Blueprint for Introducing Innovation into Wireless Mobile Networks

Kok-Kiong Yap[†] Rob Sherwood[◇] Masayoshi Kobayashi[⋆] Te-Yuan Huang[†]
Michael Chan[†] Nikhil Handigol[†] Nick McKeown[†] Guru Parulkar[†]

[†]Stanford University [◇]Deutsche Telekom Inc. R&D Lab [⋆]NEC

yapkke@stanford.edu, robert.sherwood@telekom.com, m-kobayashi@eo.jp.nec.com,
{huangty, mcfchan, nikhilh, nickm, parulkar}@stanford.edu

ABSTRACT

In the past couple of years we've seen quite a change in the wireless industry: Handsets have become mobile computers running user-contributed applications on (potentially) open operating systems. It seems we are on a path towards a more open ecosystem; one that has been previously closed and proprietary. The biggest winners are the users, who will have more choice among competing, innovative ideas.

The same cannot be said for the wireless network infrastructure, which remains closed and (mostly) proprietary, and where innovation is bogged down by a glacial standards process. Yet as users, we are surrounded by abundant wireless capacity and multiple wireless networks (WiFi and cellular), with most of the capacity off-limits to us. It seems industry has little incentive to change, preferring to hold onto control as long as possible, keeping an inefficient and closed system in place.

This paper is a "call to arms" to the research community to help move the network forward on a path to greater openness. We envision a world in which users can move freely between any wireless infrastructure, while providing payment to infrastructure owners, encouraging continued investment. We think the best path to get there is to separate the network service from the underlying physical infrastructure, and allow rapid innovation of network services, contributed by researchers, network operators, equipment vendors and third party developers.

We propose to build and deploy an open—but backward compatible—wireless network infrastructure that can be easily deployed on college campuses worldwide. Through virtualization, we allow researchers to experiment with new network services directly in their production network.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
VISA 2010, September 3, 2010, New Delhi, India.
Copyright 2010 ACM 978-1-4503-0199-2/10/09 ...$10.00.

Categories and Subject Descriptors

C.2.1 [**Computer Systems Organization**]: COMPUTER-COMMUNICATION NETWORKS—*Network Architecture and Design*

General Terms

Design, Experimentation, Measurement

Keywords

OpenFlow, Software-Defined Networks, FlowVisor, Wireless, Mobile Networks, Mobility

1. INTRODUCTION

There is currently much excitement in the air about openness in wireless and mobile computing. Users can choose from a thriving array of handsets and, in many countries, use their handset with a variety of commercial carriers. A burgeoning army of third-party developers are creating applications, games and content for mobile devices. And the Android operating system claims to be the "first truly open and comprehensive platform for mobile devices; all of the software to run a mobile phone, but without the proprietary obstacles that have hindered mobile innovation." [4]

Arguably these are all positive steps towards a more open ecosystem for the mobile world, creating more choice for users. We applaud the move towards openness and are great believers in the power of choice in the marketplace to bring innovation, efficiency and high quality service to the user. Industry benefits too. An innovative marketplace grows the business for everyone, and attracts new players eager to compete with incumbents. Openness creates choice which breeds innovation.

But despite all the progress, there are still real structural barriers to openness – barriers that industry and government will not break down on their own – requiring technical innovation. This is the domain of university researchers. In the spirit of enabling choice and innovation through openness, we describe here how we, as a research community, can set out to break down two large technological barriers:

1. **The inaccessible and closed wireless capacity around us.** Today, if we stand in the middle of a city, we can likely "see" multiple cellular and WiFi networks. But, frustratingly, these infrastructures are not available for us to use. Cellular companies restrict us to use

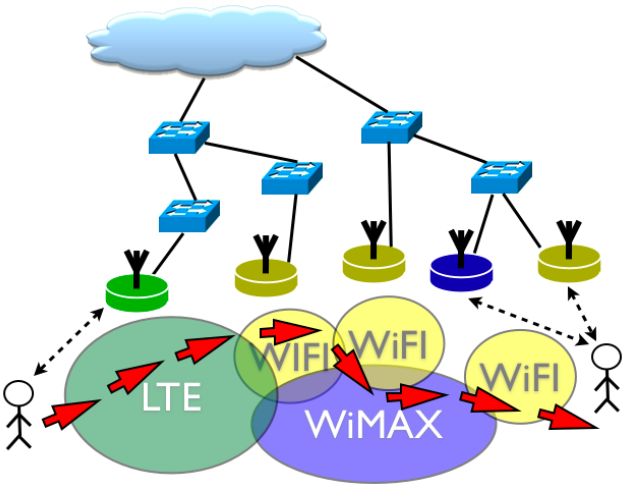


Figure 1: Vision for Future Wireless Mobile Internet: Choice of providers, networks and technologies.

their network; most private WiFi networks require authentication, and are effectively inaccessible. Although we are often surrounded by abundant wireless capacity, almost all is off-limits; our choice is almost non-existent. We believe users should be free to travel in a rich field of wireless networks with access to all infrastructure around them. Openness doesn't mean free; here it means a healthy market-place with lower-cost connectivity and broader coverage. In the extreme, if all barriers to fluidity can be removed, users could connect to multiple networks at the same time, opening up enormous capacity and coverage.

2. **A network infrastructure that is closed to innovation.** Cellular networks increasingly use IP. IP has been tremendously successful in bringing choice and innovation to the end user: Arguably its greatest feat is enabling innovation at the edges. IP is simple, standardized, and provides universal connectivity. But we believe that as-is, IP is not the right choice for the future mobile Internet: It is ill-suited to support mobility and security; and it is hard to manage. Its architecture is fixed, allowing little room to add new capabilities. Today cellular providers feel the pain from poor support for mobility, security, and innovations in general. If we tweak IP to solve these problems, we will find new limitations. We need a network that allows continued innovation, for services we can't yet imagine, while allowing existing applications to work unchanged.

So as we look to the future, we want a network that will allow any mobile computer to connect to any network, and to move freely and seamlessly from one network to another. The logical next step is for a handheld to connect to any network around it – regardless of who owns the network and what radio technology it uses, as exemplified in Figure 1. While there are obvious non-technical barriers that stand in our way, e.g., economic and regulatory, we believe a new network architecture is needed to break down these barriers.

In this new architecture, there will exist lots of service providers, lots of radios, and lots of types of radios, all tied together by lots of wired networks. There will be diversity at all levels: diversity in network (many networks to choose from), channels (more spectrum will become available), antennas (more MIMO), radios (a handheld will contain many radios). Whereas today's phones commonly have three or four radios (e.g. GSM, WiFi, Bluetooth), in future they will have more. Shrinking geometries and energy-efficient circuit design will lead to mobile devices with ten or more radios, with several of the same type. A handheld may connect to several networks at once, for robustness and increased signal quality. If users are to move freely among many networks, the service provider needs to be separate from the network owner. Service provider should handle the mobility, authentication and billing for their users, regardless of the network they are connected to. To a limited extent, this is happening: Some cellular companies allow MVNOs[1] to provide services over their network. And in WiFi networks, when we login to a hotel or airport network, a third party provides authentication and billing services. We assert that all these will allow streaming applications to operate seamlessly as we walk, drive, or fly.

Our general approach is to open up that which has been closed – to help industry move towards an open network architecture. As cellular providers make the transition to IP, we would like to enable them to innovate in their own network. We would like them to be able to research and experiment with new security models (e.g. new approaches to access control, and user authentication), and with new, more scalable alternatives to mobile IP. An open architecture allows a new industry of suppliers to provide improved features in the network; and it will help an open-source community to grow, which in turn will provide contributions to all users.

We believe the research community has a big part to play in bringing this new open architecture to fruition, and there are many interesting research problems to be solved along the way. For example, in service of end-users, we need to figure out how to cleanly cleave the network service from the underlying network infrastructure. A clean separation of "service from infrastructure' across different networks (cellular providers, home networks, enterprise networks, coffee-shop networks, etc.) and across different types of wireless network (e.g. GSM, WiFi, WiMAX, LTE) would give us access to a lot more wireless capacity, and more competition among providers. Other research includes how to create a personalized mobility manager that lives in the cloud in service of one or more customers. A personal mobility manager can implement a user's preferences for routing, network selection and pricing. An open network will enable new experiments with location-aware services. And it will enable experiments with new, large scalable directory services for a population of billions of mobile users and services. Finally, we can research ways to improve measurement and instrumentation of the network, to allow users to compare service quality from different providers and in different networks.

To support this vision and research along the way, we present OpenFlow Wireless (§ 2), a blueprint for this open network architecture. We also describe the current deployment of OpenFlow Wireless (§ 3) on our campus, and how it

[1]MVNO: Mobile Virtual Network Operator. In the US, Virgin is an MVNO in Sprint's network; Sprint owns the radios and wired network, and Virgin provides branded AAA and billing services for its customers. In some countries, notably Holland, hundreds of MVNOs compete over a small number of physical networks.

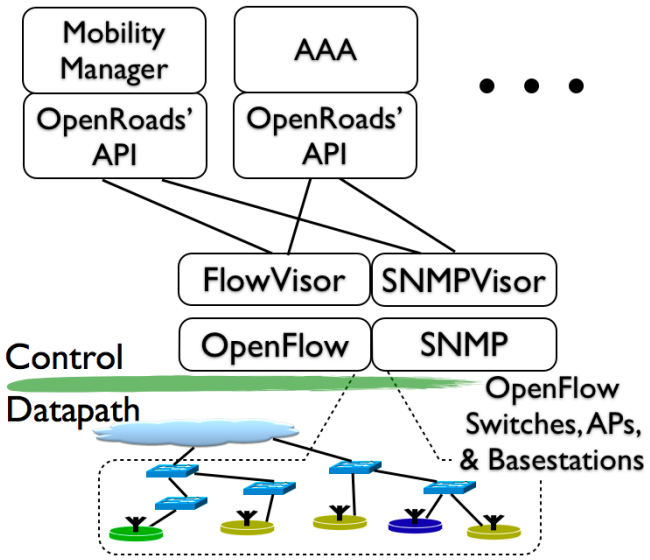


Figure 2: OpenFlow Wireless (a.k.a. OpenRoads) Architecture

has already been used in classrooms to enable new research in wireless networks (§ 3.3). We conclude the paper with a call to action for the research community to join in our expedition towards this vision of openness (§ 4).

2. OPENFLOW WIRELESS: BLUEPRINT FOR AN OPEN WIRELESS NETWORK

In support of our vision – and as a first step towards engaging the broader research community – we propose OpenFlow Wireless: a mobile wireless network platform enabling experimental research and realistic deployments of networks and services using virtualization. Figure 2 provides an overview of OpenFlow Wireless. OpenFlow Wireless uses OpenFlow to separate control from the datapath through an open API, FlowVisor [12] to create network slices and provide isolation among them, hence allowing multiple experiments to run simultaneously in *production* wireless network, and SNMPVisor to mediate device configuration access among experiments. These components which virtualize the underlying infrastructure relate directly back to our vision for future wireless Internet design, in terms of decoupling mobility from physical network (OpenFlow), and allowing multiple service providers to concurrently control (FlowVisor) and configure (SNMPVisor) the underlying infrastructure.

2.1 OpenFlow

OpenFlow [7] is a feature added to switches, routers, access points (APs) and base-stations, allowing these datapath devices to be controlled through an external, standardized API. OpenFlow exploits the fact that almost all datapath devices already contain a flow-table (originally put there to hold firewall ACLs), although current switches and routers don't have a common external interface. In OpenFlow Wireless, we add OpenFlow to WiFi APs and WiMAX base-stations as well by modifying their software; and in principle the same thing could be done for LTE and other cellular technologies.

In OpenFlow - and therefore in OpenFlow Wireless - the network datapath is controlled by one or more remote controllers that run on a PC. The controller manages the flow-table in all the datapath elements and gets to decide how packets are routed in the network. In this manner, the datapath and its control are separated, and the controller has complete control over the operation of the datapath. The controller can define the granularity of a flow. For example a flow can consist of a single TCP session or any combination of packet headers (Layer 1-4) allowing aggregation.[2]

2.2 Network OS: NOX

In our network we use a freely available open-source controller NOX [5, 10]; but in principle any controller is possible, so long as it speaks the OpenFlow protocol. NOX provides network-wide visibility of the current topology, link-state and flow-state, and all other network events. As a network OS, NOX hosts applications or plug-ins that can observe and control the network state—for example, to implement a new routing protocol, or in our case to implement a new mobility manager. The mobility manager can choose to be made aware of every new application flow in the network, and can pick the route it takes. When the user moves, the mobility manager is notified, and can decide to re-route the flow. Because OpenFlow is independent of the physical layer (i.e., whether the wireless termination point is running WiFi or WiMAX), vertical handoff between different radio networks is transparent and simple.

The openness of the controller makes it is easy to add or change the functionality of the network. For example, a researcher can create a new mobility manager (e.g., one that does faster or lossless handoff) by simply modifying an existing one. In our prototype deployment (§3.3), we have already seen this happen many times, as researchers and students exchange code and build on each others work. In this way, we believe rapid innovation is possible. Further, by separating the datapath and its control, we can reap many benefits of centralized control (see below, *A Trend Towards Centralized Control of WiFi Networks*). Anecdotally, we have found network administrators receptive to a centrally managed network that is easily monitored.

Taken to the extreme, an application could be an entire mobility service, akin to the cellular service we buy from companies like AT&T, Vodafone, Orange, etc. An application can be written to implement AAA, billing, routing, directory services and so on... all running as a program on an OpenFlow controller. And because the controller is simply a program running on a server, it can be placed anywhere in the network - even in a remote data center. In the short term, we expect applications and experiments to be much simpler; but, in the long-term, an open network can grow to support a wealth of new services.

2.3 Hosting Multiple Simultaneous Experiments

Although we've explained how we can run a new experimental service in the OpenFlow Wireless network, it still begs the question of how we can have *multiple* competing services running at the same time in the same network. And how one

[2]More information about OpenFlow can be found at `http://OpenFlow.org`, including reference systems, specifications, and a list of commercial switches supporting the OpenFlow protocol.

A Trend Towards Centralized Control of WiFi Networks

Independently, campus WiFi networks are rapidly becoming more centralized. A central controller manages many wireless access points across a campus, allocating non-overlapping channels, setting power levels to reduce interference, and authenticating users in a single consistent way. Mobile users can roam freely across a campus without changing IP address, and without dropping TCP connections. Such a network is "software-defined", with the entire network operation determined by code running in the controller. When new features are needed in the networks (e.g. new routing algorithms, new authentication policies, etc.) the manufacturers simply update the central controllers. This trend is typified by a range of products from Cisco, Aruba, Meraki, Meru [2, 1, 9, 8] and a set of standards [6]; together they show a trend towards central control, and show it is viable (although commercial controllers are proprietary and closed, and are designed to work with specific products in a particular market segment).

Our experience so far is that network administrators like centralized control of their wireless network: it provides a single point of management, network configuration (e.g., access control) is more likely to stay consistent, and the network is easier to upgrade. OpenFlow Wireless provides this in an open way, building on reliable hardware, presenting (probably the first) practical way to open up the wireless network we all use everyday. We contend the synergy between administrators' trend towards centralized control and researchers' desire to work with production systems provides a powerful "carrot" for deploying OpenFlow Wireless as campus networks.

service could allow its users to roam freely across multiple physical networks. The trick here is to *slice* or virtualize the network, allowing multiple controllers to co-exist, each controlling a different slice of the network. A slice may consist of one user or many users; one network or many networks; one subset of traffic or all traffic. So, we use the *FlowVisor*: an open-source application created specifically to slice OpenFlow networks.

FlowVisor slices the network by delegating control of different flows to different controllers. As shown in Figure 2, FlowVisor is an additional layer added between the datapath and controllers. Because the FlowVisor speaks the OpenFlow protocol to the datapath, the datapath believe they are controlled by a single controller (the FlowVisor); and because the FlowVisor speaks OpenFlow to the controllers above, the controllers think they each control their own private network of switches (meaning a virtual network); i.e. FlowVisor is a transparent proxy for OpenFlow. The trick is to correctly isolate the flows according to a policy, and hence create one slice per experiment with its own private "flowspace" (a range of header values). FlowVisor works by deciding which OpenFlow messages belong to each slice, and pass them to the controller for that slice. If, for example, Controller A is responsible for all of Alice's traffic, then the FlowVisor passes all control messages relevant to Alice to Controller A. Therefore, FlowVisor separates slices according to a policy, defined by the network manager, by providing a strict communication isolation between slices.

A direct consequence of slicing the network is that we can safely run experiments in a production network. The FlowVisor allocates flowspace by default to the production network, which can be routed using legacy protocols. Each experiment is assigned its own slice, defined by the flowspace and topology, and implemented by the FlowVisor. Because real users are already connected to the production network, it makes opt-in relatively simple. If the network is sufficiently large, then experiments can be run at the same scale as, say, a campus wireless network. Or could even be run over multiple networks on multiple campuses.

Slicing/virtualization also allows "versioning" in the production network, where new features can gradually be incorporated into the production slice. Different slices can be dedicated to different versions, some more stable than others, as new features are carefully rolled out in stages. In this way, new features can be deployed and tested quickly, then gradually made available network-wide, and even shared with the owners of other campus networks. Such an ecosystem allows for the survival of the fittest, bringing the best to users. Also, legacy clients can be supported on a separate legacy slice, and the network can now evolve without being held-back by backward compatibility.

Finally, slicing allows delegation. Network administrators can cascade FlowVisors to further delegate (or slice) the flow space allocated to them. Repeated delegation makes sense in networks with a hierarchy of control; for example, in our network the campus network manager delegates a slice of the network for research experiments in our group, which we in turn slice (using another FlowVisor) among different experiments.

2.4 Configuring the Datapath

While OpenFlow provides a means to control the OpenFlow Wireless datapath, it doesn't provide a way to configure the datapath elements: e.g. to set power levels, allocate channels, enable and disable interfaces. This job is normally left to a command line interface, SNMP or NetConf. Although simple in principle, configuration is tricky in a sliced network, as we want to configure each slice independently. For example, we might wish to disable a certain network interface in one slice, without disabling the same physical interface shared by another slice.

We slice datapath configuration using a "SNMPVisor", that runs alongside the FlowVisor, to allow an experimenter to configure his slice. FlowVisor slices the datapath, and SNMPVisor slices the configuration by watching SNMP control messages, and sending them (and possibly modifying them) to the correct datapath element. Similar to FlowVisor, SNMPVisor acts as a transparent SNMP proxy between the datapath and controllers, providing the same features of versioning and delegation.

But sometimes it's hard or impossible to slice the configuration: For example, setting power levels for different slices on a WiFi AP. If slices share a channel, then we want to set different transmit power levels for the flows in each slice - something not possible on existing APs. We follow the general mantra of slicing where we can, and exposing non-sliceable configuration parameters to the user via feedback and error messages.

3. AN OPENFLOW WIRELESS DEPLOYMENT

To gain some early experience with OpenFlow Wireless (a.k.a. OpenRoads [14]), we built a prototype – based on our blueprint – and deployed on our campus [15]. Along with the details of our implementation and deployment, we also describe an often overlooked but crucial component, our measurement infrastructure. We implemented OpenFlow Wireless on top of NOX OpenFlow controller, through which we control the switches, APs and base-stations; and we slice the network using FlowVisor. We also extended SNMP into NOX to let us control power, frequency, data rate, SSID, etc., and to capture wireless events (like when hosts associate with an AP). All of these tools are freely available under open-source licenses [11], and we encourage the community to help us improve them over time.

3.1 Datapath elements: Access Points, Base-stations, and Switches

Our initial deployment consists of 30 WiFi APs, 2 WiMAX base-station and 5 Gigabit Ethernet switches in our wiring closets.

WiFi: Our WiFi APs, constructed with embedded computers running Linux, have two radios and cost about $200. We have also made available a lower-cost OpenWRT-based AP.

WiMAX base-station: We added OpenFlow to a WiMAX base-station, and placed it in our network using an FCC research spectrum license. The base-station essentially operates as a WiMAX AP running OpenFlow.

Switches: We use Ethernet switches in our wiring closets: NEC IP8800 24/48-port GE and HP 5406zl chassis GE switches, with firmware upgrades to support OpenFlow.

For future work, we plan to (and hope the community will also) experiment with more exotic hardware. For example, we could attach experimental programmable radios to our deployment, e.g., GNURadio [3] and WARP [13], add OpenFlow interface to them, and then exploit their extra programmability through SNMP or NetConf.

3.2 Logging and Measurement

To better ensure that the results of our experiments are accurate and repeatable, we implement a comprehensive measurement component as part of the OpenFlow Wireless deployment. With NOX's event logs, we record all network events including changes in topology, link-state, flow-state, etc. We have created a number of visualization, monitoring tools and GUIs running on top of NOX to help users instrument the state of their slice, and draw temporal correlations between network events. We believe that the completeness of the measurement infrastructure will further ease innovation in open wireless networks, exposing important problems, trends and interactions in the network.

3.3 Early Mobility Experiments

As a first foray into creating experiments on OpenFlow Wireless, we charged students in a 12 week project-based class to design and deploy their own novel mobility manager, then deploy it into the network. Some interesting designs resulted. Due to OpenFlow Wireless' design, all mobility managers were immediately able to do handover between WiFi and WiMAX, resulting in insights about handovers in such a heterogeneous environment. And another group used network state information from NOX to predict which channel they should use during a handover, to minimize the hunt time. In each project, the students demonstrated the manager working in our actual production network, running simultaneously in their own slice and evaluated them.

Examples of these mobility managers are as follows.

1. One group designed a mobility manager (Hoolock) to perform lossless handoff that receives packets in-order. The handover exploits the fact that if a device can communicate through different wireless technologies, it must also have multiple radios.

   We will illustrate the working of Hoolock using an example. Imagine a host handing over from AP α to AP β. Having two interfaces, the host associates with AP β with its second interface. The routing in the network is then updated. However, delay along the route before and after rerouting can be different, packets could be in transit along both routes for a while. Exploiting the fact that packets entering each interface on the host are in order, we buffer packets in the interface connected to AP β while waiting for packets from AP α to flush. After some time, the host will dissociate with AP α and switch completely to AP β, to complete the handover.

2. Another group created a "high-reliability" service by duplicating traffic across n distinct paths, i.e., n-casting, so that every client received multiple copies of each packet over different paths and radios. This can be viewed as a generalized variant of macro-diversity described in WiMAX.

   A typical n-cast handover between AP α and AP β occurs as follows. The mobile host starts with a single connection to AP α. While on the move, the host uses its idle interface to scan for and associate with AP β. Once associated, the controller begins n-casting or bicasting to both AP α and β. The bicasting continues until the mobile device dissociates from AP α and sends a notification to the controller to resume unicast.

   During n-cast, the host is likely to receive multiple copies of the same packet. Also, differences in path latencies and loss rates could cause reordering among the two packet streams. To mitigate the effect of duplicates and out-of-order packets on TCP, we employ a custom client-side Netfilter module on the receiving network stack, to perform some level of reordering. It buffers a small amount of incoming out-of-order packets from both interfaces to remove duplicates and reordering.

The above-mentioned mobility managers exploit devices with multiple interfaces. The control software is written in such a way that this is assumed. Through virtualization in OpenFlow Wireless, we can ensure that all devices controlled by these slices are indeed having multiple interfaces. Creating and testing this in conventional wireless networks would be difficult (if not impossible).

We evaluate these mobility managers against conventional hard handover, in terms of packet loss during handover and TCP throughput. The evaluation network setup is simple: two WiFi APs and a WiMAX base station are connected to a single OpenFlow switch. A server is also connected to the OpenFlow switch to generate traffic for the experiment.

Table 1: Statistics of Number of Packet Losses in Handover

Handover Scheme (holding time)		Avg	Std Dev	Min	Max
Hard		37.72	22.10	1	98
Hoolock (1 sec)		7.1	12.1	0	39
Hoolock (2 sec)		0.034	0.18	0	1
Bicasting (1 sec)		0	0	0	0
Bicasting (2 sec)		0	0	0	0
Vertical (1 sec)	to WiMAX	161.25	32.5	67	195
	to WiFi	6.1	9.43	0	37
Vertical (2 sec)	to WiMAX	87.2	50.11	0	135
	to WiFi	1.3	4.33	0	19
Vertical (4 sec)	to WiMAX	39.9	19.5	0	80
	to WiFi	0.0	0.0	0	0

In the cases of handover experiments, a mobile client moves between two WiFi APs. In the cases of vertical handover experiment, a mobile having a WiFi and a WiMAX interface moves between a WiFi AP and the WiMAX base station.

3.3.1 *Packet Loss during Handovers*

In each experiment, a client alternated between two WiFi APs (or between WiMAX base station and WiFi APs in vertical handover) every ten second for twenty times. We send ICMP requests from the server to the client and measure the number of packet losses during handover. The interval between ICMP messages is 20 ms.

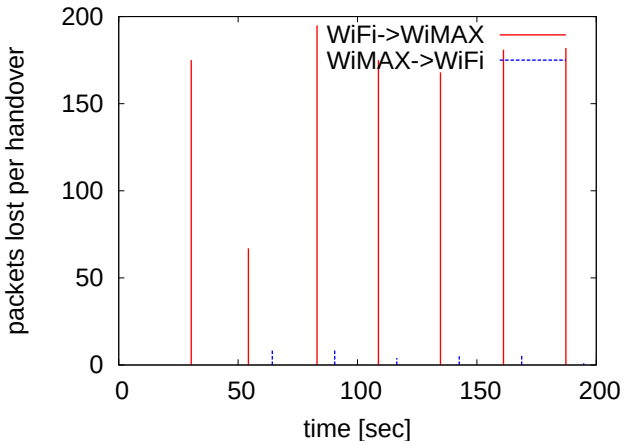
Tab. 1 shows the statistics of the packet losses. Hard handover loses the largest number of packets. Losses in hard handover occurs in spikes corresponding to the handover timings. In the case of Bicasting handover with 1 second holding time, we observed no packet loss in all twenty handovers. For Hoolock handover with holding time 2 seconds, we observed one packet loss in one out of twenty handovers. With Hoolock, increasing holding time reduces packet loss.

Fig. 3 shows the performance of vertical handover with holding times 1, 2, and 4 seconds. With vertical handover, two handover types—WiFi to WiMAX and WiMAX to WiFi—show very different results, i.e., handover from WiMAX to WiFi creates much smaller number of packet losses than that from WiFi to WiMAX. This is because network entry to WiMAX takes a long time, i.e., several seconds. If we release WiFi connection before the WiMAX connection is established, then packets will be dropped. Increasing holding time reduces packet loss.

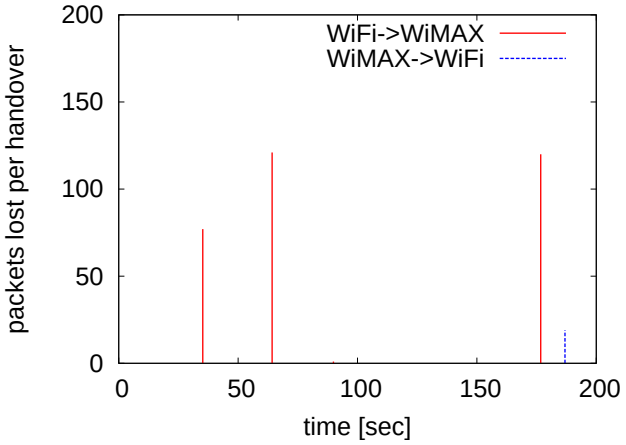
3.3.2 *TCP Throughout during Handovers*

We also measured the TCP throughput during handovers using iperf. Here the mobile host is the receiver of TCP data transfer. During the 300 seconds of data transfer, the mobile host switches WiFi APs every 60 seconds. Tab. 2 shows the aggregate results of the experiment. Fig. 4 show the evolution of good-put for hard handover, Hoolock handover and bicasting handover (both with holding time 2 seconds).

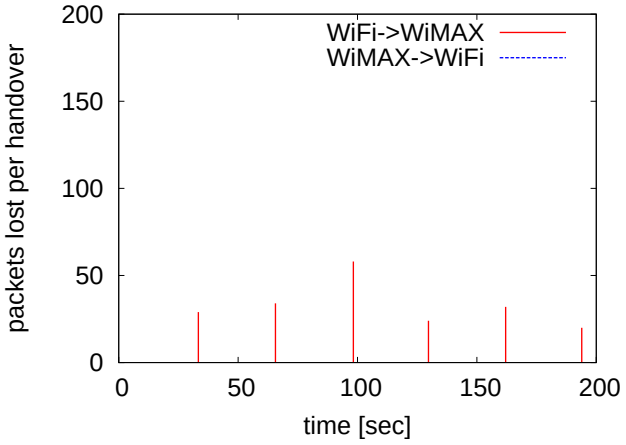
Hoolock and Bicast both improve on the simple hard handover scheme. In hard handover, we observe a long period of zero good-put. During the handover, large amounts of packet loss causes the sender to throttle its window size. On the other hand, Hoolock employs two interfaces to achieve



(a) 1 sec holding time

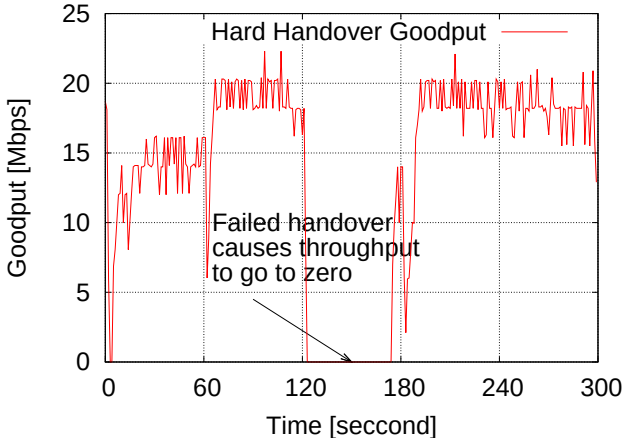


(b) 2 sec holding time

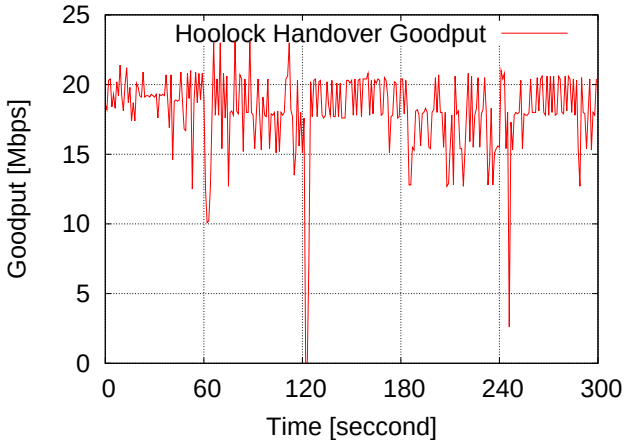


(c) 4 sec holding time

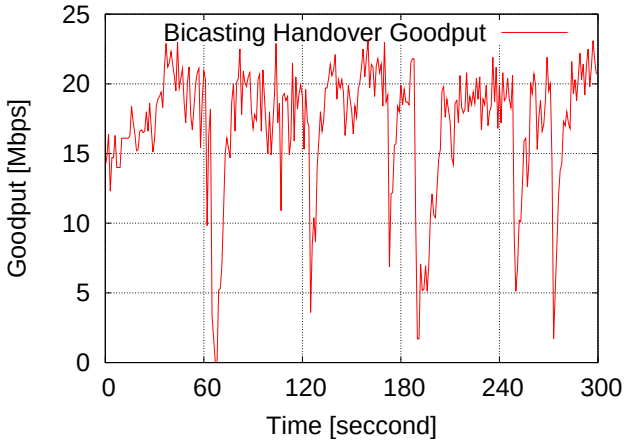
Figure 3: Packet losses in Vertical handover



(a) Hard handover



(b) Hoolock handover (holding time: 2 sec)



(c) Bicasting handover (holding time: 2 sec)

Figure 4: TCP Throughput with Handover

Table 2: TCP Throughput (unit:Mbps)

Handover Scheme	Avg	Std Dev	Min	Max
Hard	13.7	7.19	0	22.3
Hoolock (2 sec)	18.0	2.87	0	23.1
Bicasting (2 sec)	17.1	4.58	0	23.1

nearly-zero packet loss during handover and hence a much more stable good-put. The good-put of bicast is less stable for two reasons. First, the OpenFlow switch performing bicast is a software-based switch. Since bicast requires replicating packets to two output interfaces, the relatively slow software switch incurs much delay on the packet delivery. Second, the receiver maintains a buffer of out-of-order packets, which is flushed periodically. This period determines the delay and mis-sequencing of packets during handover. A small period would cause more frequent packet reordering, while a large period would incur large delay. The combined effects of delay and packet reordering are evident in the unstable good-put shortly after handovers as TCP attempts to converge to a new equilibrium. We note that choosing a suitable operating point for bicast is an interesting candidate for future work.

The point here is not that the mobility managers were radically new; it is that each one was written by non-experts in less than 4 weeks by building on top of a growing base of open-source code and a steadily improving platform. While we have many things to improve still - this is just a first prototype - we were surprised to find that each mobility manager could be written in approximately 200 lines of C++. We take these experiences as preliminary validation that a system like OpenFlow Wireless will indeed be useful to the research community by easing the innovation process in wireless networks. At the time of writing, OpenFlow Wireless has been deployed [15] and used as our production wireless network for over a year. Moving forward we expect OpenFlow Wireless to be an important part of our production network and our research alike, opening up new possibilities and enabling innovation.

4. CONCLUSION

The architecture of wireless networks is going to change significantly in the coming years, with a slow convergence of the cellular and WiFi networks. Without change, the industry will stay closed and based on proprietary equipment. Our role as the research community, is to help open up the infrastructure - to allow multiple ideas to co-exist in the same physical network - and therefore allow innovation to happen more freely and more quickly. Opening a closed infrastructure might seem like a naive pipedream; but recall the change that Linux brought to the computer industry by a dedicated community of open-source developers. We believe the best place to start opening the wireless infrastructure is on our own college campuses, by replacing our wireless networks with a more open (and backwardly compatible) alternative, which supports virtualization. We call this new network "OpenFlow Wireless".

OpenFlow Wireless builds right on top of OpenFlow (which is, itself, making progress in wireline networks). The main technical additions are the ability to slice the network using the FlowVisor (to slice or virtualize the datapath) and SNMPVisor (to slice or virtualize configuration); and the ability

for users to opt-in to experiments. As a whole, OpenFlow Wireless forms a complete production network that can be virtualized – by the existing network administrator – to create isolated slices for new experiments or new versions of features.

The blueprint we present here is a starting point. At the time of writing, we are starting to see adoption of the platform while we continue to improve and expand. We are in midst of deploying our WiMAX service for our local community, and looking at the possibility of including our work in GENI's open programmable WiMAX project. We will continue to explore how we can extend the platform to eliminate the boundary between wireless LAN and wireless WAN and enable seamless handover in the future mobile network.

More importantly, we hope our platform and this paper serve as a "call to arms" to the networking research community to come together and adopt, build, deploy and use the OpenFlow Wireless mobile wireless network architecture, building the system using existing access points, stripped down cellular base stations, wireline switches, and placed under the control of an open system that allows multiple isolated experiments to run concurrently in the network. We propose that the resulting network be widely deployed as our campus production wireless network. Done right, we believe that the community can share code, build on each other's work, and eventually create a common infrastructure in which researchers can safely try out new ideas, and network administrators can pick the best ideas to deploy in their production network and thus showcase "the future" on campuses before it is realized broadly.

Acknowledgment

The authors would like to thank Ippei Akiyoshi for his kind assistance in supporting the WiMAX base-station used. This work is supported in part by the NSF POMI (Programmable Open Mobile Internet) 2020 Expedition Grant 0832820, Stanford Clean Slate Program, Google, Xilinx, Cisco, NEC, DT, DoCoMo and the Mr. and Mrs. Chun Chiu Stanford Graduate Fellowship.

5. REFERENCES

[1] Aruba Networks. `http://www.arubanetworks.com`.

[2] Mobility - Cisco Systems. `http://www.cisco.com/en/US/netsol/ns175/networking_solutions_solution_segment_home.html`.

[3] GNU Radio. `gnuradio.org`.

[4] Official google blog: Where's my gphone? `http://googleblog.blogspot.com/2007/11/wheres-my-gphone.html`.

[5] 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.

[6] Internet-Drafts Database Interface. `http://ftp.ietf.org/drafts/draft-ohara-capwap-lwapp/`.

[7] 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.

[8] Enterprise Wireless LAN Networks: Indoor and Outdoor Wireless Networks By Meraki. `www.Meraki.com`.

[9] Meru Networks. `http://www.merunetworks.com/`.

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

[11] OpenFlow Wireless. `http://www.openflow.org/wk/index.php/OpenRoads`.

[12] Rob Sherwood, Michael Chan, Adam Covington, Glen Gibb, Mario Flajslik, Nikhil Handigol, Te-Yuan Huang, Peyman Kazemian, Masayoshi Kobayashi, Jad Naous, Srinivasan Seetharaman, David Underhill, Tatsuya Yabe, Kok-Kiong Yap, Yiannis Yiakoumis, Hongyi Zeng, Guido Appenzeller, Ramesh Johari, Nick McKeown, and Guru Parulkar. Carving research slices out of your production networks with OpenFlow. In *Proceedings of ACM SIGCOMM (Demo)*, Barcelona, Spain, August 2009.

[13] Rice University WARP - Wireless Open-Access Research Platform. `warp.rice.edu`.

[14] K.-K. Yap, M. Kobayashi, R. Sherwood, T.-Y. Huang, M. Chan, N. Handigol, and N. McKeown. Openroads: empowering research in mobile networks. *SIGCOMM Comput. Commun. Rev.*, 40(1):125–126, 2010.

[15] K.-K. Yap, M. Kobayashi, D. Underhill, S. Seetharaman, P. Kazemian, and N. McKeown. The Stanford OpenRoads Deployment. In *WiNTECH '09: Proceedings of the fourth ACM international workshop on Wireless network testbeds, experimental evaluation and characterization*. ACM, 2009.