

WiSwitcher: An Efficient Client for Managing Multiple APs

Domenico Giustiniano
Telefonica Research
Barcelona, Spain
domenico.giustiniano@tid.es

Eduard Goma
Telefonica Research
Barcelona, Spain
goma@tid.es

Alberto Lopez Toledo^{*}
Telefonica Research
Barcelona, Spain
alopez@tid.es

Pablo Rodriguez
Telefonica Research
Barcelona, Spain
pablorr@tid.es

ABSTRACT

There has been an increasing interest on designing a single-radio client for time-division access to multiple Access Points (APs) on different radio-channels. These works have focused mainly on different scheduling policies at the client-side to allocate the percentage of time to each AP. However the performance of these systems is limited by 1) the overhead to switch between APs on different radio-channels, 2) the jitter in the switching procedure, that modifies the expected percentage of time assigned by schedulers and 3) the packet losses caused by the switching.

In this paper, we introduce WiSwitcher, a client able to connect to multiple APs that i) reduces the cost of switching down to the hardware switching time and ii) increases the stability of the percentage of time assigned by schedulers, even if the station transmits in saturation mode. We implement WiSwitcher over commodity hardware and show that it achieves high aggregate throughput over the connecting APs and seamlessly transmits TCP traffic under controlled scenarios. Finally, we characterize the dependency between the switching frequency at the WiSwitcher client and the packet losses in off-the-shelf APs.

Categories and Subject Descriptors

C 2.1 [Computer System Organization]:
COMPUTER-COMMUNICATION NETWORKS *Network
Architecture and Design* Wireless communication

General Terms

Design, Experimentation, Measurement, Performance

^{*}Alberto Lopez Toledo is supported by the Institució Catalana de Recerca i Estudis Avançats (ICREA).

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.

PRESTO'09, August 21, 2009, Barcelona, Spain.

Copyright 2009 ACM 978-1-60558-446-1/09/08 ...\$5.00.

Keywords

MAC, TDMA, multi-channel, wireless network

1. INTRODUCTION

Wireless local area networks (WLANs) were traditionally envisioned with the goal of increasing the coverage range for connecting to the Internet. As a typical example, home wireless connection is nowadays a standard “de-facto” for residential Cable/ADSL subscriptions.

While the Cable/ADSL lines are generally low speed and under-utilized connections, wireless connectivity to the Access Point (AP) can achieve up to 20 times the speed of the Cable/ADSL lines. The density of these Cable/ADSL deployments with wireless connectivity tends to be high [1] and represents the bottleneck in the end-to-end communication [2]. Then, Cable/ADSL bandwidth aggregation via wireless connectivity is attractive and incurs in no extra infrastructure cost [3].

In this scenario, previous work [3, 4] has mainly focused on the definition of a time-division scheduler to assign the percentage of connection to each AP. On the other hand, little attention has been given by the literature to solutions with fine-grained timing. In fact, time-division approaches need precise time in scheduling to ensure that transmissions/receptions occur when expected.

There are two main factor of timing degradation, both related to the management of APs on different radio-channels. First, a MAC delay processing occurs while switching to an AP at a different radio-frequency, because of operations as sending probe messages, resetting the hardware, etc. This overhead has a negative impact on the throughput.

Second, the hardware queue must be drained before switching to a different frequency. This operation introduces unpredictable jitter in the timing procedure, which modifies the expected percentage of time assigned by the upper scheduler. Imprecise jitter can be resolved by long guard periods [5]. However, this would degrade the throughput further.

In this work, we present WiSwitcher, a single-radio wireless client that can connect to multi-frequency APs, and aggregate their available Cable/ADSL bandwidth. WiSwitcher increases the throughput observed by the client, thanks to a virtualized 802.11 MAC client that:

- reduces the AP switching cost down to a mere hardware-imposed switching delay.

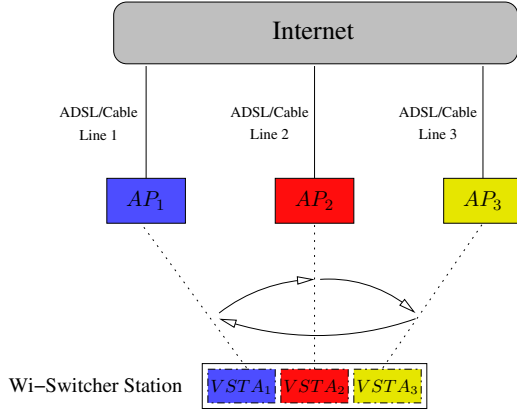


Figure 1: Topology

- increases the stability of the percentage of time assigned by the schedulers to each AP.

We design and implement WiSwitcher in commodity hardware and demonstrate the feasibility of the implementation in controlled scenarios. Finally, we also study the impact of packet losses on the performance and show that off-the-shelf APs add packet losses when switching.

The rest of this manuscript is organized as follows. Section 2 presents related work. Section 3 introduces WiSwitcher and Section 4 presents the implementation details. Section 5 validates the WiSwitcher implementation in a controlled environment and finally Section 6 gives the conclusions.

2. RELATED WORK

The idea of connecting to multiple APs through a single radio interface is shown in VirtualWiFi [4]. The authors rely on the Power Save (PS) mode feature of the WLAN standard to switch among different Wi-Fi nodes (in AP and/or Ad-hoc mode) in a time-division fashion. A client can inform the current Wi-Fi node that it is going into PS mode — so that it can buffer packets directed to it — and switch radio-frequency to other Wi-Fi nodes, only to come back to the original node before the PS period expires. Switching between networks is transparent to the applications, but at a high cost in time (30-600 msec). In fact, VirtualWiFi implements the code on top of the driver card and run a MAC instance for each network, with a scheduler that assigns more active time to the MAC instance with higher amount of data to send.

FatVAP [3] studies the problem of Cable/ADSL bandwidth aggregation via wireless connectivity. The authors introduce a scheduler to select the percentage of connection time on each AP to maximize the aggregate throughput. The solution leverages on the fact that the high speed wireless card needs to be connected on each AP for a short period of time in order to collect all the pending data. FatVAP has an average switching cost of 2.8 msec plus another 2.8 msec of standard deviation that is taken into account in the scheduler calculation. However, no study of the effect of jitter has given on the scheduler performance.

Finally, Juggler [8] focuses on the support for a seamless hand-off between WLAN APs and operates over a switching cost similar to FatVAP.

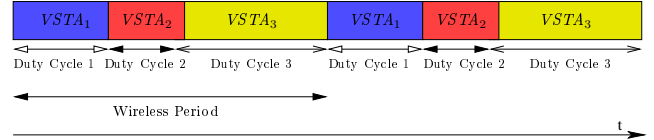


Figure 2: Relation between *duty cycle* and *wireless period*.

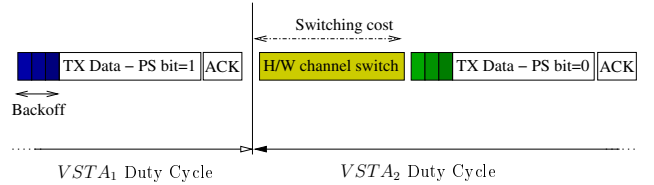


Figure 3: Procedure to switch the virtual station.

3. OVERVIEW

An example scenario with a WiSwitcher station is given in Fig. 1. In WiSwitcher, the wireless driver on top of the single radio card is *virtualized*, i.e., it appears as independent Virtual Stations ($VSTA_i$) associated to their respective Access Point AP_i . Each of these virtual clients connects to Internet via its AP backhaul, and independently of the AP radio-frequency. In the example in Fig. 1, there are 3 virtual clients $VSTA_1$, $VSTA_2$ and $VSTA_3$, each one connected to one AP.

WiSwitcher assigns the control of the card to a $VSTA_i$ for a given time, called *duty cycle* (see Fig 2). During this time, it transmits/receives frames over the AP backhaul while the other $VSTAs$ (and the corresponding APs) can only buffer packets.

WiSwitcher manages the multiple backhaul connections relying on the 802.11 PS mechanism. Particularly, referring to the example in Fig 3:

- During the reserved *duty cycle*, $VSTA_1$ transmits and receives data according to the 802.11 DCF protocol. The other $VSTAs$ are in PS mode, and hence they (and the corresponding APs) can only buffer packets.
- When the *duty cycle* expires, $VSTA_1$ sends a frame to inform AP_1 that is going to PS mode and waits for its MAC ACK. According to the 802.11 protocol, AP_i starts to buffer the packets directed to it.
- WiSwitcher assigns the control of the card to $VSTA_2$ and switches to the AP_2 radio-frequency.
- $VSTA_2$ sends a frame to announce that it can send/receive traffic and waits for its MAC ACK.

We denote *wireless period* as the sum of the *duty cycles*. The *wireless period* represents the amount of time to cycle through all the $VSTAs$.

4. IMPLEMENTATION

WiSwitcher has been implemented as a wireless client based on the MadWiFi driver 0.9.4 [6] and Click Router 1.6.0 [7]. WiSwitcher selects in a time-division fashion the

APs to connect to and does not require any modification to the APs.

In the implementation, we incur in a channel-switching cost — i.e. the time where WiSwitcher cannot transmit/receive any traffic — of 1.2 msecs for uplink traffic and 1.5 msec for downlink traffic. This cost is less than half of the one obtained in the time-division implementation given in [3, 8], thanks to key points discussed in the next section.

The bulk of the cost is caused by the hardware operation delay, which is in the order of 800 μ sec in our Atheros chipset-based cards. This cost is hardware dependent and in other chipset implementations is reduced to 200-500 μ sec [8, 9].

Key points

Let us consider the Fig 4. WiSwitcher implementation is based on four key points, below described.

First, WiSwitcher creates a MAC queue per $VSTA$. Since the Linux kernel does not implement the queues for virtual devices, we use the PS queues¹ as $VSTA$ MAC layer queues. We setup each of these queues to accept packets from the upper layer, up to a limit of 200 packets. Based on the MAC address, WiSwitcher copies the IP packets in the corresponding MAC layer queues but *only* the (single) $VSTA$ out of PS can copy the packets from the $VSTA$ MAC layer queue to the H/W queue for the subsequent transmission.

Second, WiSwitcher efficiently manages a H/W queue size equal to one (1) data packet. This feature is not supported in normal drivers, that present high performance drops in such a configuration. In order to by-pass this problem, WiSwitcher copies each packet that arrives from the IP layer to the tail of the $VSTA$ MAC layer queue. Then:

- if the hardware queue is empty (i.e. no data packet in the H/W queue) and the $VSTA$ is currently selected, the packet on top of the PS queue is copied immediately in the H/W queue and transmitted according to the 802.11 DCF protocol.
- if the hardware queue is empty (i.e. no data packet in the H/W queue) and the $VSTA$ is currently not selected, the packet on top of the PS queue is copied later in the H/W queue, when the $VSTA$ will be selected.
- if instead the hardware queue is not empty (i.e. one data packet in the H/W queue), the next packet will be copied in the hardware queue 1) if the *duty cycle* is not expired and 2) the packet in the H/W queue receives a MAC ACK or reaches the maximum MAC retry.

Third, since the Power Save mode simply relies on the Power Management bit in the MAC header, this bit is set equal to 1 or 0 according to the $VSTA$ PS state. Hence, instead of generating probe messages for sending just 1 bit of information, as normally done in 802.11 implementations, WiSwitcher uses regular data traffic buffered in the MAC $VSTA$ layer queue to switch the PS state. This feature is used in three procedures:

1. When the *duty cycle* expires on $VSTA_i$, WiSwitcher takes the packet on top of the currently active MAC

¹In MadWiFi, one PS queue is created for each virtual interface.

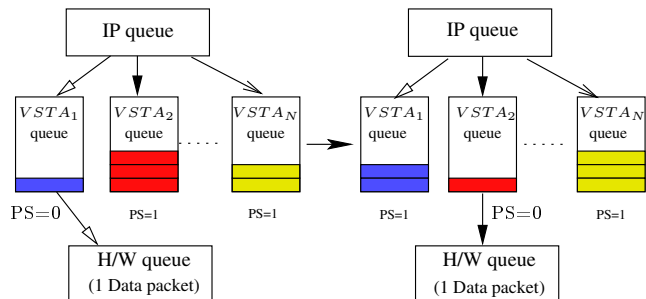


Figure 4: Queue Management in WiSwitcher.

$VSTA_i$ queue, changes its PS flag bit to one, flushes it in the H/W queue, and sends it to AP_i .

2. When the new $VSTA_j$ has been selected, WiSwitcher takes the packet on top of the new active MAC $VSTA_j$ queue, selects the flag PS bit set to zero, flushes it in the H/W queue, and sends it to AP_j ².
3. There are situations where the packet with flag PS bit set to zero is not acknowledged within the maximum MAC retransmission counter. As a recovery mechanism, WiSwitcher takes the next packet on top of the new active MAC $VSTA_j$ queue, sets again the flag PS bit to zero and sends it to AP_j . This procedure is repeated until the packet is successfully acknowledged or the *duty cycle* expired. If there are no more packets in the MAC $VSTA_j$ queue, WiSwitcher stops the recovery mechanism.

Fourth, the rate selection algorithm works independently for the different $VSTAs$. This allows to connect to APs with different quality. Anyway, in case of high traffic load and/or low wireless channel quality, the packet sent to switch to PS mode can delay the start of the connection to the next AP and increase the *wireless period*. Then, in order to minimize the effect on switching delay, this extra-time of transmission has subtracted from the next *duty cycle* assigned by the scheduler to the $VSTA$ ³. This guarantees that the wireless period does not fluctuate.

State machine management

In order to manage and keep the N $VSTAs$, the 802.11 state machine has been modified. Each operation within the state machine is scheduled in the WiSwitcher station according to the software kernel interrupts, at the granularity of 1 msec.

In the initialization phase, WiSwitcher creates N $VSTAs$, and each one of them starts to actively scan for APs (*Scan_Mode*). During the *Scan_Mode*, each $VSTA$ scans over

²Note that WiSwitcher still sends probe messages when the MAC layer queues are empty. In this case, for evaluating the switching cost, we compared the start of back-off time of the probe message with the completion time of the H/W switching procedure. For calculating the start of the back-off time we used a methodology similar to the one presented in [10].

³Note that the correct acknowledgment of the PS packet within the maximum MAC retry guarantees that the AP does not attempt to send packets to a WiSwitcher client while it is transmitting/receiving on another channel.

