED WORK

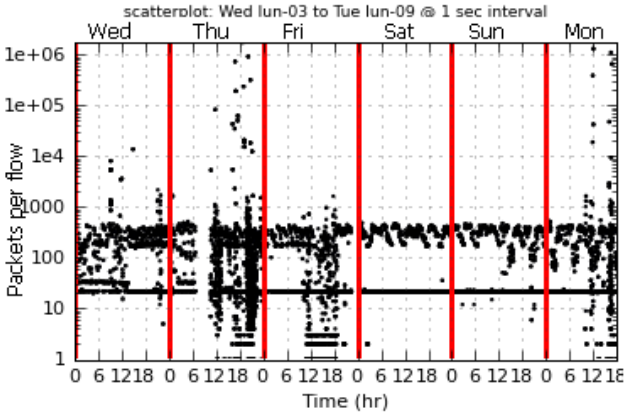
Accurate simulation or emulation of a wireless environment is difficult. As a result, many wireless testbeds [4, 16, 17, 19, 25, 22, 10] have been deployed. These testbeds concentrated on a wireless mesh network with little or no control of the wired backbone. However, a lot of experience was gained through deployment of these testbeds and some has even led to companies (e.g., Meraki).



(a) Lineplot of number of flows



(b) CDF of the duration of each flow



(c) Scatterplot of packet count of *ssh* flows (DstTcpPort=22)

Figure 5: Sample plots from the graphing tool.

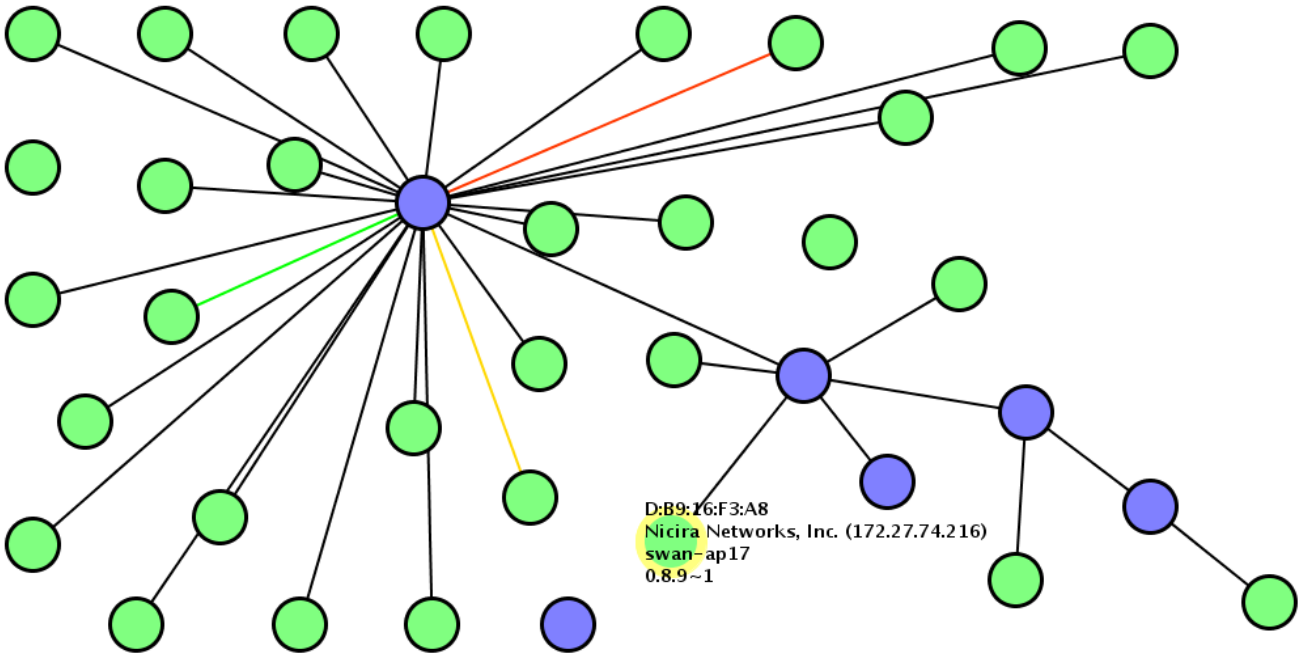


Figure 6: A screenshot of the testbed using a network visualization tool we developed to enhance our ability to monitor the testbed in real-time.

At the other end of the testbed spectrum are wired networking testbeds such as VINI [23], PlanetLab [15] and Emulab [24]. These networks have concentrated on enabling experimentation in wired networks, with the exception of Emulab which incorporated some wireless APs. Hadjichrostofi et. al. [8] also proposed an integrated wired-wireless testbed which uses both VINI and ORBIT.

While our community has substantial experience in deploying wired and wireless testbeds, deploying a mobile wireless testbed which includes control of a wired backbone network poses unique challenges such as managing users and opt-in experiments. In such a testbed, we can monitor actual user movement and traffic. This in turn allows realistic evaluations of new protocols and research ideas.

6. CONCLUSION

To evaluate our research in mobile networks at-scale we created OpenRoads. This testbed enables us to experiment in our production network. It also allows us evaluate and build upon one anothers' work. We hope that lowering the barrier of entry to realistic evaluation will help researchers innovate in mobile wireless networks, and allow us to move forward rapidly as a community.

Further, a common platform allows us to benefit from one anothers' experience and tools. Some of these tools may cater to researchers, while others may help manage production and experimental networks. In a hope to seed this process, we have open-sourced our tools for others to use and improve upon. The tools were carefully constructed to be complementary yet independent. As such, each tool can be reused for other deployments.

Managing the network and supporting many live experiments in the midst of a production network is challenging. We hope our experience and tools will make deploying similar testbeds in other campuses easier.

7. ACKNOWLEDGMENTS

This work presented is made possible by many people. We'd like to thank many people at NEC and HP for providing prototype OpenFlow switches. We are grateful to Ippei Akiyoshi for his contribution towards incorporating WiMAX into the network. Advice from the people at Nicira has also helped greatly, with Martin Casado and Ben Pfaff deserving special mention. The assistance rendered by Rob Sherwood, Miles Davis, Charlie Orgish, Jiang Zhu and Guru Parulkar is also greatly appreciated.

8. REFERENCES

[1] FlowVisor. `http://www.openflowswitch.org/wk/index.php/FlowVisor`.

[2] OpenFlow AP with PC Engine. `http://www.openflowswitch.org/wk/index.php/OpenFlow_AP_with_PC_Engine`.

[3] Openseer data graphing tool. `http://www.openflowswitch.org/wk/index.php/Deployment`.

[4] J. Bicket, D. Aguayo, S. Biswas, and R. Morris. Architecture and evaluation of an unplanned 802.11b mesh network. In *MobiCom '05: Proceedings of the 11th annual international conference on Mobile computing and networking*, pages 31–42, New York, NY, USA, 2005. ACM.

[5] Tunneling Software for OpenFlow Deployment. `http://www.openflowswitch.org/wk/index.php/Tunneling_Software_for_OpenFlow_Deployment`.

[6] GENI.net Global Environment for Network Innovations. http://www.geni.net.

[7] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker. NOX: Towards and operating system for networks. In *ACM SIGCOMM Computer Communication Review*, July 2008.

[8] G. C. Hadjichristofi, A. Brender, M. Gruteser, R. Mahindra, and I. Seskar. A wired-wireless testbed architecture for network layer experimentation based on orbit and vini. In *WinTECH '07: Proceedings of the the second ACM international workshop on Wireless network testbeds, experimental evaluation and characterization*, pages 83–90, New York, NY, USA, 2007. ACM.

[9] C. hsing Hsu, U. Kremer, and P. P. Models. Iperf: A framework for automatic construction of performance prediction models. In *In Workshop on Profile and Feedback-Directed Compilation (PFDC*, 1998.

[10] R. P. Karrer, I. Matyasovszki, A. Botta, and A. Pescapé. Experimental evaluation and characterization of the magnets wireless backbone. In *WiNTECH '06: Proceedings of the 1st international workshop on Wireless network testbeds, experimental evaluation & characterization*, pages 26–33, New York, NY, USA, 2006. ACM.

[11] Kok-Kiong Yap, Masayoshi Kobayashi, Rob Sherwood, Nikhil Handigol, Te-Yuan Huang, Michael Chan, and Nick McKeown. OpenRoads: Empowering research in mobile networks. In *Proceedings of ACM SIGCOMM (Poster)*, Barcelona, Spain, August 2009.

[12] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, April 2008.

[13] NOX: An OpenFlow Controller. http://noxrepo.org/wp/.

[14] The OpenFlow Switch Consortium. http://www.openflowswitch.org.

[15] An open platform for developing, deploying, and accessing planetary-scale services. http://www.planet-lab.org/.

[16] Purdue wireless mesh network. https://engineering.purdue.edu/mesh.

[17] D. Raychaudhuri, I. Seskar, M. Ott, S. Ganu, K. Ramach, H. Kremo, R. Siracusa, H. Liu, and M. Singh. Overview of the orbit radio grid testbed for evaluation of next-generation wireless network protocols. In *in Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1664–1669, 2005.

[18] Rob Sherwood, Michael Chan,Glen Gibb, Nikhil Handigol,Te-Yuan Huang,Peyman Kazemian,Masayoshi Kobayashi, David Underhill, Kok-Kiong Yap, and Nick McKeown. Carving research slices out of your production networks with OpenFlow. In *Proceedings of ACM SIGCOMM (Demo)*, Barcelona, Spain, August 2009.

[19] Y. Su and T. Gross. Validation of a miniaturized wireless network testbed. In *WiNTECH '08: Proceedings of the third ACM international workshop on Wireless network testbeds, experimental evaluation and characterization*, pages 25–32, New York, NY, USA, 2008. ACM.

[20] D. Underhill. An Extensible Network Visualization and Control Framework. Master's thesis, Stanford University, May 2009.

[21] D. Underhill et al. Network monitor. http://www.openflowswitch.org/wp/gui.

[22] N. H. Vaidya, J. Bernhard, V. V. Veeravalli, P. R. Kumar, and R. K. Iyer. Illinois wireless wind tunnel: a testbed for experimental evaluation of wireless networks. In *E-WIND '05: Proceedings of the 2005 ACM SIGCOMM workshop on Experimental approaches to wireless network design and analysis*, pages 64–69, New York, NY, USA, 2005. ACM.

[23] A virtual network infrastructure. http://www.vini-veritas.net.

[24] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. In *Proc. of the Fifth Symposium on Operating Systems Design and Implementation*, pages 255–270, Boston, MA, Dec. 2002. USENIX Association.

[25] J. Zhou, Z. Ji, M. Varshney, Z. Xu, Y. Yang, M. Marina, and R. Bagrodia. Whynet: a hybrid testbed for large-scale, heterogeneous and adaptive wireless networks. In *WiNTECH '06: Proceedings of the 1st international workshop on Wireless network testbeds, experimental evaluation & characterization*, pages 111–112, New York, NY, USA, 2006. ACM.