Hoolock : Seemless Mobility in OpenFlow-enabled Wireless Access Networks

Nikhil Handigol, Wei Wei
Stanford University

I. Introduction

Introduction!

Deployment of Voice-over IP (VoIP) and other real-time streaming applications has been somewhat limited in wireless LANs today, partially because of the high handoff latencies and severe transmission loss experienced by mobile users. Our goal in this work is to eliminate handoff latency and to minimize the data loss during switch by exploiting the potential of multiple radios in Wi-Fi devices. In our work, a handover scheme called Hoolock has been developed. It is essentially a NOX-based mobility manager that realizes seamless mobile handoff over Open-Flow enabled Wi-Fi networks.

II. System Design

The mobile terminal in this work is a single device with two physically separate Wi-Fi radio interfaces. As Figure 1 shows, when a mobile moves away from AP-1 towards AP-2, the quality of wireless link between client and AP-1 will degrade and the quality of link between client and AP-2 will get improved.

//figure 1 here

Traditional mobile terminals with single radio interface suffer significant degradation in performance during hard switch. However, the configuration with two radio interfaces provides the possibility of seamless handover between APs.

The four-stage protocol of Hoolock scheme is shown in Figure 2.

//figure 2 here

1. The client is using one radio for transmission, while using another radio monitoring the quality of the available wireless links (Signal to Noise Ratio in our work) to the surrounding APs. When the link quality degrades below a certain threshold, the client issues a REQ$_Switchtothe NOXcontroller.$

2. The NOX controller gets all the routing information from the flow-in events and keeps tracks of all the ¡flow route¿ pairs. After receiving REQ$_Switch, the NOXinstructstheclienttothenextavailableAPwithACK_Switch.$

3. The client only uses the current active radio interface to listen to the channel and passively collects the incoming packets. At the same time, the client sends a REQ$_Breaksignalandanewflow-ineventviathesecondradiointerface.NOXgetsthenewr$
$FlowSwitch, andthenmodifytheflowentrytableinthecorrespondingOpen-flowSwitches.$

4. The NOX waits for a maximum flush-time (max RTT/2), let all the floating packets flushed into the client, and then issues an ACK$_Breaksignaltotheclient.Theclientthenusethepreviousradiotoscantheavailablechannelsandthenusethesecondradiof$

III. Protocol

Protocol

IV. Implementation

// figure 3 here!

s it is shown in Figure 3, the Hoolock scheme realizing the four-step handover protocol consists of three parts, the NOX module, client program and bonding driver.

The bonding driver takes care of the management of operations with two physically independent radio interfaces, which has two functions provided to the client program.

// make (): perform the connection action between the active radio to a specified AP.

// break (): break the connection between radio and AP.

// Here please add the challenges when dealing with bonding driver.

The client program acts an intermediate role between the bonding driver and NOX controller. It communicates with bonding driver via the ioctl() function and interacts with NOX through the messenger according to the four-way protocol mentioned in part II. The client program provides two functions to the NOX module.

// init$_{m}ake() : sendouttheconnectionrequeststotheNOXcontroller.$

// init$_{b}reak() : sendoutthebreakrequeststotheNOXcontroller.$

The NOX controller has the knowledge about the entire network. From every flow-in event, the NOX collects the routing information and keeps the records of ¡flow route¿ pairs. For the new requests issued from the mobile, NOX will perform a series of actions to update the routing scheme. It firstly traverses the routing information and calculates for a reverse route, and compares the reverse route with the previous route using the ¡flow route¿ lookup table to derive a common partial route. After that, the NOX controller finds the common root Open-Flow switch and updates the flow entry table in the common root switch. It has three functions for the client program.

// getAP (): need to check with the actual function names with these 3 functions!

// switchRoutes ():

// some more

V. Performance Evaluation

¡¡¡¡¡¡ .mine The topologies of the networks in simulation are shown in Figure 4. It consists of three Open-Flow switches: rootopenflow, openflow0, openflow1 and two VMs: wired VM and mobile VM.

The simulation uses wirefilter to conduct the handover between mobile VM and two Open-Flow switches. The mobile VM initially connects to wired VM via openflow0 and rootopenflow, then it switches from openflow0 to openflow1. The handover can also work reversely, switch from openflow1 to openflow0.

In the performance evaluation, we conducted the hard handover scheme (shown in Figure 4.1) and our seamless handover scheme, the Hoolock scheme (shown in Figure 4.2) over the network shown in Figure 3. The comparative test results with lossy links and lossless links are summarized in Table 1.

// figure 4.1 and 4.2 here, for hard handover and Hoolock handover

With the lossless links, it is clear that the hard handover scheme has a significant data loss, approximately XXX during the switch. And with the Hoolock handover scheme, the loss is just..Obviously, there is a great improvement in data loss.

Then using the lossy links with loss rate of , the average data loss of switch over lossy links with Hoolock scheme is.. It is the same with the inherent loss rate in the lossy links. In this case, the Hoolock scheme is a truly seamless handover scheme. From the applications perspective, it is like a continuous transmission without handover. =======

¿¿¿¿¿¿¿ .r34

VI. Conclusions and Future Work

From the simulation results, we conclude that the Hoolock scheme is demonstrated to be a seamless handover scheme. It has a superior performance over hard handover scheme in terms of data loss during switch. Another advantage of Hoolock is it doesnt need extra channel scanning time. Thats because Hoolock scans the available channels while maintaining an active connection, so when a handover is requested, it can switch immediately.

Given the very limited time frame this quarter, we didnt have the chance to make deployment, so all these improvements derived from the simulation result might cast doubts. In real situation, the wireless links are influenced by various factors, like multipath fading, Doppler shift and shadowing effects. How these issues affect the network performance will be further investigated.

The NOX controller, client program and bonding driver are supposed to be running on actual devices. It will be very interesting to see how these will affect the overall computational performance as a tradeoff for the improved performance.

Moreover, the network topologies applied in our simulation are relatively fundamental. The generality should be proved using more complex networks with a number of different topologies. More through measurement metrics will be applied in further studies.

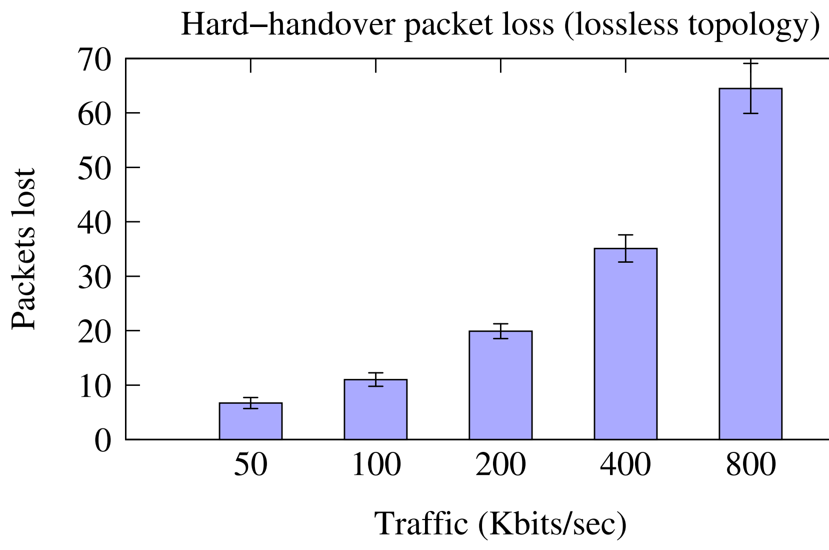

Fig. 1: Hard-handover packet loss in a lossless topology.

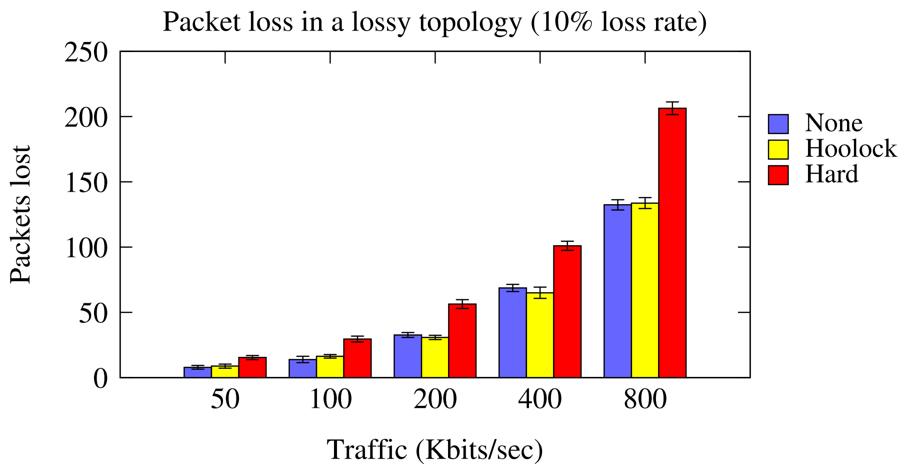

Fig. 2: Packet loss in a lossy topology.

The Open-Flow infrastructure exhibits it power in our work. As a flow-based routing architecture, it has many advantages over the rigid, traditional networks. The major difference is its capability of managing the flow entry tables in each switch. With the embedded event-trigger model, it is very convenient to reorganize the topology of the network and make highly dynamic flows. The handover application is just the tip of iceberg, the Open-Flow and its applications have a huge potential in defining the future Internet Infrastructure standards.

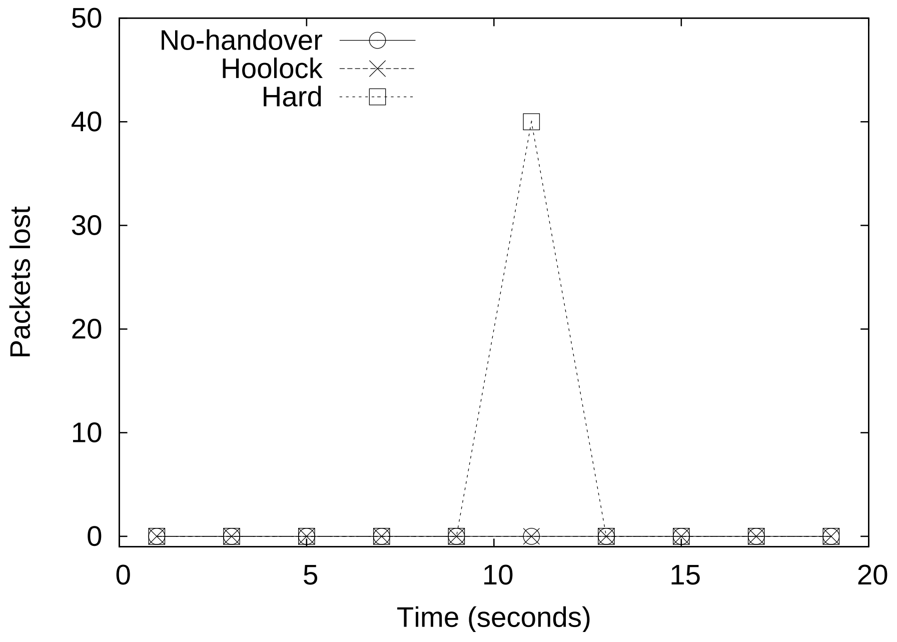

Fig. 3: Packet loss in a lossless topology.

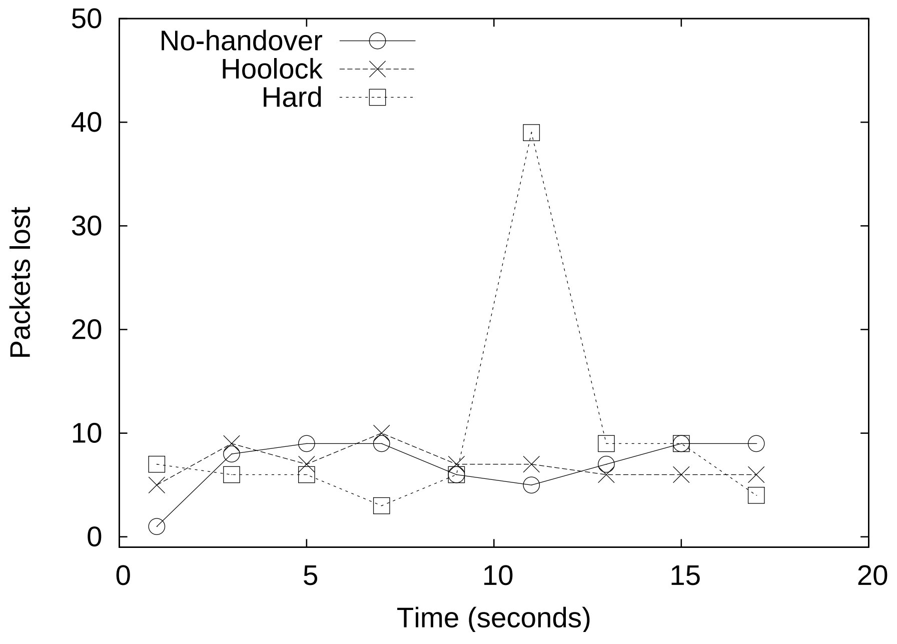

Fig. 4: Packet loss in a lossy topology.