UNIVERSITY CARLOS III OF MADRID

Department of Telematics Engineering

Master of Science Thesis

VoIPiggy: Implementation and evaluation of a mechanism to boost voice capacity in 802.11 WLANs

Author: **José Pablo Salvador García**
Dipl.Eng. in Telecommunications

Supervisor: **Pablo Serrano Yáñez-Mingot, Ph.D.**

Leganés, September 2011

Abstract

Supporting voice traffic in existing WLANs results extremely inefficient, given the large overheads of the protocol operation and the need to prioritize this traffic over, e.g., bulky transfers. In this thesis we propose a simple scheme to improve the efficiency of WLANs when voice traffic is present. The mechanism is based on piggybacking voice frames from one direction of the voice call over the acknowledgments for the frames on the other direction, which reduces both frame overheads and time spent in contentions.

We evaluate its performance by means of analysis and in a large-scale testbed consisting on 30 commercial off-the-shelf devices. The experimental results follow those from the analytical model, obtaining dramatic performance improvements in both voice-only and mixed voice-and-data scenarios.

Table of Contents

Chapter 1

Introduction

IEEE 802.11 [12] is one of the most commonly used wireless technologies. It is being commoditized for voice communication, with the proliferation of smart phones with voice applications, e.g., Viber and Skype. Given the short length of voice frames, (i) the legacy DCF operation results extremely inefficient and (ii) the voice quality is highly vulnerable to data traffic. On the one hand, the inefficiency issue is not solved by introducing higher data rates, since they do not change the protocol overhead and do not significantly reduce the fraction of time wasted due to the 802.11 backoff mechanism. On the other hand, voice quality vulnerability can be reduced by means of EDCA [11] prioritization mechanisms. As a trade-off, the performance of data frame has to be reduced to sustain a decent level of quality for voice flows.

The impact of protocol overhead on VoIP has been extensively researched in the literature. The authors of [7] and [6] have investigated on the number of VoIP calls that can be supported in a WLAN with different 802.11 versions and different audio codecs. The degradation of voice performance in presence of low-priority data traffic has been analytically tackled in [2]. In that work, the authors propose an ACK skipping policy that optimizes the performance of voice frames. Other papers also discuss the importance of the MAC parameter settings on the voice performance, e.g., [16] and [9].

The literature also provides simulation results and experimental studies based on commercial off-the-shelf (COTS) devices to measure the capacity of WLANs when voice traffic is present. For instance, the authors of [16] show that appropriate MAC tuning can improve capacity ranging between 20% and 40%. Experiments reported in [1] confirm that commercial devices need non-trivial prioritization mechanisms in order to guarantee the quality of voice. Experiments in [6] show how voice conversations dramatically impair the performance of UDP data traffic since it reduces the available bandwidth.

Motivated by the limited efficiency of the standard operation of 802.11 under voice traffic, we propose a simple mechanism to dramatically reduce the overhead of the MAC operation, this resulting in a reduction of the number of contending nodes. Our proposal, named *VoIPiggy*, consists in a mechanism that piggybacks voice frames over MAC acknowledgments (ACKs). Our approach allows VoIP traffic to be served with lower delay and jitter, and, by embedding a significant part of voice frames into MAC ACKs, it reduces the average number of nodes contending for the channel, which eventually improves overall performance. We implement VoIPiggy on COTS devices, and validate its operation against the

legacy 802.11 operation.

The main contributions of this thesis can be summarized as follows:

- We propose a mechanism, VoIPiggy, to improve the general performance in the WLAN when voice traffic is present.
- We compare the performance of the legacy operation and VoIPiggy using theoretical models that are able to predict their performance in voice-only and mixed voice-and-data scenarios.
- We describe the implementation of VoIPiggy using COTS devices.
- We present an extensive performance evaluation in a large testbed of 30 nodes, confirming the results from the analysis. These experiments show that VoIPiggy practically doubles the capacity of a WLAN in terms of voice calls.

The rest of the thesis is organized as follows. In Chapter 2 we introduce the design of VoIPiggy and compare with respect to the legacy mechanism. Chapter 3 provides the analytical tools to evaluate the performance of VoIPiggy, both in a voice-only scenario and in a mixed scenario. In Chapter 4 we describe the implementation details of VoIPiggy over a COTS platform. Chapter 5 presents an extensive performance evaluation. Finally, Chapter 6 summarizes the main results drawn from our study and presents future lines of research.

Chapter 2

The VoIPiggy Mechanism

The basic access mechanism of the 802.11 standard, namely DCF (Distributed Coordination Function), is not well-suited to support voice applications. Given the short length of voice frames, DCF incurs a huge overhead, both in terms of backoff delay and MAC layers. In order to circumvent these limitations, we propose a simple modification to the MAC operation, named VoIPiggy, which basically consists in piggybacking the voice frames over the MAC acknowledgments.

In this Chapter we first summarize the legacy operation and provide a simple model to quantify its maximum achievable efficiency. Second, we present our proposed solution and compare its performance to the legacy approach. Finally, we present the required changes to the DCF state machine introduced by VoIPiggy.

2.1 Legacy operation

In this section we briefly summarize the 802.11 DCF mechanism [10]. With DCF, a station with a new frame to transmit senses the channel. If this remains idle for a period of time equal to the DCF interframe space parameter ($DIFS$), the station transmits. Otherwise, if the channel is detected busy, the station monitors the channel until it is measured idle for a $DIFS$ time, and then executes a backoff process. When the backoff process starts, the station computes a random number uniformly distributed in the range (0, $CW - 1$), and initializes its backoff time counter with this value. CW is called the contention window and for the first transmission attempt the minimum value (CW_{min}) is used. In case of a collision CW is doubled, up to a maximum value CW_{max}. As long as the channel is sensed idle, the backoff time counter is decremented once every time slot T_e. When a transmission is detected on the channel, the backoff time counter is frozen, and reactivated after the channel is sensed idle. When the backoff time counter reaches zero, the station transmits its frame in the next time slot. A collision occurs when two or more stations start transmitting simultaneously. An acknowledgment (ACK) frame is used to notify the transmitting station that the frame has been successfully received. If the ACK is not received within a given timeout, the station reschedules the transmission by reentering the backoff process. After a failed attempt, all the retransmissions of the same frame are sent with the retry flag set. If the number of failed attempts reaches a predetermined retry limit, the frame is discarded. Once the backoff process is completed, CW is set again to CW_{min}.

The standard operation of 802.11 introduces a large overhead for the case of voice traffic, given its small frame sizes. To quantify it, let us consider the exchange of two voice frames between two nodes, namely the Access Point (AP) and one station (STA). Neglecting the impact of the backoff operation for simplicity, the frame exchange will follow the one illustrated in upper part of Fig.2.1. According to the figure, the total time T_s required to perform this simplified two-frame exchange using DCF is given by:

$$T_s = 2\left(DIFS + 2T_{plcp} + \frac{H+L}{R} + SIFS + \frac{ACK}{R_c}\right), \tag{2.1}$$

where T_{plcp} represents the duration of the preamble, L is the payload length, H is the layer-2 header, ACK is the length of the acknowledgment and R and R_c are the transmission rates for data and control traffic, respectively.

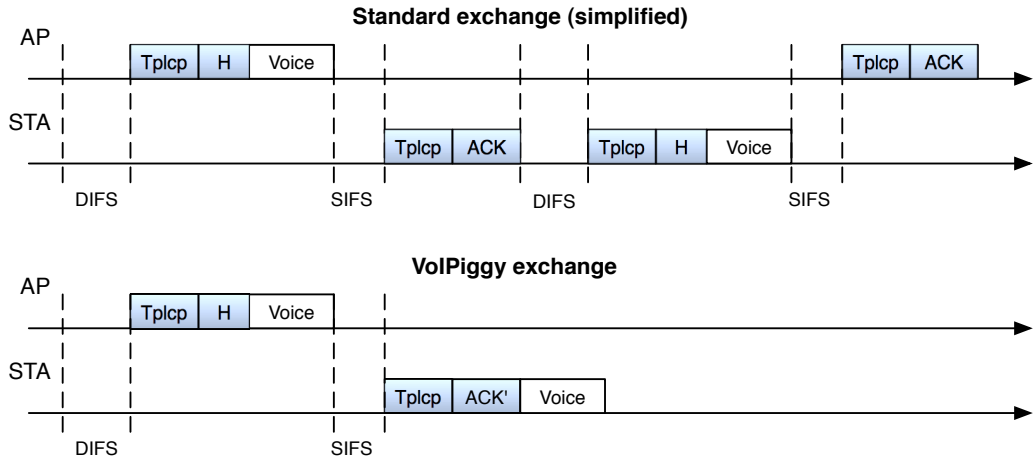


Figure 2.1: Simplified frame exchange (top) and VoIPiggy proposal (bottom).

In Table 2.1 we compute the value of T_s for different configurations of the physical layer, for a voice frame of 60 bytes transported over UDP. The results show that the total exchange T_s is significantly larger than the time actually devoted for payload transmissions $2L/R$, and the efficiency η_s, computed as

$$\eta_s = \frac{2L/R}{T_s},$$

worsens as the transmission rate R increases. In particular, for 2 Mbps, T_s is more than twice than the value of $2L/R$, while for 54 Mbps is almost 8 times larger.

2.2 VoIPiggy

2.2.1 Overview

In this section we detail the operation of our proposal. First, we note that there are two sources of inefficiencies, inherent to the bi-directional nature of voice conversations: (i) the station (STA) sends an ACK frame immediately after its voice frame. Even neglecting the

impact of the backoff, this basically doubles the introduced overhead (header and preamble) if a voice frame is immediately transmitted afterwards. Therefore, "merging" the upcoming voice frame with the precedent ACK frame seems an obvious choice to improve the efficiency; (ii) furthermore, if a voice frame is sent in reply to a received voice frame a $SIFS$ time after the reception, it will not need to contend for channel access, thus preventing resource wastage.

These two observations motivate the design of our mechanism, whose operation is illustrated in the bottom part of Fig.2.1. As the figure shows, a $SIFS$ time after the reception of the first voice frame, the STA sends in the same frame both the ACK and its voice frame towards the AP, which no further acknowledges its reception.[1] In addition, the VoIPiggy mechanism increases the free airtime, that is, more stations access to the medium, hence the number of voice calls allocated per infrastructure gets boosted.

In this way, we address the two sources of inefficiency identified above. Indeed, in this case the total time required for the two-frame exchange can be computed as:

$$T_v = DIFS + SIFS + 2T_{plcp} + \frac{H + ACK' + 2L}{R}, \tag{2.2}$$

where ACK' is the length of the modified acknowledgment header. As compared to the T_s values provided in Table 2.1, T_v provides time savings between 55% and 75%. Similarly to the previous case, we also compute the efficiency of VoIPiggy η_v. The table shows that the efficiency improvements of our scheme, denoted as $\Delta\eta$, reaches almost 90%.

Table 2.1: Total lengths of the frame exchanges of Fig.2.1

$Mode$	MCS [Mbps]		$2L/R$ [μs]	T_s [μs]	T_v [μs]	η_s	η_v	$\Delta\eta$ [%]
	R_d	R_c						
802.11b	1	1	1408	2968	2236	0.47	0.63	33
	2	2	704	1928	1340	0.37	0.53	44
	5.5	2	256	1338	770	0.19	0.33	74
	11	2	128	785	415	0.16	0.30	89
802.11g	6	6	235	503	377	0.47	0.62	33
	9	6	156	400	277	0.40	0.56	44
	12	6	117	348	227	0.34	0.52	53
	18	6	78	272	177	0.29	0.44	53
	24	18	59	246	153	0.24	0.38	61
	36	18	39	220	127	0.18	0.31	72
	54	18	26	203	111	0.13	0.24	82

It should be noted that the voice frame from the STA to the AP is not acknowledged. We argue that this is not very critical, given that the main source for frame losses are collisions, and this frame is protected from them as it is sent a $SIFS$ time after the medium was buys.

[1] Our approach might resemble the one proposed in the 802.11 standard [12] for the PCF protocol. However, in our case the stations are the ones piggybacking the data addressed to the AP

2.2.2 Specification

We next specify the changes introduced by VoIPiggy to the standard DCF state machine in the AP and the STA.

AP modifications

For the case of the AP, although the first transmission follows the standard exchange, the node should be able to decode the subsequent frame from the STA, that includes both the acknowledgment for the frame (which triggers its removal from the transmission queue) and a new frame to be delivered to the upper layer. Furthermore, this frame shall not be acknowledged. Then, in this case we just modify the standard operation for the reception.

In Fig. 2.2 we illustrate with a dashed line the modification introduced by VoIPiggy over the state machine for DCF. As the figure shows, the main modification is after the AP waits for the ACK; if the length of the received ACK is larger than the default one, implying that it is a modified one, the AP needs to pass the embedded voice frame to the upper layers.

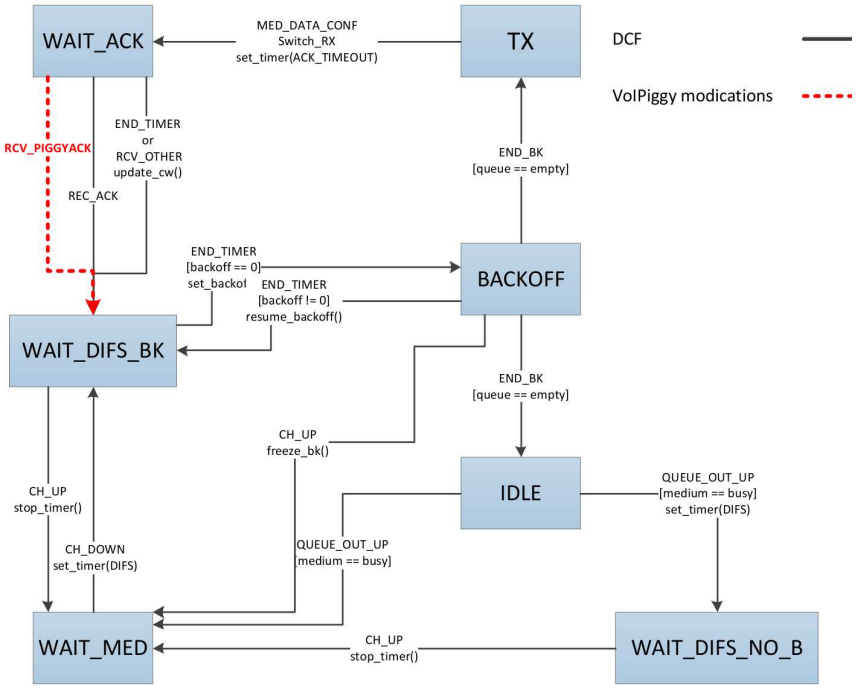


Figure 2.2: Changes introduced by VoIPiggy to the DCF TX State machine (AP).

STA

For the case of the STA, when a new frame from the AP is received the standard acknowledgment should be sent only if there are no pending voice packets towards the AP. Furthermore, to maximize the probability of piggybacking, incoming voice frames from the application layer will be queued until a timeout expires. In addition to these modifications, in

order to prioritize voice traffic we will set the minimum contention window to $CW_{\min} = 2$, which is the minimum allowed by the standard.

Fig. 2.3 illustrates the modified state machine for the standard reception mechanism. As before, the dashed lines show the main changes introduced. Upon the reception of a voice frame from the AP, the STA checks if it has pending voice traffic addressed to the AP; if that is the case, the station piggybacks the voice frame over the ACK.

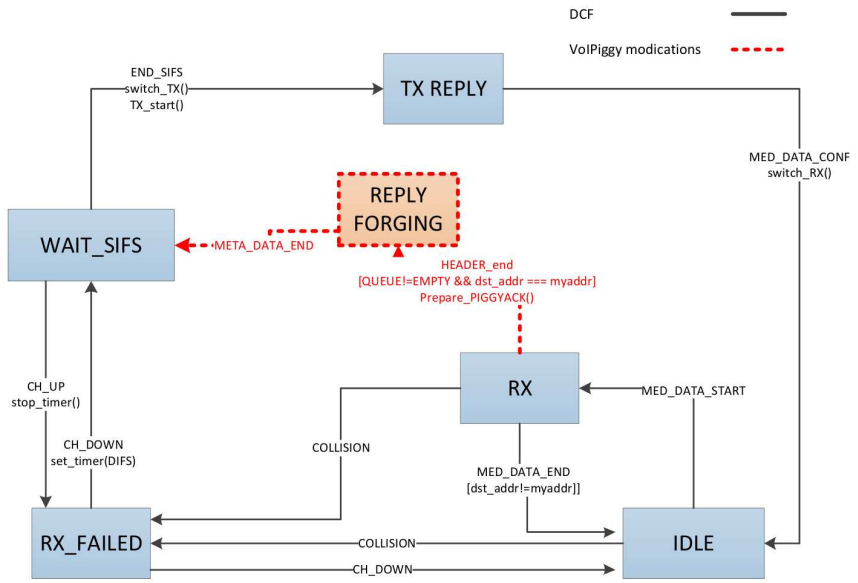


Figure 2.3: Changes introduced by VoIPiggy to the DCF RX State machine (STA).

A detailed description of the required changes to implement VoIPiggy over a COTS platform is provided in Chapter 4.

Chapter 3

Performance Analysis

In this chapter we present a theoretical analysis of VoIPiggy performance. More specifically, first we compute the maximum number of conversations that can be supported in a voice-only scenario, following a simple bandwidth-based model. Next we present the model for the distribution of the delay in the WLAN, which supports a finer call admission control based on, e.g., guaranteeing the 90th-percentile of the delay. Then we analyze throughput and delay in the case of a mixed voice-and-data scenario.

3.1 Voice-only scenario

3.1.1 Maximum number of conversations

Let us assume an 802.11 WLAN with one Access Point and n stations, where all conversations are destined outside the WLAN. The VoIP application periodically generates a packet of L_v bytes every T_f seconds, and each packet is served in T_v seconds, according to (2.2). Following a similar model to the one of [7], the system will not be congested as long as the total traffic arrival $\lambda = n/T_f$ is below the service rate $\mu = 1/T_v$. Therefore the maximum number of voice conversations n^* is given by:

$$n^* = \left\lfloor \frac{T_f}{T_v} \right\rfloor . \tag{3.1}$$

3.1.2 Delay performance

We next analyze the delay performance of a WLAN using VoIPiggy with n stations. We focus on the downlink direction, i.e., traffic from the AP to the STAs, as it constitutes the "bottleneck" in our scenarios. Indeed, the downlink is shared by n flows, while the uplink is ideally contention-free, since all voice frames are transmitted piggybacked within ACKs.

Given the setting of $CW_{min} = 2$, our system is a single server queue with periodic arrival process and deterministic service times, illustrated in Fig. 3.1. With this model, the survivor function $F^{-1}(t)$ for the queuing delay is given by [4]:

$$F^{-1}(t) = \frac{P_{n-1}(t, T_f, T_v)}{T_f^{n-1}} \ , \ \ x \geq 0. \tag{3.2}$$

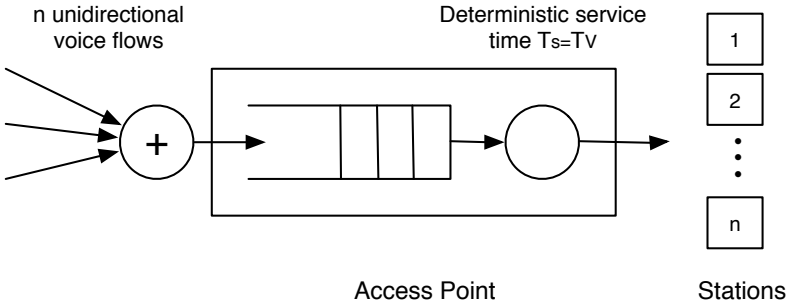


Figure 3.1: Queuing model for VoIPiggy, voice-only scenario.

$P_k(t,u,\epsilon)$ is computed for $t,\epsilon \geq 0$ and $k \geq 1$ as:

$$P_k(t,u,\epsilon) = \sum_{l=0}^{k-1} q_{k,l}(t,\epsilon)(u-k\epsilon+t)^l,$$

with the coefficients $q_{k,l}(t,\epsilon)$ computed recursively as follows:

$$q_{0,l}(t,\epsilon) = 0;$$

$$q_{k,0}(t,\epsilon) = \left((k\epsilon - t)^+\right)^k;$$

$$q_{k,l}(t,\epsilon) = \frac{k}{l}\sum_{j=l-1}^{k-2}\binom{j}{l-1}\epsilon^{j-l+1}q_{k-1,j}(t,\epsilon);$$

where $y^+ = \max(0,y)$. In this way, the cumulative distribution function (CDF) of the queuing delay can be obtained as $F(t) = 1 - F^{-1}(t)$, and its numerical derivative $f(t)$ supports the computation of any other statistic, e.g., the average or the standard deviation of the total delay.

For the computation of the total delay in the system, it should be noted that the time T_v (including the piggyback) determines the minimum time between consecutive queue departures, but the actual transmission time of a voice frame from the AP to the STA is given by:

$$T'_v = DIFS + T_{plcp} + \frac{H+L}{R}, \tag{3.3}$$

which does not include the ACK and the piggyback.

Therefore, the CDF for the total delay D for the delivery of downlink voice frames is a shifted version of $F(x)$:

$$D(t) = F(t - T'_v) \ , \ x \geq T'_v, \tag{3.4}$$

which terminates the delay analysis of VoIPiggy in a voice-only scenario.

3.2 Mixed voice and data scenario

Here we want to evaluate the performance of one fully backlogged data flow (i.e., saturated) in presence of one or more voice flows. We consider both the case of legacy 802.11 operation and VoIPiggy.

We assume that there is only one voice flow originating at each voice station, and the data station attempts to transmit a data frame as soon as the channel gets idle, following the legacy 802.11 MAC operation.

Note that the contention windows adopted by the AP for voice frames is set to $CW_{\min} = 2$, while for the case of data is given by W_d. For the case of VoIPiggy, we will assume that all upstream voice frames are piggybacked within MAC ACKs, so that the WLAN can be described in terms of two transmitters only, namely the AP and the data station.

3.2.1 Maximum number of conversations

Considering the case of VoIPiggy operation and n active voice flows, we can model the system serving voice frames as a queue with n CBR inputs, yielding a periodic arrival process, and a random service time which depends on the activity of the data station. In particular, the distribution of the service time of a voice frame is a function of the channel access probability of the data station, τ_d, whose computation will be detailed in Section 3.2.3.

Let T_v be the fixed time needed for the transmission of a complete voice exchange (downstream frame plus the corresponding piggybacked upstream frame), and T_c be the fixed duration of a collision, determined by the length of the data frame. Like [3], we assume that the collision probability is constant and independent among transmission attempts. Furthermore we assume that the length of backoff process is negligible since we assume that voice frames are transmitted with small contention windows, and collisions with data frames are much longer than the average backoff interval.

With the above, the distribution for the service time T_s of a voice frame is given by:

$$\begin{cases} Pr(T_s = T_v + kT_c) = \tau_d^k(1-\tau_d), & 0 \le k < m; \\ Pr(T_s = mT_c) = \tau_d^m, \end{cases} \tag{3.5}$$

where m is the maximum MAC retry limit. For large $m \gg 1$ or $\tau_d \ll 1$, the resulting average service time can be approximated by:

$$E[T_s] = T_v + \frac{\tau_d}{1-\tau_d} T_c. \tag{3.6}$$

Since the voice arrival rate at the AP is $\lambda_{AP} = n/T_f$, the system is stable if and only if the number of VoIP pairs is $n < T_f/E[T_s]$, yielding $\rho = \lambda_v\, E[T_s] < 1$. Therefore, the maximum number of conversations n_d^* that can be supported in presence of a data flow is given by:

$$n_d^* = \left\lfloor \frac{T_f}{T_v + \frac{\tau_d}{1-\tau_d} T_c} \right\rfloor \le \left\lfloor \frac{T_f}{T_v + \frac{2T_c}{W_d - 1}} \right\rfloor. \tag{3.7}$$

The inequality in (3.7) is obtained by considering that $\tau_d \le \frac{2}{1+W_d}$, which is the value achieved when no retransmission occurs. Comparing (3.1) to (3.7), and considering that

typical values for W_d are 16 or 32, and therefore $\tau_d << 1$, we conclude that the presence of a data flow has a negligible impact on the maximum number of sustainable conversations in the WLAN. As an illustrative example, Table 3.1 reports the maximum number of conversations that can be supported in 802.11b WLANs, for the legacy case and with and without data transmissions for the VoIPiggy solution, as computed via (3.1) and (3.7) for $W_d = 32$. As predicted, the values of n^* and n_d^* are very similar, a result further confirmed by the experimental evaluation in Section 5.

Table 3.1: Maximum number of conversations in 802.11b using VoIPiggy

Rate	1 Mbps	2 Mbps	5.5 Mbps	11 Mbps
n_{DCF} (without data)	6	10	14	25
n^* (without data)	8	14	33	47
n_d^* (with data)	8	13	32	45

3.2.2 Delay performance

With the above computation for the average service time, we can proceed with the approximated analysis of the delay of voice frames. Observing that the distribution (3.5) of T_s resembles in a geometric distribution, we assume it can be approximated by a random exponential variable with the same average of the original distribution, i.e., $\mu^{-1} = E[T_s]$. With this assumption our system becomes a $G/M/1$ queue with periodic arrivals.

In a $G/M/1$ system, the distribution of the queue size found by an arrival is geometric, and the cumulative distribution of the time spent waiting in the queue is given by [14]:

$$Q(t) = 1 - \sigma e^{-\mu(1-\sigma)t}, \quad \forall t \geq 0, \tag{3.8}$$

where σ is the unique solution in the range $(0, 1)$ of the following equation:

$$\sigma = A^*(\mu - \mu\sigma), \tag{3.9}$$

where $A^*(s)$ is the LST transform of the interarrival time process. The solution exists and is unique in the range $(0, 1)$ if the system is stable [14]. The cumulative distribution $F_W(t)$ has a step of amplitude $(1 - \sigma)$ in the origin, so that the associated *pdf* is:

$$p_Q(t) = \mu\sigma(1 - \sigma)e^{-\mu(1-\sigma)t} + (1 - \sigma)\delta(t), \quad \forall t \geq 0. \tag{3.10}$$

For the case of periodic arrivals for n VoIP connections, let us denote with a_i, $i = 1, 2, ..n$, the offset time in between the activation of a voice flow and the next. Due to the periodicity of the arrivals, the a_i represents the interarrival times at the AP, and each of the n possible interarrival values has a probability $\frac{1}{n}$, therefore exhibiting the following probability density function:

$$p_a(t) = \sum_{i=1}^{n} \frac{1}{n}\delta(t - a_i); \tag{3.11}$$

whose LST transform is given by:

$$A^*(s) = \frac{1}{n}\sum_{i=1}^{n} e^{-sa_i}. \tag{3.12}$$

Therefore, we can compute the distribution of the waiting time of a VoIP frame by numerically solving $\sigma = \frac{1}{n}\sum_{i=1}^{n} e^{-(\mu-\mu\sigma)a_i}$ and replacing σ in (3.8).

The total per-frame voice delay in presence of data traffic, namely D_d, can be computed by performing the convolution of the distributions of serving time (3.5) and waiting time (3.10). After some manipulations, the cumulative distribution of the delay can be expressed as follows:

$$G(t) = (1-\tau_d)\sum_{k=0}^{\left\lfloor \frac{t-T_v}{T_d} \right\rfloor} \tau_d^k \left[1 - \sigma e^{-\mu(1-\sigma)(t-T_v-kT_d)}\right]. \tag{3.13}$$

Recalling that D_d has been computed as the delay incurred in completing a voice exchange (a downstream frame plus an uplink frame, carried in piggyback), the actual delay incurred by the downstream frames is obtained by subtracting from D_d the duration of a SIFS plus an ACK with piggyback. i.e., by shifting backward $G(t)$ by $T_v - T_v'$:

$$D_d(t) = G(t + T_v - T_v'), \tag{3.14}$$

with T_v and T_v' given, respectively by (2.2) and (3.3).

3.2.3 Data Throughput

Our model for voice and data targets the computation of the throughput achieved by the data station when all active voice flows can be fit in the channel. To compute the throughput, we assume that, apart from MAC ACKs carrying piggybacked frames, the transmission activities of stations and AP are all independent.

Legacy operation. Using results from renewal theory, with n active voice flows, and with no piggyback adopted, the throughput of the data station can be expressed as follows:

$$X_d = \frac{P_d L_d}{P_d T_d + P_{AP} T_v + P_{VS} T_v + P_i T_i + P_{cd} T_{cd} + P_{cv} T_{cv}}, \tag{3.15}$$

where L_d and L_v represent the size of a data packet and a voice packet, respectively; the terms P_d, P_{AP}, and P_{VS} represent the success probability of a transmission from the data, AP, and voice station, respectively; the length of a data transmission is given by T_d, including the MAC ACK, while the AP and the voice stations transmit frames of length $T_v < T_d$, including the MAC ACK; P_i is the probability of an empty slot, whose length is given by T_i (e.g., 20 μs in 802.11g); P_{cd} is the probability that a collision with a data transmission occurs, lasting $T_{cd} > T_d$, and P_{cv} is the probability that a collision between voice stations occurs, lasting T_{cv}.

Similarly, the throughput of AP and voice stations is expressed by means of the same parameters:

$$X_{AP} = \frac{P_{AP} L_v}{P_d T_d + P_{AP} T_v + P_{VS} T_v + P_i T_i + P_{cd} T_{cd} + P_{cv} T_{cv}}; \tag{3.16}$$

$$X_{VS} = \frac{P_{VS} L_v}{P_d T_d + P_{AP} T_v + P_{VS} T_v + P_i T_i + P_{cd} T_{cd} + P_{cv} T_{cv}}. \tag{3.17}$$

Probabilities $P_{(.)}$ in (3.15) to (3.17) depend on the channel access probability $\tau_{(.)}$ of the different nodes and can be computed as:

$$\begin{aligned} P_{AP} &= \tau_{AP}\,(1-\tau_d)(1-\tau_{VS})^n; && (3.18)\\ P_{VS} &= n\,\tau_{VS}\,(1-\tau_{AP})(1-\tau_d)(1-\tau_{VS})^{n-1}; && (3.19)\\ P_d &= \tau_d(1-\tau_{AP})(1-\tau_{VS})^n; && (3.20)\\ P_i &= (1-\tau_d)(1-\tau_{AP})(1-\tau_{VS})^n; && (3.21)\\ P_{cd} &= (1-P_{AP}-P_{VS}-P_d-P_i)\,\tau_d; && (3.22)\\ P_{cv} &= (1-P_{AP}-P_{VS}-P_d-P_i)(1-\tau_d). && (3.23) \end{aligned}$$

The transmission attempt probability of a fully backlogged stations is given by [3]:

$$\tau_d = \frac{2}{1+W_d+W_d\,p_d\sum_{i=0}^{m-1}(2\,p_d)^i}, \tag{3.24}$$

where W_d is the minimum contention window of the data station, m determines the maximum contention window, and $p_d = 1-(1-\tau_{AP})(1-\tau_{VS})^n$ is the collision probability of the data station, given that the data station attempts to transmit a frame.

In order to compute the access probabilities, and hence the throughputs, for a given value of n, note that there are basically two possibilities for the case of voice traffic:

- All traffic is served, and therefore the throughput obtained by the AP is n times the throughput obtained by a station. In this case, the equations required to compute τ_{AP} and τ_{VS} are derived from

  $$X_{AP} = n \cdot X_{VS} = \frac{nL_v}{T_f}$$

  where T_f represents the voice interarrival time.

- Not all voice traffic is served. In this case, voice stations and the AP become saturated, and therefore their transmission probabilities are computed using a equation similar to (3.24).

The above constitutes a system of non-linear equations that can be solved using numerical techniques. We first assume that voice traffic is not saturated, and check if the numerical solution is inline with the assumption $X_{AP} = nL_v/T_f$. If this is the case, the analysis is completed; if not, we then assume that voice traffic is saturated, and solve the system of equations correspondingly. Note that in the latter case we conclude that the WLAN does not support the given number of voice flows n.

VoIPiggy operation. Using VoIPiggy, we assume that all uplink voice transmissions are piggybacked, so that the only contending nodes in the network are the AP and the data station. Therefore, the throughput of the data station and the one of the AP can be expressed as follows:

$$X'_y = \frac{P_y L_y}{P_d T_d + P_{AP} T_{AP} + P_i T_i + P_c T_c}, \tag{3.25}$$

where $y = \{d, AP\}$, $L_{AP} = L_v$, and T_{AP} is the time needed to transmit a voice frame and receive the corresponding MAC ACK containing a piggybacked voice frame. Eq. (3.25) is

similar to (3.15) and (3.16), but for the fact that there are no voice stations to be accounted for, and there is only one type of collision which involves a voice frame and a data frame. The probabilities P_{AP}, P_d, and P_i in (3.25) can be derived from (3.18), (3.20), and (3.21) for $n = 0$ and $\tau_{VS} = 0$, while $P_c = (1 - P_i - P_{AP} - P_d)$.

In this case, assuming that voice traffic is always served (which is the scenario most interesting, as it supports the provision of QoS guarantees), we have the following a system of two equations for τ_{AP} and τ_d:

$$\begin{cases} \tau_d = \dfrac{2}{1 + W_d + W_d\,\tau_{AP} \sum_{i=0}^{m-1} (2\,\tau_{AP})^i}; \\ X_{AP}(\tau_d, \tau_{AP}) = r_{AP} = \dfrac{nL_v}{T_f}; \end{cases} \tag{3.26}$$

Like in the previous case, the assumption on the non-saturation cannot be accepted if the resulting τ_{AP} is higher than the value imposed by the adopted contention window in a fully backlogged scenario, and therefore we conclude that the WLAN cannot support n voice connections.

Chapter 4

Implementation Details

In this chapter we detail the implementation of the most relevant features of VoIPiggy. We first briefly describe the platform used for our implementation, namely the Broadcom chipset with the b43 driver and the OpenFWWF firmware. Subsequently, we describe the overall architecture and the modifications to the Linux kernel and the device firmware to implement the VoIPiggy mechanism.

4.1 Linux and 802.11

The 802.11 network stack in Linux, depicted in Fig. 4.1, spans over three layers[1]:

- The hardware agnostic interface module *mac80211*. This module takes care of all operations required to handle 802.11 traffic, from packet generation and rate control to authorization and BSS setup.

- The *device driver*, which is a wrapper between the Linux internal 802.11 packet buffers and the physical device. It adapts packets to the proprietary device layout by translating payload, abstracts access parameters (like rates and number of attempts), and pushes/pulls packets to/from the device through Direct Memory Access transactions. It also accesses device registers to tune radio and MAC parameters according to upper layers indications.

- The *device internal logic*, commonly referred to as *firmware*, controls time critical operations such as the ACK sending or the packet retransmission. These operations are offloaded within the hardware and hidden to both *mac80211* and device drivers, due to the unpredictable latency and jitter that affect hardware interfaces like PCI bus.

Since this is a proof-of-concept implementation, we opt to work at both the device driver and internal logic levels leaving *mac80211* unchanged.

[1] Linux Wireless: `http://linuxwireless.org/`

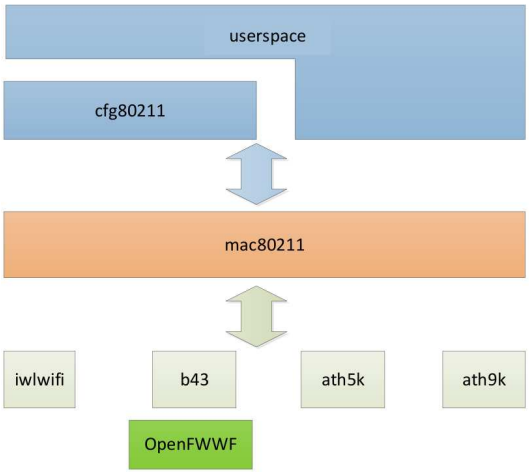


Figure 4.1: Framework of mac80211 module.

4.2 The Broadcom chipset

The Broadcom chipset, whose architecture is depicted in Fig.4.2, is based on a MAC processor (MP) that coordinates the data exchange among the different device blocks by running a binary firmware (FW) code. This code drives the transitions of the protocol state machine by reacting on external conditions, such as the arrival of a new frame or the expiration of programmable timers. The main blocks that compose the chipset are:

- **Direct Memory Access(DMA) Controller and MMIO** A programmable DMA Controller controls the data traffic from/to the host. It interfaces the MP to the host kernel and the access to all device registers is MMIO-based.
- **TX FIFO queues**: These queues are driven by the DMA controller and deliver outgoing packets composed by the host kernel.
- **TX Engine**: Given a sequence of bytes from a selected source (e.g., a TX queue), the TXE prepares a frame for transmission adding the PLCP header and the CRC coefficient. Then, it waits for a transmission opportunity within a selected InterFrame Space (IFS) time. As soon as an empty slot is detected, the frame is delivered to the air medium.
- **Template RAM** Here the MP forges arbitrary packets to be transmitted, e.g., Acknowledgments and Beacons.
- **RX Engine and FIFO queue**: The RXE decodes the signal received from the air, checks the validity of the CRC coefficient and reports the length of the received packet.
- **SHared Memory (SHM)**: The MP maintains the state variables and may copy portion of TX and RX packets for inspection.

- **Hardware registers and External Conditions (EC)**: The MP sets these hardware registers in response to changes in the EC to program the radio interface and set up timers.

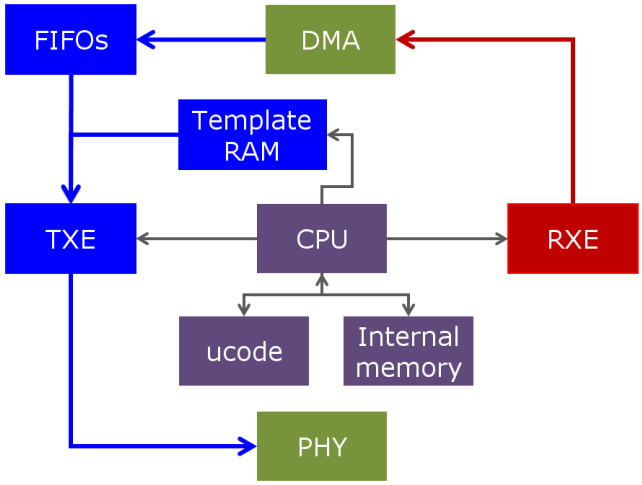


Figure 4.2: Architecture of a Broadcom chipset.

The Broadcom chipset is supported in the Linux kernel by the open source driver, *b43*. The b43 driver passes each outgoing packet to the device together with a long block of data to setup the hardware on a per packet transmission basis, according to *mac80211* decisions. New private data can be passed to the device logic by simply extending this data structure.

The b43 driver loads the firmware at Stuart's, therefore a different non-official firmware can be used replacing the original one. Thus, we use the OpenFWWF firmware that enables a very flexible customization of time critical operations. This firmware has already been used to implement some MAC customizations, e.g., a Block Based Recovery (BBR) algorithms [8] or TCP ACK-piggybacking [5].

4.3 Implementation of the VoIPiggy

The VoIPiggy exchange described in Section 2.2 involves a legacy data frame from the Access Point followed by a Data+ACK frame sent by the corresponding station after a SIFS. For simplicity we decide to implement our VoIPiggy reply by extending a legacy ACK. To accomplish this, we append the VoIP payload together with the IP and UDP headers skipping part of the data-type MAC header, as we are just interested in sending the payload and the MAC header is already provided by the acknowledgment. Furthermore, the MAC address of the sending station is added between the legacy ACK frame and the appended IP packet, so that the AP can recognize it.

The legacy mode is used by the AP for any kind of packets it transmits. Meanwhile, VoIPiggy mode is used by the station whenever it receives a VoIP packet incoming from the AP and the Head-of-Line (HOL) packet in its TX FIFO queue is VoIP-data. This is identified

by checking the data stored in the structure `sk_buff`, which handles the network packets that are received or to be transmitted:

Listing 4.1: HOL packet checking at driver level

```
if(data[0] == 0x08 &&             /*No QoS data*/
   fragment_len >= 32 + 28 &&     /*At least an IP+UDP header*/
   data[32] == 0x45 &&            /*IP header=20 bytes*/
   data[32 + 9] == IPPROTO_UDP) { /*L4 proto is UDP=0x11*/
        ...
        if(LEN < MAX_PIGGYBACK_PACKET_LENGTH &&
          (ntohs(*udpdst) == PIGGY_BACK_PORT ||
            ntohs(*udpsrc) == PIGGY_BACK_PORT)) {
               is_voip = 1;    /*HOL packet marked as VoIP*/
        }
}
```

In case the queue is empty or the HOL packet is not VoIP-type, the station will use the legacy mode.

4.3.1 Driver modifications

In order to develop our mechanism we modify the *b43* driver so that outgoing UDP traffic toward or from a specific port are marked as `PIGGYBACK_ENABLED` (PE), by extending the private data structure `b43_txhdr` used by the driver to push outgoing packets to the device. Besides this flag, we add some fields to tell the firmware how many bytes to skip from the MAC header and the exact length of the resulting packet when activated the VoIPiggy mode. Other parameters concerning the set up of the resulting transmission and that turn out to be infeasible to be computed by the MP are included as well.

We adjust the *b43* driver to optionally force the transmission of the VoIP traffic at a given modulation coding scheme configured by the user, which makes possible a better assessment of the VoIPiggy efficiency. Finally, we add a hook in the receiver code to intercept long ACK frames from the device. The driver must transform them back into full featured data packets by moving the IP section, inserting the missing MAC parts and finally sending them to the *mac80211* module.

At the receiving process, we must handle the reception of the DATA+ACK frames that are placed in the buffer `skb_buff`. The function `b43_rx` is the one responsible for passing a frame to the *mac80211* module. Here we reconstruct the data frame by adding the correspondent data-type MAC header, consequently upper layers can decode the packet as UDP type. In addition, we add here some statistics in order to achieve a better understanding of performance and to support a better debugging. These statistics include the number of piggyback packets received or the number of legacy ACKs. For that purpose, we define a struct named `voipiggy_t` within the struct `b43_wldev` for the wireless device, formed by variables dedicated to monitor transmission and reception statistics.

At the transmission side, we need to prepare the header for the firmware when a piggybacked frame is required. To this aim, we modify the function `b43_generate_txhdr` by adding the number of bytes to be skipped (`PIGGY_BACK_SKIP_BYTES`) and tagging the

packet as voipiggy eligible (`VOIP_TO_PIGGYBACK_TYPE`). Besides, we add some statistics `dev->voipiggy.tx_packets_might_piggyback` and set up the rate and the fallback rate to speed up the process at the firmware level.

4.3.2 Firmware modifications

In this section we detail the changes at the firmware level required for the AP and the STA.

Access Point

The code to be run in the AP is modified so that it can recognize long acknowledgments (i.e., VoIPiggy replies) by checking the received packet length as reported by the RXE engine. When such packets are received they are pushed directly to the host kernel. In addition, the AP is prevented from acknowledging VoIP packets if they were piggybacked.

Listing 4.2: ACK-type checking at the AP

```
mov SPR_RXE_FRAMELEN, GP_REG13;
je  GP_REG13, LEGACY_ACK_RX_LEN, discard_legacy;
mov 1, VOIP_PIGGY_RECEIVED;
```

Station

The firmware at the stations required a significant amount of changes, such as:

- Prevent STA from waiting for an ACK for the piggybacked voice frame. This is achieved by removing the HOL packet in the queue immediately after its transmission under VoIPiggy mode. To this aim, we set the following:

  Listing 4.3: STA modification

  ```
  // Remove the packet from the FIFO by
  // calling report_tx_status_to_host.
  jzx 0, 1, [TXHDR_HK4,off0], 0x000, skip_voip_piggyback_work;

  // Toggle acked bit
  orx 0, 0, 0x001, [TXHDR_STAT,off0], [TXHDR_STAT,off0];

  // Inform kernel only 1 tx attempt was done
  orx 3, 12, 0x01, [TXHDR_STAT,off0], [TXHDR_STAT,off0];
  jext  COND_TRUE, report_tx_status_to_host;
  ```

- Tuning the ACK transmission. Before transmitting the packet the firmware overwrites the first ten bytes to mimic a legacy ACK frame header. Then, our modification properly sets up the TXE engine (`SPR_TXE0_CTL`) when the PE flag is set by using the values precomputed by the kernel, i.e., bytes to

skip, packet length (SPR_RXE_FRAMELEN) and timing to use to build the PLCP (extra_fallback_info_for_pb).

- Delay outgoing packets when PE flag is set until a voice frame is received or a maximum threshold T is reached. By delaying VoIP frames (through the registers VOIP_PIGGY_TIME_L and VOIP_PIGGY_TIME_H) it is more likely that a voice frame from the AP will arrive and therefore the HOL frame can be piggybacked – otherwise, voice traffic might not benefit from our mechanism.

Figs. 4.3 and 4.4 illustrate the flow diagram for the VoIPiggy mechanism both for transmission and reception. We highlight the firmware modifications carried out to implement the solution. For the transmission state, these changes are subject to the AP that needs to wait for the ACK. The access point checks if the length of the received ACK is larger than the legacy one. In that case, the AP needs to send the voice frame up to the host. While, for the reception state, we modify the behavior of the station. Upon the reception of a voice frame from the AP, the STA checks if it has pending voice traffic addressed to the AP. In that situation, the station piggybacks the voice frame to the acknowledgment.

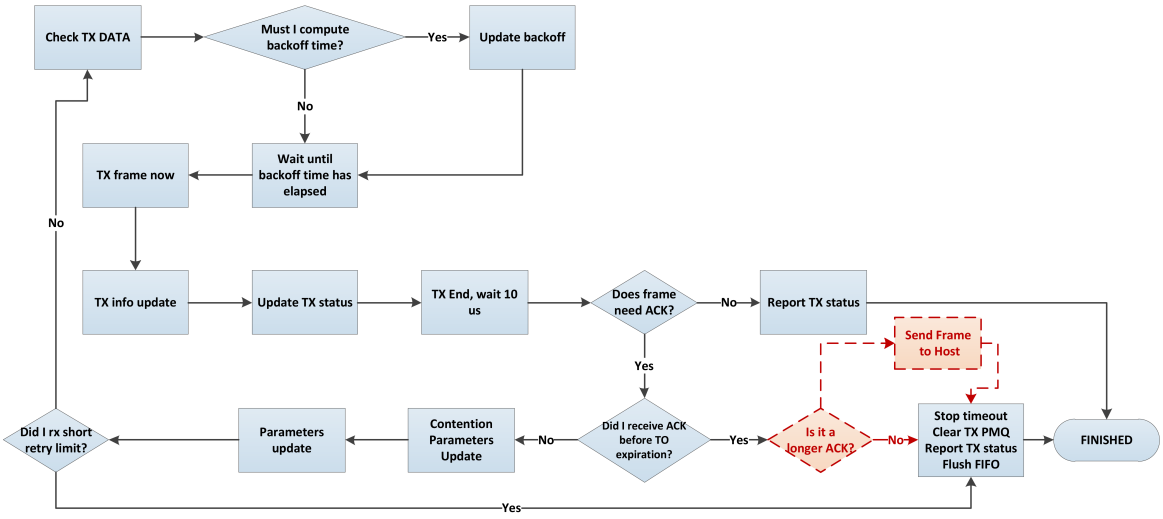
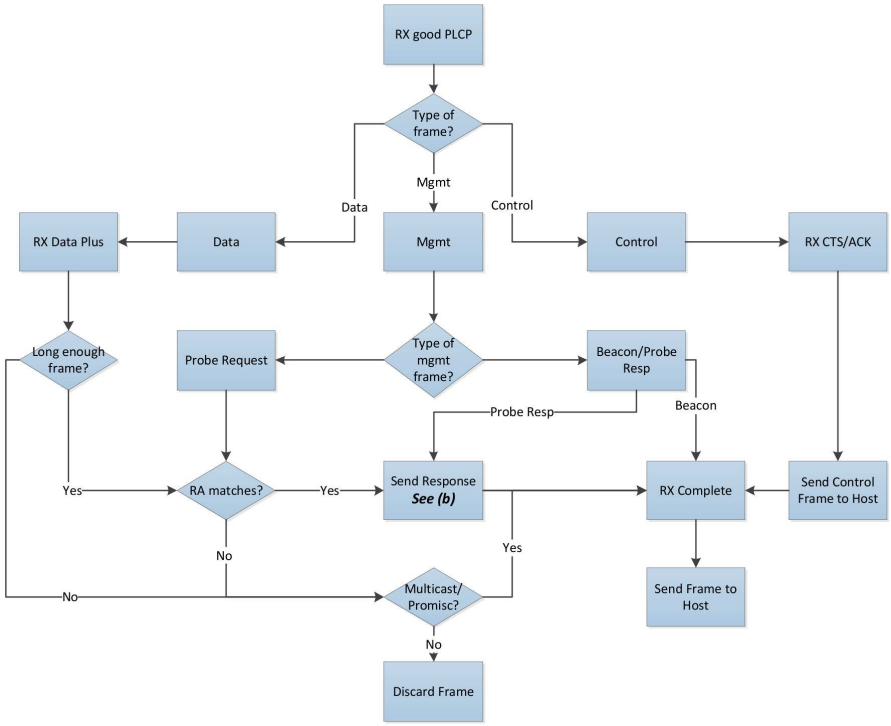
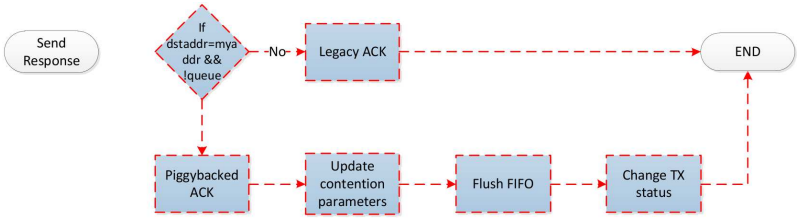


Figure 4.3: Flow diagram of the OpenFWWF TX operation supporting voipiggy.



(a) Complete flow diagram for OpenFWWF reception.



(b) Modified function to give VoIPiggy support

Figure 4.4: Flow diagram of the OpenFWWF RX voipiggy operation supporting voipiggy.

Chapter 5

Performance Evaluation

In this chapter we experimentally validate our VoIPiggy implementation by means of a real testbed deployment. Additionally, we compare experimental results and analytical results, obtained by running the models of Chapter 3 with the help of Matlab and Maple software. We show that our experimental results closely match obtained those analytically, confirming that our solution improves the performance when voice traffic is present in the WLAN.

5.1 Testbed Description

In this section we describe the testbed deployed to validate the proposed VoIPiggy mechanism. Our testbed is composed of 33 Alix 2d2 devices from PC Engine,[1] as depicted in Fig. 5.1. These embedded devices are popular low-cost computers, equipped with a Geode LX800 AMD 500 MHz CPU, 256 MB DDR DRAM, 2 Mini-PCI sockets and a Compact Flash socket, to which we attached a 4 GB card to make room for the installation of a current Linux distribution. As a wireless interface we installed a Broadcom BCM94318MPG 802.11b/g MiniPCI card, while as software platform we installed Ubuntu 9.10 Linux (kernel 2.6.29), using the modified b43 WLAN driver described in the previous sections. The selection of the wireless card is driven by the need to use *OpenFWWF*, an "Open source FirmWare for WiFi networks" [15], which supports large flexibility.

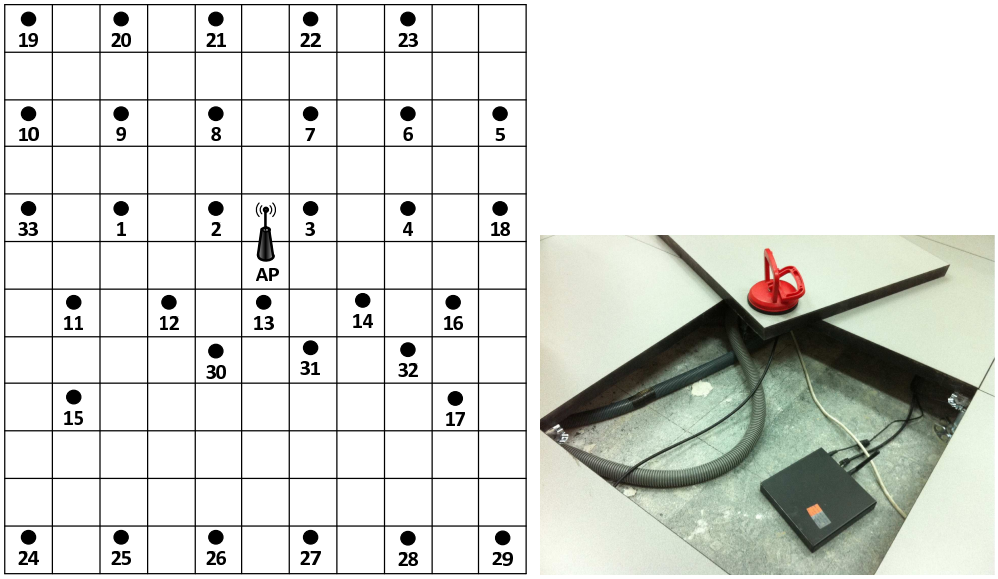
One of the devices acts as AP, while the rest are stations associated to the AP, distributed as Fig. 5.1 shows. All nodes are equipped with a 5-dBi omnidirectional antenna and use a transmission power of 27 dBm. Nodes that act as stations are spaced a few meters from each other (squares in Fig. 5.1 represent $60\ cm \times 60\ cm$ floor units), and the link quality is excellent for all nodes to communicate with each other.

The deployment is set up under a raised floor, which protects devices from physical harm and provides some radio shielding to some extent [17]. Configuration and control of the experiments is centralized in a single terminal, connected via wired Ethernet to all devices, which allows for automated execution of the experiments.

For traffic generation we use *mgen*,[2] which supports the computation of relevant voice

[1] PC Engines: `http://www.pcengines.ch/`

[2] The Multi-Generator Toolset: `http://cs.itd.nrl.navy.mil/work/mgen/`



(a) Testbed scenario (b) Detailed view of a node

Figure 5.1: Deployed testbed with 33 wireless nodes.

traffic metrics, such as delay, jitter, and loss rate. In particular, latencies can be evaluated at the receiver side, by means of the timestamps inserted by *mgen* in all packets, provided that all nodes are synchronized. We run the *PTP*[3] *daemon* over the wired interfaces of the nodes, achieving synchronization with μs accuracy.

We emulate the voice behavior by running independent instances of the *mgen* traffic generation tool, each transmitting a 60-byte voice frame every $T_f = 20$ ms. These are the characteristics of G.726 codec (ITU-T ADPCM[4] speech codec standard) [13]. In the case of data emulation, we use an instance of *mgen* run on a single station under saturation conditions with a packet length of 1472 bytes. The timeout threshold during which stations wait for a voice packet from the AP is set to 25 msec, this value being obtained after some calibration process.

5.2 Validation of the implementation

The first test consists of validating the proper operation of the VoIPiggy mechanism. For that purpose, we use the packet analyzer software wireshark [5] to verify that the voice packets transmitted by the stations are indeed piggybacked when expected, i.e., there are voice frames in the queue addressed towards the sender. Fig.5.2 captures the format of a piggybacked voice frame and compares it to a legacy ACK frame. The figure shows how the piggybacked frame contains the voice packet, highlighting the IP header, UDP header and UDP payload.

Upon receiving the acknowledgment which embeds the voice frame, the firmware passes

[3] *Precision Time Protocol*: `http://ptpd.sourceforge.net/`
[4] ADPCM: *Adaptive Differential Pulse-Code Modulation*
[5] Wireshark Project: `http://www.wireshark.org/`

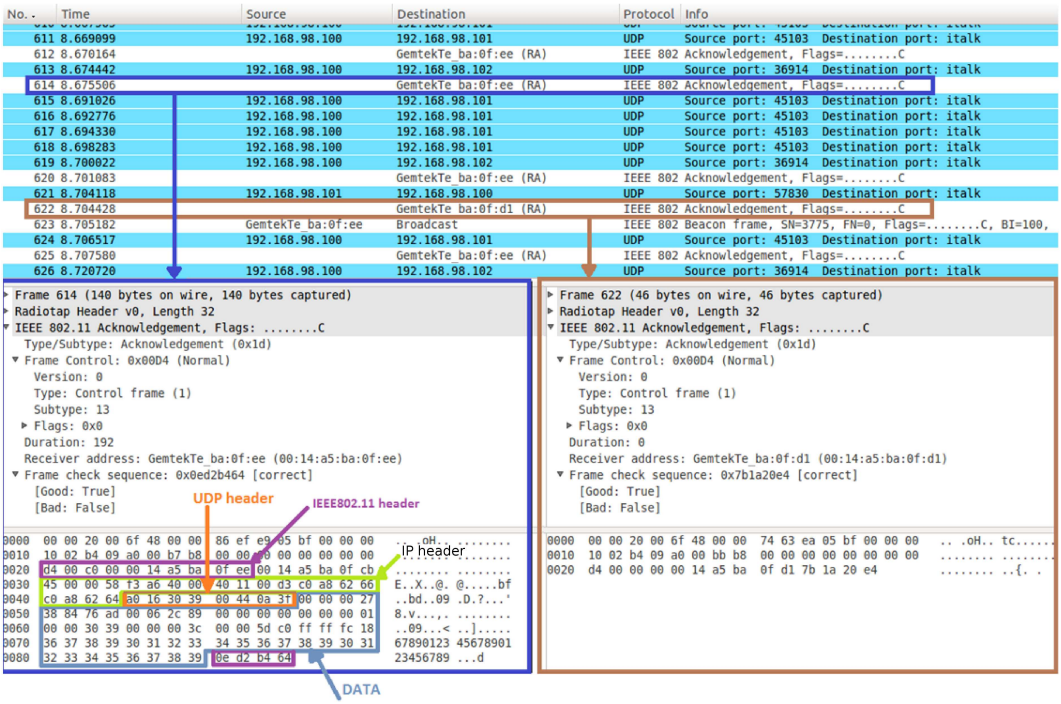


Figure 5.2: Comparison of a piggybacked ACK frame and a legacy ACK.

it up to the host. Once at the driver level, we reconstruct the frame according to a data frame type. Fig.5.3 depicts the typical frame exchange between AP and STA.

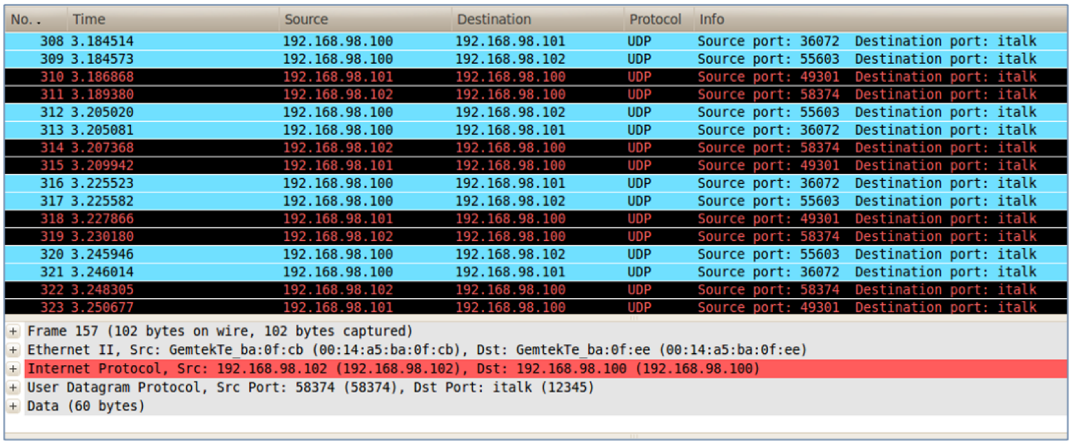


Figure 5.3: Voice frame exchange between AP-STA.

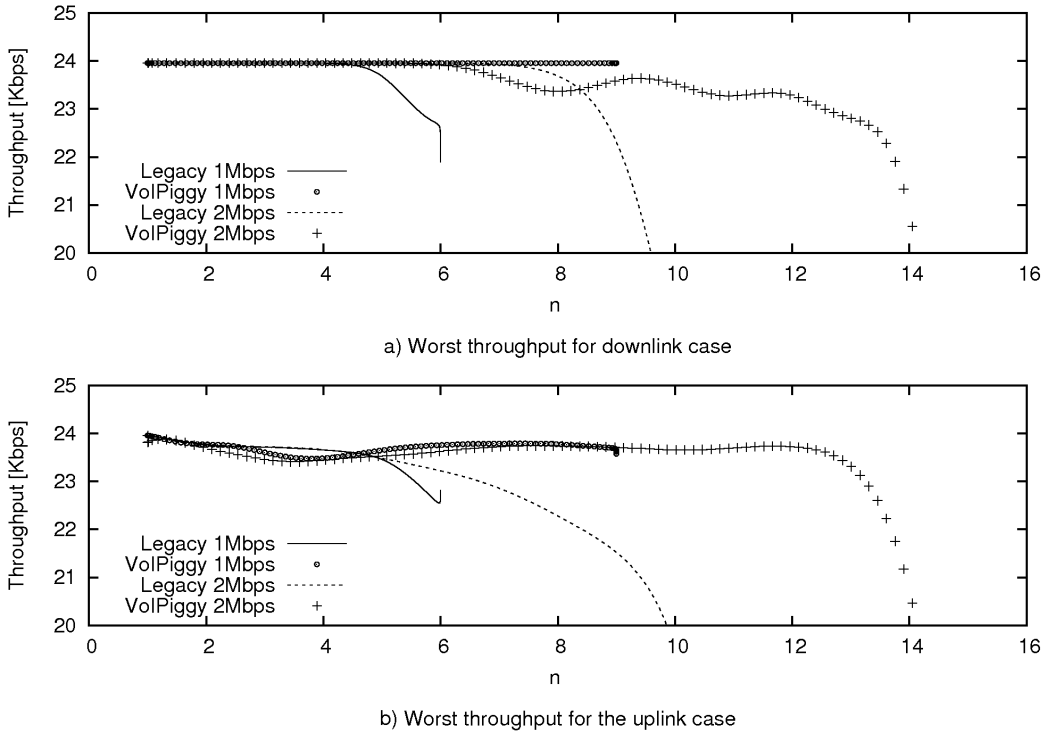


Figure 5.4: Throughput delay for downlink and uplink cases (voice only).

5.3 Voice-only scenario

We start our performance evaluation with a scenario in which only voice traffic is present. In all considered cases, we will assume that a voice call is active between a station in the WLAN and a node outside the WLAN, which is translated into a "downlink" (DL) flow from the AP to the wireless station, and a corresponding "uplink" (UL) flow in the other direction.

We first analyze the maximum number of flows that can be admitted in the WLAN. To this aim, we compute the obtained *mgen* throughput in each direction as a function of the number of voice flows n in the WLAN. The obtained results for the worst performing flows are depicted in Fig. 5.4 for the DL (top of the figure) and UL (bottom part of the figure) directions, for the case of standard DCF operation (denoted as *Legacy*) and for our VoIPiggy mechanism, for two modulation coding schemes (R) of 802.11b, namely $R = 1$ Mbps and $R = 2$ Mbps.

The results show that the use of VoIPiggy is able to significantly increase the number of voice conversations supported in the WLAN. Indeed, while for the legacy case the maximum n values before losses become very large are $n = 5$ for $R = 1$, and $n = 6$ for $R = 2$, for the case of VoIPiggy these values grow to $n = 8$ and $n = 13$, respectively. These results are in line with the model predictions reported in Table 3.1. Results show that VoIPiggy almost doubles the voice capacity in the WLAN.

Although these throughput results validate the performance improvements of VoIPiggy, they do not support more sophisticated QoS guarantees. To this aim, we next analyze the CDF of the DL delay using VoIPiggy, as this direction is the most likely to become the

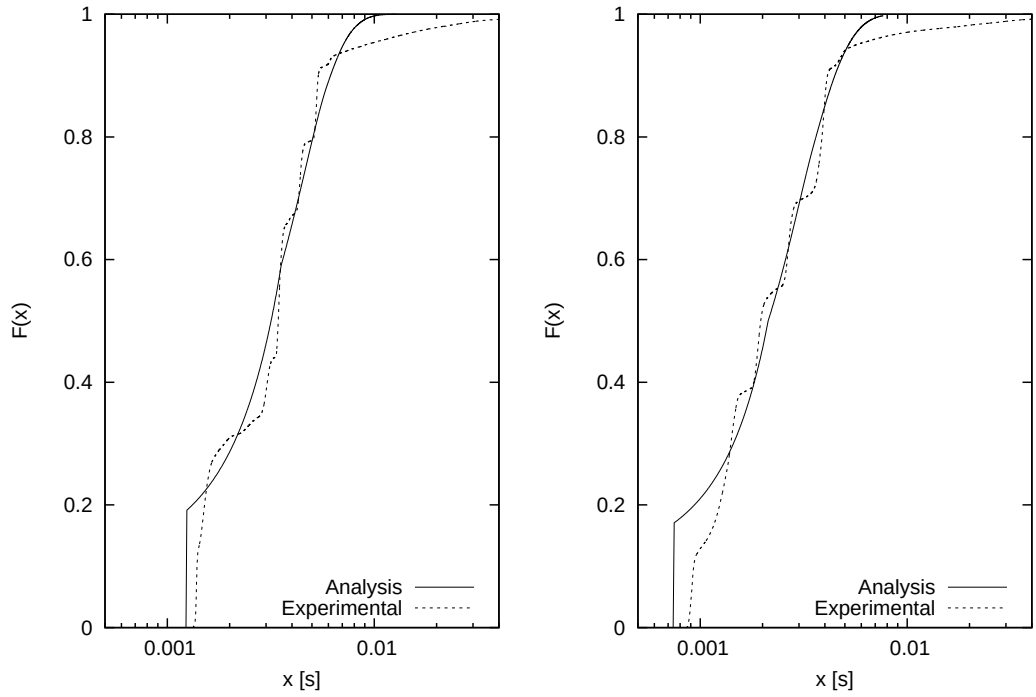


Figure 5.5: CDF of the delay for the voice model (voice only).

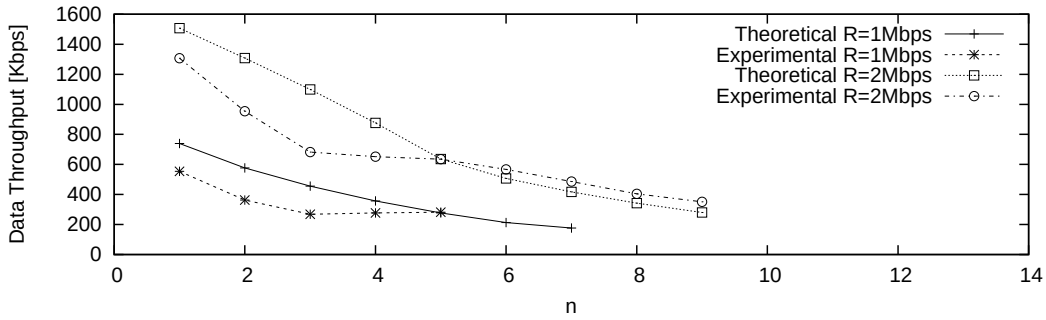
bottleneck in QoS provisioning scenarios (as the UL traffic will not have to contend for channel access). In Fig. 5.5 we plot the obtained CDFs for the experimental results (dotted line) and the analytical model given in Section 3.1.2 (straight line), for the case of $(n^* - 2)$ voice conversations. According to the Fig.5.5, both with rate $R = 1$ Mbps and $R = 2$ Mbps, the results show a good match between analytical and experimental values, which confirms the accuracy of the model. Furthermore, Fig.5.5 shows that the voice delay rarely exceeds 10 ms, and the delay dispersion is very limited. Therefore, we confirm the good properties of VoIPiggy, which is well suited to provide voice traffic with QoS guarantees.

5.4 Mixed voice-and-data scenario

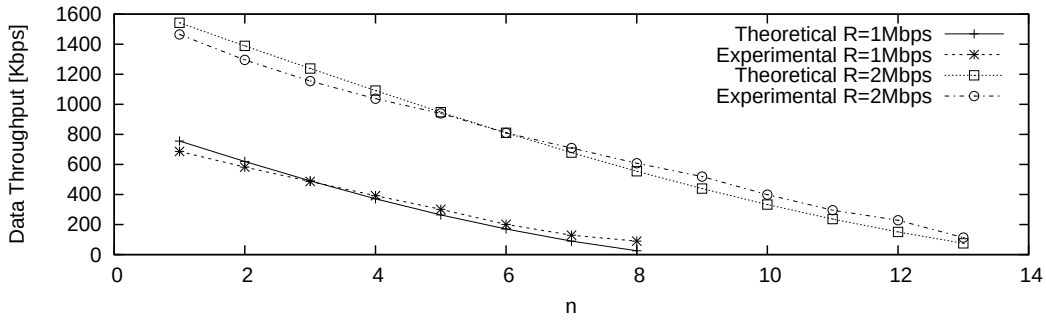
The performance evaluation of this scenario includes the evaluation of the maximum number of conversations in presence of a data flow, the evaluation of the data throughput performance, and the evaluation of the voice delay performance.

The upper part of Fig. 5.6 considers the legacy 802.11 MAC, and depicts the data throughput achieved by the data flow for an increased number of voice conversations, showing a good match between experimental and analytical results. The system cannot tolerate the presence of 2 voice flows with $R = 1$ Mbps, and no more than 3 flows with $R = 2$ Mbps. Therefore, legacy MAC turns out to be highly inefficient also in this scenario

The bottom part of Fig. 5.6 reports the results obtained with VoIPiggy. Results show that:



a) Data throughput for the legacy case



b) Data throughput for the VoIPiggy case

Figure 5.6: Data throughput for data plus voice scenario.

- The model can predict experimental results with high accuracy.
- The maximum number of possible conversations with VoIPiggy is equal to 8 and 13, respectively with $R = 1Mbps$ and $R = 2Mbps$, which coincides with the model predictions. reported in Table 3.1
- The data throughput, for a given value of n, is higher when using VoIPiggy as compared to DCF. This result shows that VoIPiggy increments the voice capacity of the WLAN and frees more resources for data flows.

Finally, let us consider the delay performance of voice in this scenario. Fig. 5.7 sketches the behavior of the delay CDF for $R = 1$ Mbps and $R = 2$ Mbps. The model is able to capture the behavior of the experiments, although it results optimistic.

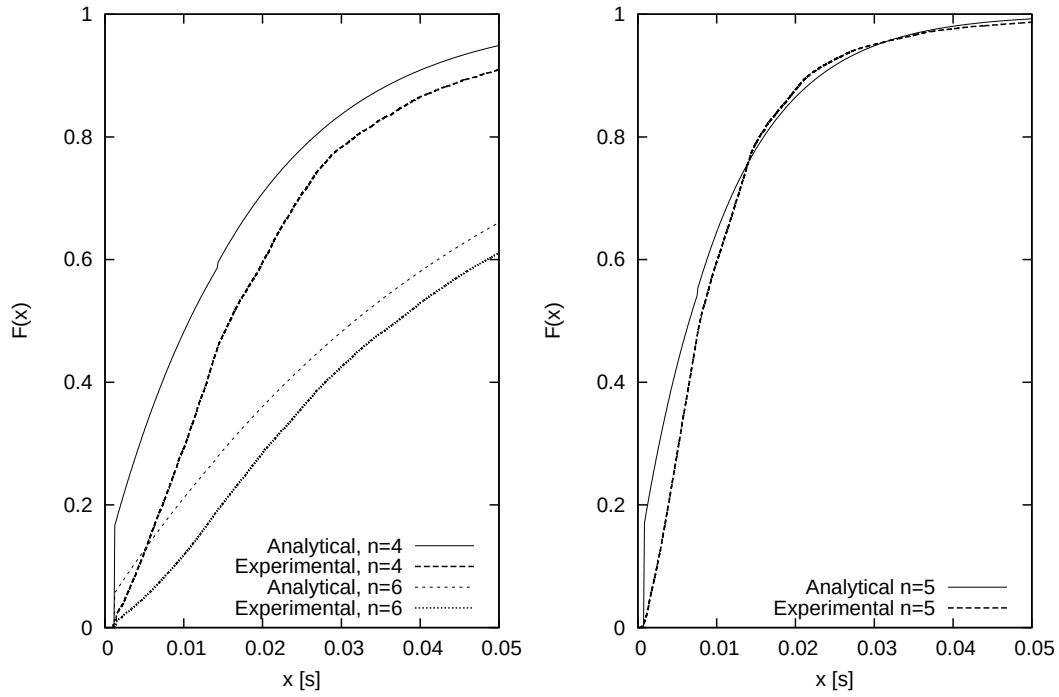


Figure 5.7: CDF of the voice delay for the data plus voice scenario.

Chapter 6

Summary and Future Work

In this thesis we have designed, analyzed, implemented and evaluated a mechanism to dramatically improve the efficiency of MAC operation when voice traffic is present in 802.11 WLANs. In contrast to legacy operation, which spends large amounts of time in contention and overhead transmissions, VoIPiggy extends the control frames sent from the stations to the AP with user data, thus practically halving the time required to transmit voice frames. Indeed, the analytical models presented confirm the good performance figures of VoIPiggy in voice-only and voice-and-data scenarios.

We have described the modifications required by VoIPiggy, which are supported by existing COTS devices. To validate VoIPiggy and assess its effectiveness, we have deployed a large-scale testbed consisting of 33 devices. Through extensive performance evaluation we have demonstrated the performance improvements yielded by our mechanism, these results being in a good match with analytical results.

As future work we envision the following tasks:

- Assessing the performance when using real VoIP clients, i.e.: Skype, Ekiga or Viber. We have already carried out some preliminary tests with Ekiga, obtaining good results.
- Implementing different MAC mechanisms apart from VoIPiggy, such as e.g. the AP piggybacks the packets addressed to the station. Note that this is more difficult since it requires analyzing the transmission queue to search if there is a packet addressed to a given station.
- Testing with higher modulation coding schemes and number of data stations.

References

[1] F. Anjum, M. Elaoud, D. Famolari, A. Ghosh, R. Vaidyanathan, A. Dutta, P. Agrawal, T. Kodama, and Y. Katsube. Voice Performance in WLAN Networks – An Experimental Study. In *Proceedings of GLOBECOM'03*, San Francisco, CA, USA, December 2003.

[2] A. Banchs, P. Serrano, and L. Vollero. Reducing the Impact of Legacy Stations on Voice Traffic in 802.11e EDCA WLANs. *IEEE Communications Letters*, 11(4), April 2007.

[3] G. Bianchi. Performance analysis of the IEEE 802.11 distributed coordination function. *JSAC*, 18(3):535–547, Mar 2000.

[4] A. Eckberg Jr. The Single Server Queue with Periodic Arrival Process and Deterministic Service Times. *IEEE Transactions on Communications*, 27(3):556 – 562, March 1979.

[5] P. Gallo, F. Gringoli, and I. Tinnirello. On the Flexibility of the IEEE 802.11 Technology: Challenges and Directions. In *Proceedings of the Future Network & Mobile Summit 2011*, Warsaw, Poland, June 2011.

[6] S. Gang and M. Kappes. An experimental study of throughput for UDP and VoIP traffic in IEEE 802.11b networks. In *Proceedings of WCNC*, New Orleans, LA, USA, March 2003.

[7] S. Garg and M. Kappes. Can i add a VoIP call? In *Proceedings of ICC'03*, Anchorage, Alaska, USA, May 2003.

[8] B. Han, A. Schulman, F. Gringoli, N. Spring, B. Bhattacharjee, L. Nava, L. Ji, S. Lee, and R. Miller. Maranello: Practical Partial Packet Recovery for 802.11. In *Proceedings of NSDI'2010*, pages 205–218, Mar/Apr 2010.

[9] G. Hanley, S. Murphy, and L. Murphy. Adapting WLAN MAC parameters to enhance VoIP call capacity. In *In MSWiM '05: Proceedings of the 8th ACM international symposium on Modeling, analysis and simulation of wireless and mobile systems (ACM MSWiM'05)*, pages 250–254, Montreal, Quebec, Canada, October 2005.

[10] IEEE 802.11 WG. IEEE Standard for Information Technology-Telecommunications and Information Exchange Between Systems-Local and Metropolitan Area Networks-Specific Requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, August 1999. IEEE Std. 802.11-1999.

[11] IEEE 802.11 WG. Amendment to Standard for Information Technology. LAN/MAC Specific Requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: Medium Access Control (MAC) Enhancements for Quality of Service (QoS). (802.11e), November 2005. Supplement to IEEE 802.11 Standard.

[12] IEEE 802.11 WG. IEEE Standard for Information Technology-Telecommunications and Information Exchange Between Systems-Local and Metropolitan Area Networks-Specific Requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, 2007. Revision of Std. 802.11-1999.

[13] U. It. ITU-T recommendation G.191. Technical report, International Telecommunication Union, 1990.

[14] L. Kleinrock. *Queueing Systems: Theory*, volume 1. J. Wiley & Sons, 1975.

[15] OpenFWWF: OpenFirmWare for WiFi networks. http://www.ing.unibs.it/openfwwf/.

[16] P. Serrano, A. Banchs, J. F. Kukielka, G. D'Agostino, and S. Murphy. Configuration of IEEE 802.11e EDCA for Voice and Data traffic: An Experimental Study. In *Proceedings of ICT-MobileSummit 2008*, Stockholm, Sweden, June 2008.

[17] P. Serrano, C. J. Bernardos, A. de la Oliva, A. Banchs, I. Soto, and M. Zink. FloorNet: Deployment and Evaluation of a Multihop Wireless 802.11 Testbed. *EURASIP Journal on Wireless Communications and Networking*, 2010.