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
alopezt@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).

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

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.

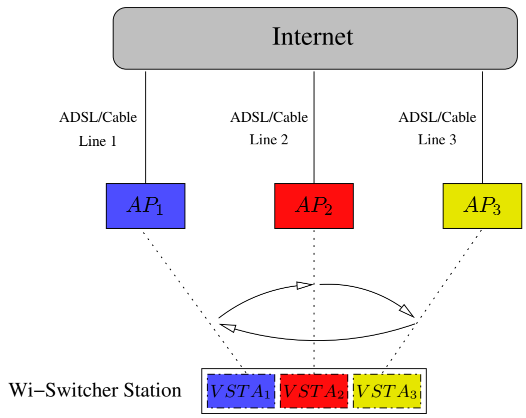


Figure 1: Topology

- 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 introduces 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.

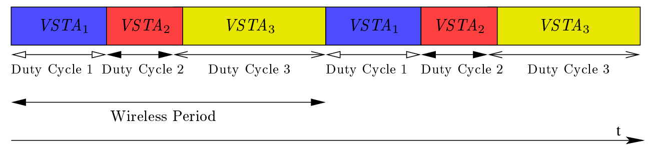


Figure 2: Relation between *duty cycle* and *wireless period*.

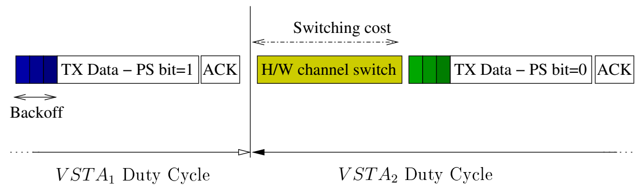


Figure 3: Procedure to switch the virtual station.

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 μsec in our Atheros chipset-based cards. This cost is hardware dependent and in other chipset implementations is reduced to 200-500 μsec [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 queues[1] 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 $VSTA_i$ 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* 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.

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

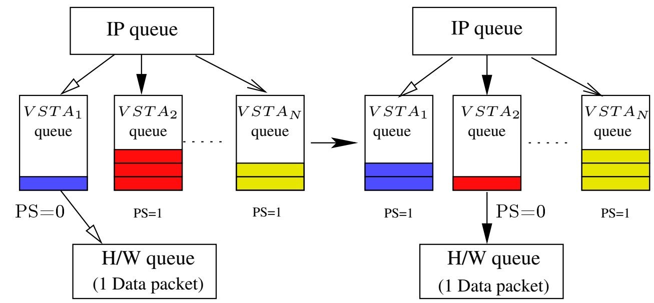


Figure 4: Queue Management in WiSwitcher.

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.

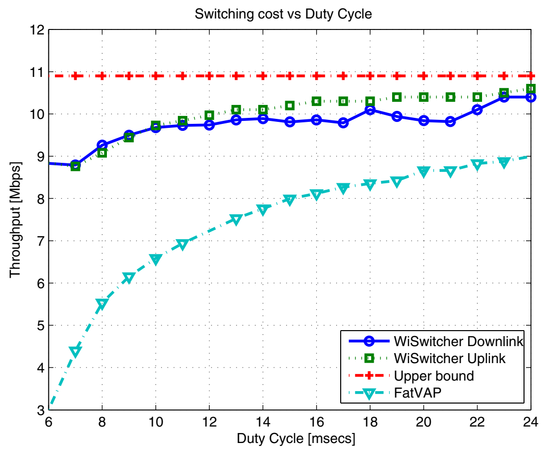
State machine management

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.

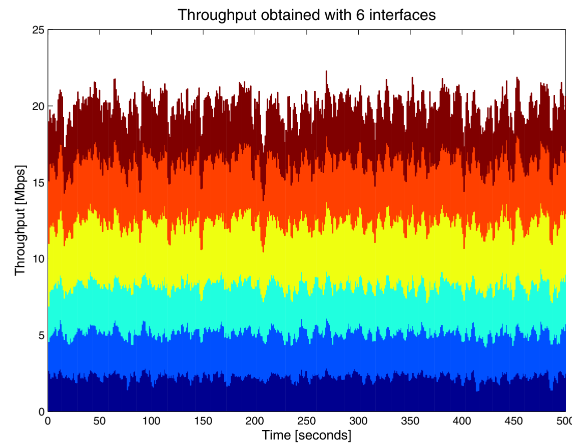
In the initialization phase, WiSwitcher creates N *VSTAs*, and each one of them starts to actively scan for APs (`Scan_Mode`). During the `Scan_Mode`, each *VSTA* scans over

[2]Note 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].

[3]Note 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.



(a) Impact of switching cost on throughput (50% connection to one AP).



(b) TCP aggregate throughput. Each color represents the throughput component given by each connection on different radio-frequency per second.

Figure 5: Duty Cycle Assessment.

a time of *wireless period*/N before switching to the next *VSTA*.

APs are dynamically selected based on their essid. In fact each 802.11 standard compliant AP of our network uses the same essid and APs are only differentiated based on their MAC address. When one *VSTA* is associated to the corresponding AP (`Run_Mode`), the driver schedules only these subset of interfaces, for a time equal to the assigned *duty cycle*. On the other hand, if the *duty cycle* of one virtual interface is forced to zero, the corresponding *VSTA* disassociates from its AP.

WiSwitcher also does a background scanning of the APs in range both for the *VSTAs* in `Run_Mode` and the *VSTAs* in `Scan_Mode`. This procedure can start only if there is no traffic pending on the current selected *VSTA* in `Run_Mode`. The scanning can be eventually stopped in case WiSwitcher needs to switch *VSTA*, and later resumed if again there is no traffic pending. This also implies a dynamic update of the available APs without affecting the foreground connectivity.

Statistics per channel

Wireless channel occupancy can differ according to the traffic load and transmission rate. Moreover, the occupancy is radio-frequency dependent and nearby frequencies have correlated statistics.

In order to efficiently manage the available spectrum and estimate the radio-frequency utilization, WiSwitcher estimate the utilization of each 802.11 radio-frequency. This metric is defined as the conditional probability that the channel is busy when the *VSTA* is not transmitting. The statistics are updated at both `Scan_Mode` and `Run_Mode` and rely on specific 802.11 baseband registers, wherein the NIC card updates both the busy and the total time with an accuracy of the 802.11 clock (i.e. 44 MHz).

Reverse-NAT

APs commonly use NAT to share a wired link and assign IP addresses from private blocks. Then, in order to guarantee transparency to higher layer, we implement a reverse-network address translation (NAT) module with two functions: i) assure that the packets leave the host with the correct source IP address (i.e. the one corresponding to the outgoing virtual interface) and ii) that the incoming packets are presented to the OS with the expected IP address. The implementation of this module uses the Click modular router [7] and it is similar to the one given in [3] and [8].

5. ASSESSMENT

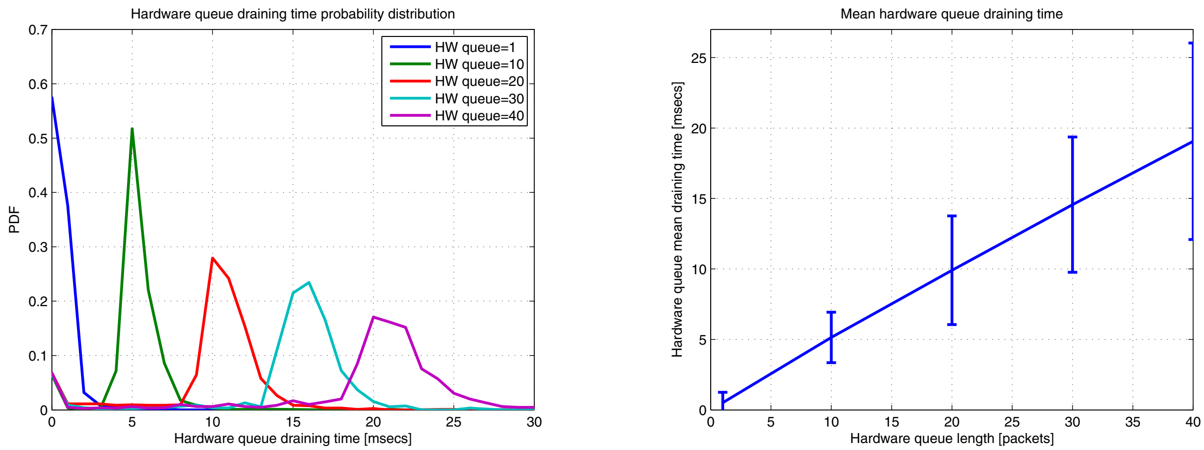
In this section we perform an experimental evaluation of the WiSwitcher implementation. The results show that:

- there is a low impact of the switching cost on the throughput, even for the case of a few milliseconds of connection time.
- the jitter caused by the draining of the hardware queue at each switching is minimized.
- there is a constant number of packet losses generated by off-the-shelf APs in each switching procedure.

Throughput

In this section we show the throughput performance of a WiSwitcher client in two different configurations, i) as function of the *duty cycle* and ii) in presence of a high number of APs.

First, we performed a set of tests with a WiSwitcher station connected to 2 servers at 100 Mbps Ethernet cable speed via 2 APs using a physical transmission rate of 54Mbps. In this controlled scenario, we fixed the percentage of connection to 50% on each of the two APs. In the testbed, we measured the TCP throughput over 1 AP connection as a function of the *duty cycle* and compared it to the expected maximum bandwidth — i.e. half of throughput that we get without switching (upper bound curve). The results are shown in Fig. 5(a). For each point of the plot, we ran five tests with a duration of 100 secs, with the *duty cycle* that sweeps from 6 ms to 24 ms.



(a) Hardware queue draining time probability distribution function.

(b) Mean draining times for different hardware queue lengths.

Figure 6: Impact of the hardware queue length in the duty cycle duration.

We observe that WiSwitcher can get high throughput performance both in uplink and downlink using very small connection time, with a slightly better performance of uplink for given *duty cycle*. This finding can be easily explained as follows: in the uplink traffic, a data packet is always on top of the *VSTA* MAC layer queue. Then this packet can be used to go and return from PS mode. Instead, in the downlink, the session may have TCP ACK sent with the delayed ACK option — i.e. it sends one TCP ACK every two TCP data — resulting in an empty virtual queue at the *VSTA* at the moment of switching. When the queue is empty, in order to switch to PS, the *VSTA* sends a probe message with NULL data, which represents an overhead for WiSwitcher. This has the effect of an increase the switching cost from 1.2 msec in uplink to 1.5 msec in downlink.

Fig. 5(a) also shows the performance of a FatVAP station. Whereas WiSwitcher starts at 85% of the expected throughput with 6ms of *duty cycle*, FatVAP is unable to get half of the expected maximum throughput. This is caused by the higher switching cost in the FatVAP implementation. The benefits of WiSwitcher reduce at higher *duty cycles*, with a 22% higher throughput respect to FatVAP at the *duty cycle* of 15 msecs. At this *duty cycle*, WiSwitcher achieves the 95% of the expected throughput. As a result of the tests, we conclude that the minimum *duty cycle* to get a stable throughput performance is 15ms.

We now evaluate the throughput of a WiSwitcher client connected to six APs at six different radio-frequencies, where each *VSTA* connects to the corresponding AP for a *duty cycle* of 15msec. Fig. 5(b) depicts with a different color band the throughput achieved on each of the 6 APs versus the time. The sum of the contribution is very close to the theoretical maximum of 21.7 Mbps that can be achieved connecting 100% of the time to only 1 AP. This throughput confirms that the switching cost can be effectively neglected at a *duty cycle* of 15msec.

Reducing Jitter

In this section we show that WiSwitcher allows for a transparent design of the scheduler by reducing the fluctuation of the *duty cycle* caused by the draining time of the H/W queue prior to each switching.

In stations with unmodified MadWiFi driver the hardware transmit queue is set to 40 data packets. In this configuration, when a station has to switch from one AP to another and it transmits at high speed, it might fill up the hardware queue of the wireless card. The draining of this queue before switching to the next AP might take a considerable amount of time — compared to the *duty cycle* duration— and depending on the i) wireless channel contention and ii) the *wireless period* size. As a result of this *draining time*, the real amount of time connected to a certain AP_i increases in each *duty cycle* — respect to the expected value given by the upper layer scheduler — and consequently does the *wireless period*. It follows that the draining procedure leads to an uncontrolled fluctuation of the percentages of connection initially set by the upper layer scheduler since these *duty cycles* will change independently.

Then, it is important to ensure that the *wireless period* and the *duty cycle* remain as stable as possible to guarantee a perfect match between the expected *duty cycles* given by an upper layer scheduler and the real values.

In order to assess the variation of the *duty cycle* depending on the cost of draining the hardware queue, we performed five tests for five different hardware queue length's: 1 packet (which is the configuration of WiSwitcher), 10, 20, 30, and 40 [4]. In the test, we used a *wireless period* and a *duty cycle* for the WiSwitcher station of 100ms and 50ms respectively. In order to fill the transmission hardware queue, the test was done in uplink. No queuing disciplines were set in the wired network in order to push the wireless network as much as possible with the highest throughput and making it to become the bottleneck.

Fig. 6(a) shows the probability distribution function (pdf) of the draining time for each hardware queue length. We observe that the mean value increases with the hardware

[4]To perform this test we needed to optimize the relation and communication between the different queues as described in Section 4.

queue length increases. From this figure we can conclude that the draining time increases with the H/W queue length. In addition, the draining time standard deviation increases with the hardware queue length.

In Fig. 6(b) we show the mean draining time respect to the hardware queue length with the corresponding error bars. The figure shows the linear increase of both the average and standard deviation in the draining time. We see that the most stable *duty cycle* can be obtained with the smallest hardware queue, which is the configuration used by WiSwitcher.

Packet losses at the AP

Packet loss assessment is fundamental in protocols as TCP, that treats losses as an indication of congestion and reduces its sending rate in their presence. While Cable/ADSL exhibits low packet loss rate [11], this is not guaranteed in the wireless path. In presence of good wireless channel quality[5], the main cost of packet losses in WiSwitcher is the switching procedure.

In order to characterize the effect of switching, we measured the TCP RTT, the amount of TCP packet retransmissions and the number of times that TCP detects a congestion signal. As a result of using SACK, congestion signals are mainly caused by fast retransmissions due to duplicated ACKs because its goal is to avoid retransmission timeouts. We then considered packet loss rate as the percentage of congestion signals triggered per acknowledged TCP packet.

Fig. 7 shows the packet loss rate versus the *duty cycle* duration and compares them to the evolution of the number of switching procedures per second. For the test, we considered a 50 % of time connected to the AP under analysis. Each value of the plot is the average over 5 tests of 300 secs each one, where we subtracted the average packet loss rate we found without switching.

As it can be seen in Fig. 7, initially, the packet loss rate decreases exponentially with the *duty cycle*, following the same shape of the number of switching per second (labeled as *switching frequency* in the figure). The implication of this result is that the higher is the switching frequency, the higher are the packets lost and in consequence, it is possible to assume a certain rate of packets loss per switching procedure.

After this initial exponential phase, the packet loss rate remains around 0.01% of increase respect to the no-switching situation. Particularly, the frequency switching introduces a low number of packet losses as long as the *duty cycle* is kept above or equal to 30 msecs (around $0.25 * 10^{-3}$ of the total transmitted TCP segments at 20 msecs).

Concluding, these results are caused by current APs implementations. In fact, they do not support fast switching rate and then, the high frequency switching causes TCP to perceive a higher packet loss rate, likely generated by both out-of-order packets and packet losses in the AP H/W and PS queues.

6. CONCLUSION AND FUTURE WORK

In this paper we have presented WiSwitcher, a wireless station that can connect to different APs in range on different radio-frequencies and aggregate their unused AP backhaul bandwidth. WiSwitcher achieves fine-grained timing at MAC/PHY level and allows stable performance even under saturated conditions. Future work aims at reducing the packet losses at the APs in open-source drivers and testing the performance of WiSwitcher with different schedulers.

[5] Negligible number of MAC frames that reach the maximum MAC retransmission counter.

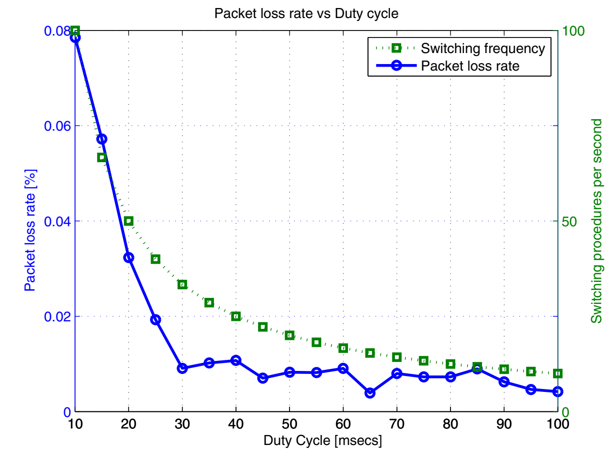


Figure 7: Packet Losses are generated at high switching rate in off-the-shelf APs.

7. REFERENCES

[1] D. Han, A. Agarwala, D. G. Andersen, M. Kaminsky, K. Papagiannaki, S. Seshan. "Mark-and-Sweep: Getting the "Inside" Scoop on Neighborhood Networks", IMC 2008.

[2] M. Siekkinen, D. Collange, G. Urvoy-Keller, and E. W. Biersack "Performance Limitations of ADSL Users: A Case Study", PAM 2007.

[3] S. Kandula, K. Lin, T. Badirkhanli and D. Katabi, "FatVAP: Aggregating AP Backhaul Capacity to Maximize Throughput", NSDI 2008.

[4] R. Chandra, P. Bahl, and P. Bahl. "MultiNet: Connecting to Multiple IEEE 802.11 Networks Using a Single Wireless Card", INFOCOM, 2004.

[5] A. Sharma, E. M. Belding, "FreeMAC: Framework for Multi-Channel MAC Development on 802.11 Hardware", Presto 2008.

[6] The MadWiFi driver, `http://madwifi-project.org/`

[7] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M.F Kaashoek, "The Click modular router", in ACM Transactions on Computer Systems, volume 18, pages 263-297, August 2000.

[8] A. J. Nicholson, S. Wolchok, and B. D. Noble "Juggler: Virtual Networks for Fun and Profit", to appear in IEEE Transactions on Mobile Computing.

[9] `http://datasheets.maxim-ic.com/en/ds/MAX2820-MAX2821A.pdf`

[10] D. Malone, I. Dangerfield, and D. Leith, "Verification of Common 802.11 MAC Model Assumptions", PAM 2007.

[11] M. Dischinger, A. Haeberlen, K. P. Gummadi, S. Saroiu, "Characterizing Residential Broadband Networks", IMC 2007.