VoIPiggy: Analysis and Implementation of a Mechanism to Boost Capacity in IEEE 802.11 WLANs Carrying VoIP Traffic

Pablo Salvador, *Student, IEEE,* Vincenzo Mancuso, *Member, IEEE,* Pablo Serrano, *Member, IEEE,* Francesco Gringoli, *Member, IEEE* and Albert Banchs, *Senior Member, IEEE*

Abstract—Handling voice traffic in existing WLANs is extremely inefficient, due to the large overhead of the protocol operation as well as the time spent in contention. In this paper, we propose a simple scheme (VoIPiggy) to improve the efficiency of WLANs with voice traffic. The key idea of the mechanism is to piggyback voice frames onto the MAC layer acknowledgments, which reduces both the frame overhead and the time wasted in contention. To quantify the gains of our proposal, we first study its performance by means of a capacity and delay analysis of a WLAN operating under the VoIPiggy mechanism. Then, we present an implementation of the mechanism using commercial off-the-shelf devices, which involves programming at the driver and firmware levels. The performance of the proposed scheme is evaluated in a large-scale testbed consisting on 30 devices. Our extensive measurements, which comprise different network conditions in terms of number of active nodes, traffic load and transmission rates, confirm that the experimental results match the analytical ones, and show a dramatic performance improvement for both "voice only" and "voice and data" scenarios.

Index Terms—WLAN, 802.11, experimental analysis, VoIP, performance, piggybacking, VoIPiggy, MAC enhancement.

1 Introduction

IEEE 802.11 [2] is one of the most commonly used wireless technologies nowadays. It is being commoditized for voice communications, with the proliferation of smartphones running voice applications like, e.g., Skype or Google Hangouts. Given the short length of voice frames, the operation of the legacy IEEE 802.11 Distributed Coordination Function (DCF) is extremely inefficient, and the voice quality is highly vulnerable to data traffic. This inefficiency is not alleviated by introducing higher data rates, since these do not change the protocol overhead and hence do not significantly reduce the fraction of time wasted due to the 802.11 backoff mechanism. While voice quality vulnerability can be reduced by means of the prioritization introduced by the Enhanced Distributed Channel Access (EDCA) mechanism [3], this incurs a substantial level of contention overhead and does not solve the problem of vulnerability to legacy traffic [4]. As a result, the performance of data traffic has to be reduced to sustain enough quality for voice traffic.

- *P. Salvador, V. Mancuso and A. Banchs are with Institute IMDEA Networks, 28912 Leganés, Spain, and with Univ. Carlos III de Madrid, 28911 Leganés, Spain. E-mail:{josepablo.salvador,vincenzo.mancuso,albert.banchs}@imdea.org*
- *P. Serrano is with Univ. Carlos III de Madrid, 28911 Leganés, Spain. E-mail: pablo@it.uc3m.es*
- *F. Gringoli is with Univ. of Brescia, Brescia, Italy. E-mail: francesco.gringoli@ing.unibs.it*

This paper is an extended version of our work "VoIPiggy: Implementation and evaluation of a mechanism to boost voice capacity in 802.11 WLANs" [1], presented at IEEE INFOCOM 2012 Mini-Conference.

Motivated by the low efficiency of the standard operation with voice traffic, in this paper we propose a simple yet effective mechanism to reduce the overhead of the MAC operation, which also results in a reduction of the time spent in contending for the wireless medium. Our proposal, called *VoIPiggy*, piggybacks voice frames onto MAC acknowledgments (ACKs). By embedding a significant part of voice frames into MAC ACKs, this approach reduces the MAC overhead *and* the average number of nodes contending for channel access, improving the overall system performance. As a result, VoIPiggy increases the number of voice and data flows that can be served as well as the delay performance of voice traffic. By implementing VoIPiggy in commercial off-the-shelf devices (COTS) and validating its operation against the legacy 802.11 operation, we show that it achieves dramatic performance improvements. The **main contributions** of this paper are:

- We propose a novel mechanism, VoIPiggy, which dramatically improves the overall performance of WLANs when voice traffic is present.
- We present a theoretical analysis of a WLAN operating under the VoIPiggy scheme for the throughput of the WLAN, its capacity region (i.e., the number of voice and data flows that can be supported) and the delay of voice traffic.
- We describe the implementation of our scheme on COTS devices, which introduces modifications at the driver and firmware levels.

- We present an extensive performance evaluation of our scheme in a large testbed of 30 nodes. These experiments confirm the accuracy of the analytical model, and show that VoIPiggy practically doubles the capacity of the WLAN.

The rest of the paper is organized as follows. In Section 2 we summarize the existing work in the area. The VoIPiggy mechanism is introduced and described in Section 3, and its performance is analyzed in Section 4. Section 5 describes the implementation of the proposed scheme and its functional modules. Section 6 presents our testbed and reports the extensive experimental evaluation conducted for the proposed algorithm in a wide set of network conditions. Finally, Section 7 closes the paper with some final remarks.

2 Related Work

The impact of protocol overhead on VoIP has been extensively researched in the literature. The authors of [5] and [6] have investigated 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 [4]. Other papers also discuss the importance of the MAC parameter settings on the voice performance, e.g., [7] and [8]. The scheme we propose achieves a much higher performance improvement than any proposal in the above papers.

The literature also provides simulation results and experimental studies based on COTS devices to measure the capacity of WLANs when voice traffic is present. For instance, the authors of [7] show that appropriate MAC tuning can improve capacity by 20% to 40%. Experiments reported in [9] confirm that commercial devices need non-trivial prioritization mechanisms in order to guarantee the quality of voice. Experiments in [6] show how voice conversations impair dramatically the performance of UDP data traffic since they reduce the available bandwidth. All the above papers rely on the default MAC protocol operation; in contrast, in this paper we implement a new mechanism at the driver and firmware level.

The VoIPiggy mechanism can be related to the *reverse direction* (RD) mechanism of 802.11n [10], in which an *RD initiator* (holding a TXOP) grants channel access to an *RD responder* during the TXOP. The RD mechanism has been evaluated using simulations, for the case of voice and data traffic in [11], and for the case of online gaming applications in [12]. In contrast to these works, in this paper we evaluate the performance of the proposed mechanism based on analytical and experimental results.

Besides, VoIPiggy has some similarities with the HCCA mechanism, in which the AP polls station for data. However, the use of HCCA has some shortcomings: (i) it introduces significant complexity, including, e.g., the scheduling configuration [13], (ii) it uses very small transmission rates for the data piggybacked on the CF-Poll frame, which results in the "CF-Poll piggyback problem" identified in [14], (iii) if the downlink voice frames are sent in the contention-free period (CFP), additional downlink delay is incurred while waiting for this period [15], and (iv) if they are sent in the contention-period (CP), the ACK for the uplink frame cannot be piggybacked, which leads to the use of three frames per exchange. In contrast, VoIPiggy achieves a substantially more efficient operation using a very simple scheme.[1] It is also worthwhile noting that HCCA has not been widely deployed, while here we present a working implementation of VoIPiggy.

In [16], authors present a novel framework for the rapid prototyping of new MAC schemes and show the use of piggybacking for TCP acknowledgments as an example; their approach differs substantially for ours, is based on a different implementation and is evaluated in a scenario with only two stations. Other former works have proposed the use of piggybacking to improve WLAN performance [17], [18], yet their evaluation is performed exclusively via simulations.

The proposal that mostly relates to our work is *Softspeak* [19], which consists in aggregating voice frames at the AP, and a TDMA-like operation for voice stations based on *coarse-grained* time slots of 1 ms. These coarse slots not only limit the scalability of the proposal, but also require dealing with, e.g., slot allocation and time synchronization between nodes. Furthermore, the performance evaluation is carried out in a small scenario consisting of 10 nodes, and lacks analytical support.

A preliminary description of VoIPiggy was recently presented [1]. This article extends [1] very substantially, by introducing an analytical model for its performance (capacity region and voice delay), and a much more thorough experimental evaluation.

3 The VoIPiggy mechanism

In this section, we discuss the VoIPiggy mechanism. We first present the motivation behind the scheme. Then, we describe the mechanism as well as the modifications that it introduces to the standard operation of access points (APs) and stations (STAs).

3.1 Motivation

The standard operation of 802.11 introduces a large overhead for voice traffic, given the small size of its frames and its bi-directional nature. To quantify this overhead, let us consider a scenario consisting of one AP and one STA, and the exchange of two voice frames between them, one per direction. Neglecting

1. For instance, a typical exchange at 48 Mbps (data frames) and 12 Mbps (control frames) for a 160-byte voice frame lasts 166 μs with VoIPiggy, while with HCCA in CP, it results in 227 μs.

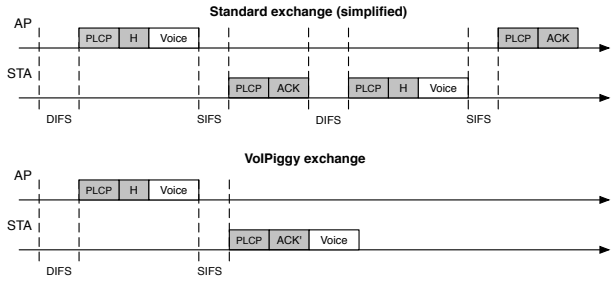


Fig. 1: Frame exchange for the standard case (top) and our VoIPiggy mechanism (bottom).

TABLE 1: Efficiency of the standard for the case of the short-length frame exchange of Fig. 1

Mode	R (Mbps)	R_c (Mbps)	T_{std} (μs)	$T_{\min}$ (μs)	η	$\frac{2T_{ack}}{T_{std}}$
802.11b	1	1	3084	1408	46%	20%
	2	2	1964	704	36%	26%
	5.5	2	1323	256	19%	39%
	11	2	1139	128	11%	45%
802.11g	6	6	553	235	42%	20%
	9	6	441	156	35%	25%
	12	6	385	117	30%	28%
	54	24	226	26	12%	36%

the impact of the backoff operation, the frame exchange follows the operation illustrated in the upper part of Fig. 1. According to this, the total time required to perform this exchange following the standard operation is given by:

$$T_{std} = 2\left(DIFS + 2\,T_p + \frac{H + l_v}{R} + SIFS + \frac{ACK}{R_c}\right), \quad (1)$$

where $DIFS$ and $SIFS$ are constant times defined by the standard, T_p represents the duration of the preamble, l_v is the length of the voice frame, including the IP and UDP headers, 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.

The minimum time required to perform the exchange is given by the time devoted to exchange voice data without any overhead, i.e., $T_{\min} = 2l_v/R$. Based on this, we can define the *efficiency* of the standard exchange as $\eta = T_{\min}/T_{std}$. In Table 1, we provide some values of this (in)efficiency for different combinations of the modulation and coding scheme (MCS) of data and control frames, for 802.11b and 802.11g, assuming a frame length of $l_v = 88$ bytes, which results from adding the corresponding IP and UDP headers to a voice frame of 60 bytes.

The values obtained show that, for these typical MCSs, efficiency becomes worse as transmission rates increase, and reaches values as high as 46%. Note that we have considered the best conditions (i.e., only two stations, no backoff operation) and therefore this is the *best possible case* for the efficiency, which will be even smaller under more realistic conditions. Motivated by this poor figures, we next present the modifications introduced by VoIPiggy to improve performance.

3.2 Description

We identify the following sources of inefficiency in the standard exchange depicted in Fig. 1: (i) after the reception of the voice frame from the AP in the "downlink" direction, the STA has to send two consecutive frames in the "uplink" direction (i.e., to the AP), namely an acknowledgment frame and a voice frame, separated by a $DIFS$, each of which involves a substantial overhead; (ii) furthermore, the second of these two frames is transmitted after contending for channel access, which adds an extra overhead; and (iii) the AP finally sends an acknowledgment frame back to the STA to confirm the correct reception of the second voice frame, which again involves an additional overhead. In order to eliminate these sources of inefficiency, our mechanism relies on the following key ideas:

- The first key idea of our proposal is to send the voice frame in the uplink direction after a $SIFS$ following the transmission of the voice frame in the downlink direction. In this way, we take advantage of the fact that the medium is already "cleared" for transmission after the first voice frame, and hence save the overhead due to a new contention.
- The second idea is to merge the two consecutive frames in the "uplink" direction (the acknowledgment and the voice frames) into one single transmission by using the second voice frame to carry the ACK information in the opposite direction. This saves the channel time devoted to the transmission of the ACK which, given that the length of a voice frame is relatively similar to that of an ACK frame, improves very substantially the efficiency with low implementation costs.
- The third idea is to omit the last ACK frame. This is based on the argument that the probability that a frame suffers from a failure in the uplink direction is much smaller than in downlink, as the uplink frame is sent after a SIFS, and hence is protected from collisions, which is the most common source for failures in 802.11. To ensure that uplink frames are not lost even in the unusual case that they suffer failures, we have designed the mechanism described in Section 3.3, which retransmits failed uplink transmissions without requiring to acknowledge these frames.

Based on the above ideas, we propose to modify the operation of the protocol for the case of voice traffic as illustrated in the bottom part of Fig. 1. Our proposal, as we formally specify in Section 3.3, introduces two small changes to the standard operation of the STA and the AP: (i) assuming that there is a pending voice frame in the output queue of the STA (this is further discussed in Section 3.4), a $SIFS$ time after the reception of a voice frame from the AP, the standard ACK reply is replaced by a different

frame, which includes both the control information corresponding to the ACK and the voice frame from the STA; (ii) upon the reception of this new frame from the STA, the AP proceeds as if a standard ACK was received (thus clearing the output queue) and it processes the voice data carried by the frame, but does not acknowledge the reception of the voice frame.

As Fig. 1 illustrates, these modifications save approximately the time spent in the transmission of the two ACK frames ($2T_{ack}$, where $T_{ack} = SIFS + T_p + ACK/R_c$). According to the values in the last column of Table 1, this results in savings between 20% and 45% over the standard exchange T_{std} (depending on the MCS used). Furthermore, an additional improvement is that, by means of this scheme, the channel contention is largely reduced as voice stations are "polled" by the AP, thus yielding additional efficiency improvements as detailed in the performance evaluation of Section 6.[2]

Note that the VoIPiggy design relies on the assumption that voice applications do not implement silence suppression. We argue that this is not a limiting assumption in realistic scenarios, since the most widely used voice applications nowadays, such as e.g. Skype or Google Hangouts,[3] do not implement silence suppression [20]. As reported in [21], this results in better voice quality and maintains UDP bindings at the NAT, among other advantages. In Section 6.5, we evaluate Skype and Google Hangouts and confirm experimentally that they do not use silence suppression.

3.3 Required Modifications

Next, we formalize the operation of our proposal by describing the modifications introduced in the standard operation of the AP and the stations.

Changes to the AP. Given that the mechanism is triggered by the voice frame transmitted by the AP, we give the highest priority to the delivery of this type of traffic by setting the contention window configuration for the voice queue to $CW^{VO}_{\min} = CW^{VO}_{\max} = 2$, which is the minimum allowed by the standard. This configuration has been chosen to ensure that the stringent delay requirements of voice traffic are satisfied.[4] Another modification is the processing of the new piggybacked frames upon receiving them. To this aim, the AP has to identify the piggybacked frame. If the frame is received and the piggybacked data can be decoded, we proceed as if it were an ACK, removing the voice frame from the NIC queue and informing the upper layers, *without* triggering the transmission of an ACK. If the AP either does not receive the uplink frame carrying the ACK or cannot decode the voice data piggybacked to the frame, then it acts as it had not received the ACK and retransmits the downlink frame (following the standard backoff procedure to retransmit frames).

Changes to the STA. To take advantage of the savings introduced by the VoIPiggy exchange, the station needs to have a voice frame in the output queue, ready to be piggybacked. To this aim, we enforce that, when a station has a voice frame to transmit, it does not transmit it immediately but waits for a frame in the downlink direction. In order to avoid waiting too much time, which would harm voice performance, we limit the maximum waiting time by δ. The value of δ adapts to the voice codec used in order to minimize the delay suffered by outgoing frames, as we describe next. Upon the reception of a voice frame from the AP, the station has to forge a frame of type `Data+ACK` and transmit it after a $SIFS$. In case of an uplink transmission failure, following the AP operation described above, the AP retransmits the downlink frame; the STA identifies such a retransmission as it has the Retry flag in the header set, and replies by retransmitting the failed frame (piggybacked to an ACK).

With the above modifications, the operation of VoIPiggy resembles that of a *polling scheme*: the voice frames sent by the AP play the role of *poll* frames, to which the stations respond (when polled) by sending the voice frames in the opposite direction (to the AP). This similarity has already been mentioned in Section 2, where we report the main differences between VoIPiggy and a polling scheme such as HCCA. In the rest of the paper, we refer to this way of operating as *polling*-like operation.

We illustrate the changes introduced by VoIPiggy over the standard IEEE 802.11 state machine in Fig. 2. Modifications to standard operation of 802.11 are represented by means of shadowed circles and dashed or dotted lines, depending on whether they affect the AP (dashed lines) or the station (dotted lines). A detailed description of the implementation of the above modifications in our platform, performed at the driver and firmware levels, is provided in Section 5.

3.4 Setting of δ

In the following, we design the algorithm to compute δ. This parameter determines the maximum time a station holds a voice frame while waiting for a frame in the downlink direction. Following the explanation provided above, we want δ to be, on the one hand, large enough to make sure that we piggyback as many frames as possible and, on the other hand, as small as possible while meeting this condition, to avoid introducing unnecessary delays.

Ideally, in voice traffic the inter-arrival time between two consecutive frames is fixed, equal to a

2. The improvement on channel contention results from the fact that voice stations do not contend for the channel but piggyback their packets to ACK frames. Note that if we had designed VoIPiggy to piggyback the packets of the AP rather those of the stations, voice stations would contend for the channel and hence we would not see such improvement in channel contention.

3. *Google Hangouts:* http://www.google.com/+/learnmore/hangouts/

4. Even though we are prioritizing voice traffic, VoIPiggy does not only improve the performance of voice but also of data traffic, as shown by the results of Section 6.

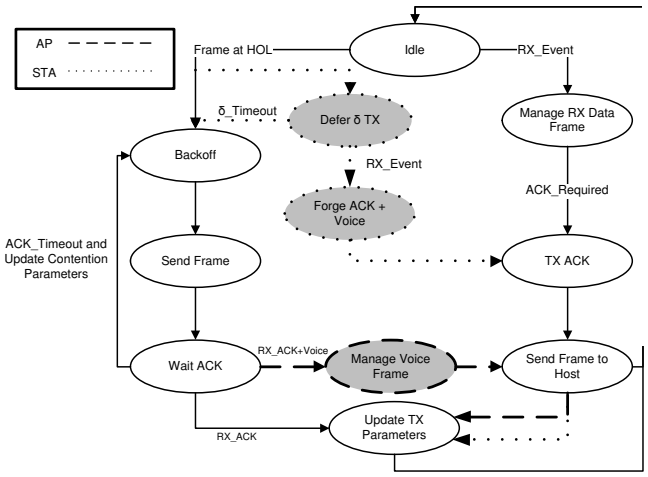


Fig. 2: Changes introduced in the standard IEEE 802.11 MAC state machine by VoIPiggy. Dashed and dotted lines represent new state transitions, while shadowed circles represent new states.

constant term T (typically, $T = 20\,\text{ms}$). In this ideal case, it would be sufficient to set $\delta = T$ to make sure that the above objective is met since, with this setting, even in the very *worst case* that a voice frame is generated right after receiving a frame from the AP, it will be held long enough to wait until the next frame of the AP. However, in reality the spacing between frames from the AP may not be perfectly regular as there may be some deviations caused by, e.g., the backoff process. To account for these deviations, we follow a similar approach to the "Algorithm 4" proposed in [22].

Let t_i be the time we receive the i^{th} frame from the AP, and T_i be the estimation of the *average* inter-arrival time up to this frame. The computation of T_i is performed as:

$$T_i = (1-\alpha)T_{i-1} + \alpha(t_i - t_{i-1}), \tag{2}$$

where α is a fixed constant (following [22], we set $\alpha = 0.125$). Furthermore, we also estimate the average deviation of the inter-arrival time from the estimated average as follows:

$$v_i = (1-\alpha)v_{i-1} + \alpha|t_i - t_{i-1} - T_i| \tag{3}$$

Once we have calculated the above estimates, we set δ_i (the value of δ at the i^{th} frame) as:

$$\delta_i = T_i + Kv_i \tag{4}$$

where K is a positive constant (following [22], we set $K = 4$). The purpose of the Kv_i term is to set δ large enough to absorb the deviations suffered by the inter-arrival times. Note that, in case these deviations are negligible, then the term v_i will be very small and we will have $\delta_i \approx T$, which corresponds to the ideal case mentioned above. The effectiveness of the above algorithm for the setting of δ is experimentally evaluated in Section 6.

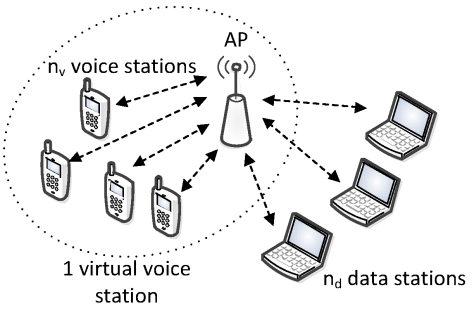


Fig. 3: Scenario: WLAN with n_v voice stations and n_d data stations.

4 Performance Analysis

We next analyze the performance of the VoIPiggy mechanism presented in the previous section in terms of throughput performance, capacity region and delay of voice traffic.

4.1 Throughput performance

We consider a WLAN scenario with one AP and n_v VoIP associated clients, all of them with the VoIPiggy functionality enabled. Each VoIP client is connected to a remote client located outside the WLAN. We assume that the VoIP application generates packets of fixed size l_v. In the same WLAN, we consider the presence of another set of n_d independent stations, each generating data frames of fixed length l_d.

As a result of our scheme to configure the parameter δ, we can assume safely that all frames from the n_v clients are piggybacked. Therefore, voice traffic is served by the AP in a *polling*-like way, with only one station (the AP) contending for the channel. Thus, we can model the voice activity in the WLAN as a single "virtual station" (see Fig. 3). We denote with τ_v the probability that this virtual station transmits in a *slot time* [23]. Similarly, we denote with τ_d the probability that a data station transmits in a slot time.

Let $p_{s,v}$ and $p_{s,d}$ be the probabilities that a successful transmission occurs in a slot time for a VoIP station and a data station, respectively, and let T_v and T_d be the corresponding slot lengths in these cases, which can be computed as:

$$T_v = DIFS + SIFS + 2T_p + \frac{H + ACK' + 2l_v}{R}, \tag{5}$$

$$T_d = DIFS + 2T_p + \frac{H + l_d}{R} + SIFS + \frac{ACK}{R_C}, \tag{6}$$

where ACK' is the length of the modified acknowledgment header for piggybacked voice packets. Throughout the article we will assume that the parameters {frame lengths, MCS} are such that $T_d > T_v$, which is the typical case, although the model could be easily extended to account for different settings of the parameters (following, e.g., our previous work [24]).

Similarly, we denote with $p_{c,d}$ and $p_{c,dv}$ the probabilities of having a collision involving data stations only and voice and data stations, respectively (note

that under our assumptions voice stations do not collide with each other), and with $T_{c,v}$ and $T_{c,dv}$ the corresponding duration of these collisions. Finally, let p_e be the probability of having an empty slot (the duration of such a slot time is a constant specified by the standard, T_e). With this notation, the average duration of a slot time can be expressed as:

$$T_{slot} = p_e T_e + p_{c,d} T_{c,d} + p_{c,dv} T_{c,dv} + p_{s,v} T_v + p_{s,d} T_d. \quad (7)$$

With the above, the throughput of a voice and a data station in the WLAN (denoted as R_v and R_d, respectively) can be computed as:

$$R_v = \frac{1}{n_v} \frac{p_{s,v}\, l_v}{T_{slot}}, \quad R_d = \frac{1}{n_d} \frac{p_{s,d}\, l_d}{T_{slot}}. \quad (8)$$

Assuming that all data stations and the virtual one are independent, the probabilities above can be computed as follows:

$$p_e = (1-\tau_v)(1-\tau_d)^{n_d}, \quad (9)$$

$$p_{c,dv} = \tau_v \left(1-(1-\tau_d)^{n_d}\right), \quad (10)$$

$$p_{c,d} = (1-\tau_v)\left(1-(1-\tau_d)^{n_d} - n_d \tau_d (1-\tau_d)^{n_d-1}\right), \quad (11)$$

$$p_{s,v} = \tau_v (1-\tau_d)^{n_d}, \quad (12)$$

$$p_{s,d} = n_d \tau_d (1-\tau_d)^{n_d-1}(1-\tau_v). \quad (13)$$

Finally, to compute the slot time durations, we use T_d and T_v as defined in (6) and (5), and we consider that the collision duration is determined by the longest frame exchange, which leads to:

$$T_{c,dv} = \max(T_v, T_d) = T_{c,d} = T_d. \quad (14)$$

With the above, we can compute the throughput obtained by each traffic type (voice and data) for a given scenario of n_v and n_d stations *if the probabilities τ_d and τ_v are known*. The remaining challenge is hence to determine the value of these probabilities, which we refer to as the *point of operation*. We next address this challenge.

4.2 Point of operation

Here, we compute the pair (τ_d, τ_v) given the transmission parameters of the WLAN, the input variables n_v, n_d, l_v, l_d and the traffic generation rates of a voice flow (r_v) and a data flow (r_d).

Our analysis follows the technique described in [24]. We define *saturation rate* as the rate that a station would obtain if it always had a packet ready for transmission (i.e., if it were constantly backlogged). Based on this definition, we can classify stations as *saturated* (when the traffic generation rate is above the saturation rate) or *non-saturated* (when the traffic generation rate is below the saturation rate). We first describe how to compute the transmission probabilities depending on whether the traffic type is saturated or not, and then address the general case.

Saturated stations. We first consider the case in which stations are saturated. Following [23], the transmission probability of a station can be computed based on its minimum contention window W, and its conditional collision probability p. For the case of voice traffic, given their backoff configuration, they always use the same contention window independent of the number of retransmissions, i.e., $W_v = CW_{min}^{VO} = CW_{max}^{VO}$, and thus we obtain the following result:

$$\tau_v = \frac{2}{1+W_v}. \quad (15)$$

For the case of data traffic, the conditional collision probability can be computed as the probability that at least another station transmits data or voice:

$$p_d = 1 - (1-\tau_d)^{n_d-1}(1-\tau_v); \quad (16)$$

the corresponding transmission probability is [23]:

$$\tau_d = \frac{2}{1 + W_d + W_d\, p_d \sum_{i=0}^{m-1} (2\, p_d)^i}, \quad (17)$$

where m is the maximum backoff stage and W_d is the CW_{min} used for the data traffic type.

The system (15)–(17) can be solved numerically to obtain the transmission probabilities (τ_d, τ_v), and based on these we can use (8) to compute the voice and data throughput under saturation conditions.

Non-saturated stations. Under non-saturation, we assume that all traffic generated is served, and therefore the following holds for the case of the AP (i.e., voice traffic):

$$\frac{1}{n_v} \frac{\tau_v (1-p_v)\, l_v}{T_{slot}} = r_v, \quad (18)$$

where p_v is the conditional collision probability for voice, given by the probability that at least one data station transmits:

$$p_v = 1 - (1-\tau_d)^{n_d}. \quad (19)$$

Similarly, for the case of a data station, we have:

$$\frac{\tau_d (1-p_d)\, l_d}{T_{slot}} = r_d, \quad (20)$$

where p_d is computed as in Eq. (16).

Therefore, once n_d, n_v, r_v and r_d are known, and assuming that both types of traffic are not saturated, τ_d and τ_v can be obtained by solving the system of equations (16), (18)–(20).

General case. We next combine the above analyses for the all-saturated and non-saturated cases to obtain the operational point of the WLAN in a generic scenario in which, depending on the offered load r_v and r_d, it could happen that both, none, or only one traffic type is saturated. In order to compute the point of operation, we proceed iteratively as follows.

1) We first assume that all stations are saturated and compute τ_v and τ_d by solving the system (15)–(17).
2) Using the transmission probabilities obtained, we compute the per-station throughput for each type of traffic, R_v and R_d, with (8).

3) If the obtained throughput is less than or equal to the traffic generation rate for both data and voice traffic, the analysis is terminated and the pair (τ_v, τ_d) determines the point of operation of the WLAN. Otherwise, there are three possible choices:
   a) Neither voice traffic nor data traffic are saturated (i.e., $R_v > r_v$ and $R_d > r_d$). In this case, the system of equations to solve is given by (16), and (18)–(20).
   b) Voice traffic is not saturated and data traffic is saturated (i.e., $R_v > r_v$ and $R_d < r_d$). In this case, the system of equations to solve is given by (16)–(19).
   c) Voice traffic is saturated and data traffic is not saturated (i.e., $R_v < r_v$ and $R_d > r_d$). In this case, the system of equations to solve is given by (15), (16), and (20).
4) We next solve the corresponding system of equations to obtain the probabilities τ_v and τ_d, computing again the throughputs R_v and R_d. If these throughputs meet the same conditions as the ones used to derive the corresponding τ_v and τ_d probabilities, the algorithm terminates. Otherwise, we go back to step 3.

4.3 Capacity region

Based on the above analysis of the point of operation, we now address the issue of computing the set of feasible allocations in a WLAN, i.e., the number of flows that can be supported without throughput losses, which we refer to as the "capacity region" of the WLAN. Our capacity region analysis assumes that all voice stations use the same codec, and that all data stations generate the same traffic, although these assumptions could be easily relaxed at the cost of a more complex derivation.

Voice-only scenarios. We first consider the simple case when there is only voice traffic in the WLAN. As explained above, we can assume that in this case VoIPiggy enforces a *polling*-like operation across voice stations. In these circumstances, all voice traffic can be served as long as the total arrival rate at the AP is below the maximum service rate R_v^*, which is computed from Eqs. (7)–(13) with $n_v = 1$ and $n_d = 0$:

$$R_v^* = \frac{\tau_v \, l_v}{(1-\tau_v) \, T_e + \tau_v \, T_v}, \tag{21}$$

where τ_v is computed via Eq. (15), i.e., in saturation conditions. Assuming a generation rate of r_v, the maximum number of conversations in a voice-only scenario, which we denote as n_v^*, can then be computed as:

$$n_v^* = \left\lfloor \frac{R_v^*}{r_v} \right\rfloor. \tag{22}$$

Voice and data scenarios. We next consider the general scenario in which there are voice and data stations. We say that a given set of (n_v, n_d) of voice and data stations is supported by the WLAN if all the traffic generated by the stations can be sent to the WLAN, i.e., both traffic types are not saturated and hence there are no losses. In this case, we say the pair (n_v, n_d) lies within the capacity region $\mathcal{C}(r_v, r_d)$ of the WLAN, i.e.,

$$(R_v = r_v, R_d = r_d) \Leftrightarrow (n_v, n_d) \in \mathcal{C}(r_v, r_d)$$

Based on our analytical model, one way to compute the convex hull of $\mathcal{C}$ is by performing a sweep on n_v from 0 to n_v^*, and for each value of n_v increase the number of data stations n_d until any of the two traffic types (voice or data) becomes saturated.[5]

4.4 Delay performance

The above analysis serves to determine if a set of voice and data flows is supported by the WLAN; however, in order to provide QoS guarantees to voice applications, more sophisticated models involving delay performance are required. To tackle this, we next analyze the delay performance of voice in a WLAN under the VoIPiggy mechanism. Our analysis focuses on the downlink direction, i.e., traffic from the AP to the STAs, as this is the "bottleneck" of the network (in the uplink direction, stations hold their frames up to δ, and hence the delay is bounded by a low value).

Voice-only scenario. We first analyze the delay performance of a WLAN serving n_v flows. Given that the minimum value for CW is used and there is no data traffic present, we can assume that service time for voice frames is almost constant and equal to T_v. In these conditions, our system is a single server queue with periodic arrival processes and deterministic service times, and can thus be modeled following [25]. With this model, the survivor function $F^{-1}(t)$ for the queuing delay in this scenario can be computed as:

$$F^{-1}(t) = \frac{P_{n_v-1}(t, t_v, T_v)}{t_v^{n_v-1}} \quad , \quad x \geq 0, \tag{23}$$

where t_v is the inter-arrival time of voice traffic, given by $t_v = l_v / r_v$, and $P_k(t, t_v, T_v)$ is computed as:

$$P_k(t, t_v, T_v) = \sum_{l=0}^{k-1} q_{k,l}(t, T_v)(t_v - kT_v + t)^l, \tag{24}$$

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

$$q_{0,l}(t, T_v) = 0, \tag{25}$$

$$q_{k,0}(t, T_v) = \left((kT_v - t)^+\right)^k, \tag{26}$$

$$q_{k,l}(t, T_v) = \frac{k}{l} \sum_{j=l-1}^{k-2} \binom{j}{l-1} T_v^{j-l+1} q_{k-1,j}(t, T_v), \tag{27}$$

5. Although we have considered that data traffic is generated at a finite rate r_d, following a similar procedure we could compute, e.g., the maximum rate that one data station could get in presence of n_v voice flows.

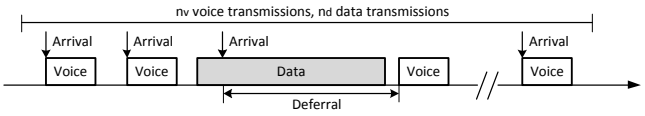


Fig. 4: Channel deferral by the AP in presence of data traffic.

where $y^+ = \max(0, y)$. From the above, the cumulative distribution function (CDF) of the queuing delay can be obtained as $F(t) = 1 - F^{-1}(t)$.

The above provides the queuing delay suffered by a frame; to obtain the total delay, we need to add the transmission time of a voice frame from the AP to the STA, which is given by

$$T'_v = DIFS + T_p + \frac{H + l_v}{R}. \quad (28)$$

Note that, in contrast to T_v, the above transmission time only accounts for the time elapsed until the voice frame reaches the AP and hence includes neither the ACK nor the piggybacked voice frame. The addition of the queuing delay and transmission time leads to the following CDF for the total delay D of the delivery of downlink voice frames, which is a *shifted* version of $F(t)$:

$$F_D(t) = F(t - T'_v), \quad t \geq T'_v. \quad (29)$$

Once we have obtained the CDF for the total delay, we can obtain its numerical derivative $f_D(t)$, and from this it is straightforward to obtain any performance figure for the downlink traffic in the voice-only scenario, e.g., average delay, 95-percentile or standard deviation.

Voice and data scenario. We next focus on the case when data traffic is also present in the WLAN. In this case, the impact of data traffic is twofold: not only voice transmission from the AP might collide with data transmissions, but also the AP has to perform channel deferral whenever it senses the medium and detects that it is busy.

In [26], it has been observed that channel deferral dominates the performance in 802.11 wireless networks, while collisions have a much smaller impact. With VoIPiggy, the impact of collisions is even smaller, since there are fewer flows contending for channel access. Based on this, in our analysis we neglect the impact of collisions and assume that a voice frame from the AP is either transmitted immediately, or it has to wait until an ongoing data transmission has finalized (i.e., for a residual time), as shown by Fig. 4. By denoting the probability of deferral by p_{def}, the CDF of the total delay can be expressed as:

$$F_D(t) = (1 - p_{\text{def}})U(t - T_v) + p_{\text{def}} F_D^R(t - T_v), \quad (30)$$

where $U()$ is the unit step function, and $F_D^R(t)$ is the CDF of the residual time of a data transmission. Given that these are of fixed length, if we assume that the residual time follows a uniform distribution, we can express $F_D^R(t)$ as:

$$F_D^R(t) = F_{\mathcal{U}}(t, T_d + t), \quad (31)$$

where $F_{\mathcal{U}}(0, T_d)$ is the CDF of a random variable uniformly distributed between 0 and T_d.

The pending challenge is to compute p_{def}. Our key approximation to compute this is to assume that whenever a voice frame is preceded by a data frame, it has to wait for the data transmission to end, as depicted in Fig. 4. Thus, the probability of channel deferral is equal to the probability that the previous transmission corresponds to a data frame. Given that on average in a given period of time T there are n_v voice transmissions and n_d data transmissions, if we consider a tagged voice transmission out of the n_v, this probability can be expressed as:

$$p_{\text{def}} = \frac{n_d}{n_d + n_v - 1}. \quad (32)$$

which completes the model for the delay performance in a mixed scenario.

5 Implementation Details

In this section we detail the implementation of the most relevant features of VoIPiggy, which is based on Alix 2d2 devices from PC Engine.[6] The implementation of the 802.11 stack for the chosen platform is composed of three main software/firmware modules: (i) the `mac80211` framework that takes care of the high-layer operations; (ii) the device `Driver`, which is a wrapper between the internal buffers and the physical device; and (iii) the `Firmware` of the device, which implements the internal logic that controls time-critical operations such as, e.g., packet retransmissions, which cannot be performed at neither kernel nor application level, due to the unpredictable delay introduced by the buses when crossing the protocol stack. For the communication between the firmware and driver, we rely on the block of data that the driver attaches to the structure that contains each packet to enable per-packet configurations, and extend it to pass information to the firmware. In the following, we describe the required modifications to the device firmware and to the Linux kernel.

5.1 Firmware Modifications

Some of the functions of the VoIPiggy mechanism, as they are very time critical, need to be implemented at the firmware level. To this end we built our system on `OpenFWWF`,[7] an "Open source FirmWare for WiFi networks" that supports customization of the device internal operations and has been used in the past to introduce extensions to the 802.11 default behavior [27], [28]. The use of this firmware enables modifying the protocol state machine by reacting to some

6. *PC Engines*: http://www.pcengines.ch/
7. *OpenFWWF*: http://www.ing.unibs.it/openfwwf/

conditions, and grants access to key internal modules of the device.

In order to support VoIPiggy at the **station** side, we first have to hold outgoing voice frames, which are marked by the driver as we describe next, up to a maximum time δ. This is done by loading, upon the arrival of a voice frame, the time synchronization register `SPR_TSF_WORD0` with the δ value computed by the driver, and holding the frame until the register reaches zero or a voice frame arrives. Next, we have to modify the interrupt request triggered upon the reception of a frame. More specifically, we implement a new procedure, `VOICE_RX.IRQ`, which is triggered when a voice frame from the AP arrives and the head-of-line frame is a voice frame. This procedure forges the VoIPiggy frame, which extends the default ACK to include the voice frame and the sender MAC addresses (updating the packet length and PLCP accordingly). After transmitting the voice frame, we still hold it in case it has to be retransmitted. This happens when a voice frame with the retry flag set is received from the AP. The frame is held until the AP transmits a new frame (with the retry flag unset) or the application generates a new voice packet.

At the **Access Point**, the main modification at the firmware level is the ability to identify and process the *voipiggied* frames received from the stations. For simplicity, we do this by comparing the length of the received frame, which is provided in the `SPR_RXE_FRAMELEN` register, against the length of a legacy ACK. If longer, and the piggybacked data is correctly decoded, the complete frame is passed to the driver for its processing, as we describe in the next section; otherwise, the default ARQ procedure is triggered.

5.2 Driver Modifications

The less time critical functions were implemented at the driver level in the kernel, in the C programming language.

At the **station** side, when the driver identifies a frame as VoIP,[8] it marks it as suitable for piggybacking. This is done by modifying the `b43_generate_txhdr()` function of the driver to extend the `b43_txhdr` data structure, in order to include a bit that we denote as `PIGGYBACK_ENABLED`. We also pre-compute in this function some variables (e.g., PLCP header) to speed-up the forging of the piggybacked frame in the firmware.

At the **AP** side, we add a *hook* in the driver to process the "long" ACK frames placed in the `skb_buff` buffer. This processing consists on first forging a regular data frame for the voice payload, by inserting the missing sender and receiver addresses, the LLC header, etc. Then, the frame is passed to the `mac80211` module (which remains oblivious to the VoIPiggy operation) via the `b43_rx()` function.

Finally, to compute the δ parameter, and also for debugging and monitoring purposes, we extended the `b43_wldev` structure that contains various transmission and reception statistics (e.g., total number of frames, transmission attempts) to include those related to the operation of VoIPiggy (e.g., inter-arrival time, number of *voipiggied* frames).

6 Performance Evaluation

In this section we first describe the testbed used in our experiments. Then, we evaluate the performance of VoIPiggy under a large number of different scenarios, in terms of number of stations, data traffic and MCS used. Throughout our experiments we compare (whenever possible) the resulting performance against that obtained with the standard channel access, in which voice traffic and data traffic are configured with the recommended CW parameters of the VO and BK queues of the EDCA standard, respectively (denoted as "EDCA").

6.1 Testbed Description

Our testbed consists of 30 Alix 2d2 devices acting as wireless stations, and one desktop machine acting as AP. The AP uses a 7 dBi omnidirectional antenna located in the center of the testbed. The stations are deployed around this antenna and equipped with 2 dBi omnidirectional antennae. All nodes use a transmission power of 20 dBm. Given that the 2.4 GHz band is well populated in our testbed, we took great care in performing the experiments when the traffic activity was low.

We use `mgen`[9] as traffic generator, to emulate the behavior of standard voice codecs and data transfers. This tool supports the computation of one-way delay figures by inserting timestamps in all packets. As this requires that nodes are synchronized, we run the *PTP daemon*[10] over the Ethernet interfaces of the nodes, achieving 0.1 ms synchronization accuracy. The wired interface is also used to perform other control and management plane operations, such as e.g. remote execution of tests or retrieval of the results for off-line processing, so that they do not interfere with the actual measurement data on the wireless medium.

We emulate the traffic generation of voice applications by running independent instances of the `mgen` tool, each transmitting either a 60-byte or a 160-byte voice frame every $t_v = 20$ ms, thus emulating the behavior of the G.711 and G726 codecs, respectively. For the case of UDP data traffic, `mgen` is configured

8. There are various alternatives to detect if the frame is VoIP, such as: by having the application set the DSCP code, by looking at the *Payload Type* field from a RTP header, at the frame length, etc. In this paper, for simplicity we do it by checking if the destination port matches with a predefined value.

9. *MGEN*: http://cs.itd.nrl.navy.mil/work/mgen/

10. *Precision Time Protocol*: http://ptpd.sourceforge.net/

TABLE 2: Maximum number of conversations supported in a voice-only scenario.

Voice codec	MCS (Mbps)	n_v experimental EDCA	VoIPiggy	Gain	n_v model VoIPiggy	Gain
G. 711	2	5	9	80%	9	80%
	5.5	10	18	80%	18	80%
	11	12	26	116%	26	116%
	6	19	29	53%	29	53%
	9	24	≥ 30	-	49	71%
	12	26	≥ 30	-	52	100%
	54	≥ 30	≥ 30	-	125	-
G. 726	2	8	14	75%	14	75%
	5.5	10	26	160%	26	160%
	11	12	≥ 30	-	33	175%
	6	23	≥ 30	-	49	113%
	9	26	≥ 30	-	66	153%
	12	≥ 30	≥ 30	-	79	-
	54	≥ 30	≥ 30	-	154	-

to send 1453 byte frames every 23 ms, i.e., an approximate rate of 500 kbps.

6.2 Capacity Region

Voice-only scenarios. We start our performance evaluation with a scenario in which only voice traffic is present. To this aim, we analyze the maximum number of conversations supported in the WLAN, for different configurations of the voice codec and MCS used. More specifically, we increase the number of stations n_v from 1 until $n_v^* + 1$ is reached, which is the minimum number of voice stations that leads to traffic losses. In order to have good statistical guarantees, for each configuration of n_v we repeat the test, each of them run for 30 s, 5 times and only increase the number of conversations if none of the 5 experiments suffers from traffic losses. We also obtain, following a similar procedure, the maximum number of voice conversations that can be supported with the standard EDCA configuration. We present the obtained results in Table 2, in which we also provide the value predicted by our analytical model of Section 4.3, and the resulting improvements in the maximum number of conversations of VoIPiggy over EDCA (columns "Gain"). Note that we mark as ≥ 30 those experiments in which the number of supported conversations reached the maximum number of nodes in our testbed.

There are two main conclusions that can be drawn from the table. First, the experimental results show a perfect match with those from the theoretical model for all the cases that can be evaluated with the number of nodes available in the testbed (i.e., $n_v^* < 30$). Second, the results show that VoIPiggy significantly increases the efficiency of WLANs in the presence of voice traffic. Indeed, as compared to the standard recommended configuration, the improvements range between 53% and 175%, which is a dramatic gain at the relatively small cost of implementation.

The above confirms that, in voice-only scenarios, VoIPiggy is able to boost the capacity of the network, which is a consequence of reducing the inefficiency of the standard protocol when short packets are sent in both directions (as discussed in Section 3.1). We next analyze how the efficiency improvements introduced by VoIPiggy in the delivery of voice also increase the capacity of WLANs when data traffic is present.

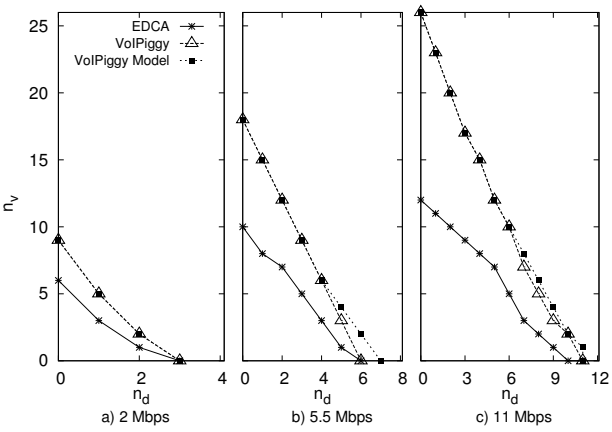


Fig. 5: Capacity region of an 802.11b scenario with voice and data stations.

Voice and data scenarios. We next analyze the case of a mixed scenario with n_v bi-directional voice conversations and n_d uplink data flows, in order to experimentally assess the capacity region $\mathcal{C}$ of VoIPiggy and to validate our analytical model presented in Section 4.3. To this aim, we perform a sweep on the number of voice stations n_v (from 0 to n_v^*) and, for each value of n_v, we sweep on the number of data stations n_d, starting from 1 and increasing it while the throughput demands of voice and data flows are satisfied. Like in the previous case, we repeat each test 5 times, and only increase the number of stations if no significant losses were measured.[11]

We start our measurements for the case of 802.11b, assuming the G. 711 voice codec and data stations transmitting at $r_d = 500$ kbps, and the following values for the MCS={2, 5.5, 11} Mbps. For each configuration of the transmission rate, we experimentally measure the capacity region for the EDCA configuration and for VoIPiggy, also providing the theoretical values in this case ('VoIPiggy Model'). The results are depicted in Fig. 5. Experiments show that when there are no voice conversations in the WLAN there is no increase in the capacity, given that no piggybacking is taking place. In contrast, the more voice flows are present, the larger the difference between the EDCA performance and the one achieved by means of VoIPiggy; indeed, for the MCS presented in the figure, the "area" of the capacity region is practically doubled with the use of VoIPiggy. Finally, it is also worth remarking the good match between experimental and

11. The capacity region figures are presented for the case when losses are below 1%. We have repeated the experiments for up to 10% losses, and have obtained very similar results.

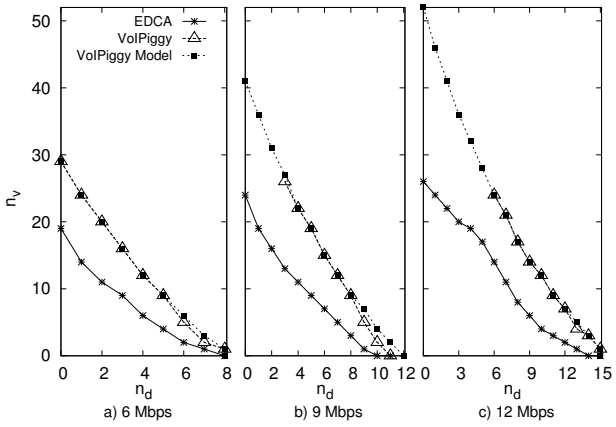


Fig. 6: Capacity region of an 802.11g scenario with voice and data stations.

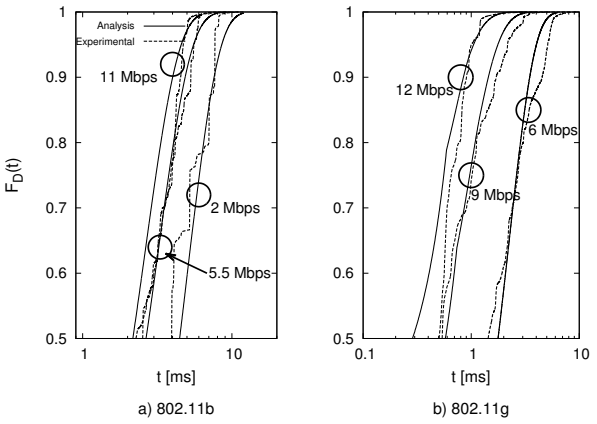


Fig. 7: CDF of the delay for voice-only scenarios.

analytical results.

We proceed similarly for the case of 802.11g, with the 6, 9, and 12 Mbps MCSs. The results are depicted in Fig. 6. Given that the MCSs considered are more efficient when dealing with voice traffic, the performance improvements are slightly below the ones presented in the previous case, but they are still very significant. As the capacity region $\mathcal{C}$ is notably large in this case, we can only confirm the accuracy of the model when $n_v + n_d \leq 30$, but we also show in the figure the analytical results for topologies larger than our testbed, which helps to properly illustrate the large improvements due to VoIPiggy.

Based on these results, we conclude that VoIPiggy is able to almost double the capacity region of 802.11 WLANs, i.e., the number of stations that can be supported with no throughput loss, as confirmed both by analytical and experimental results. We next analyze the delay performance when using VoIPiggy for scenarios at the boundary of the capacity region.

6.3 Delay Performance

The results of the previous section validate the performance improvements of VoIPiggy and the accuracy of the analytical model for the capacity region. Such results can be used, e.g., to perform call admission control (CAC) decisions if the performance metric of interest is throughput. Conversely, if we are interested in the *quality* of the service received by voice conversations, it is important to characterize delay performance in addition to throughput. To this end, we next analyze the CDF of the downlink delay using VoIPiggy.

Voice-only scenarios. We first consider the case of a voice-only scenario. We consider a scenario with the maximum number of conversations supported with VoIPiggy for different MCSs (i.e., n_v^*), all of them using the G.711 voice codec, and compute the CDF of the service delay. Each experiment is run for 60 s, and we perform 5 repetitions to confirm that results did not vary much across experiments. The results are depicted in Fig. 7 for the case of 802.11b (left) and 802.11g (right). We use dotted lines to represent the experimental results, and solid lines for the numerical figures computed using the analytical model presented in Section 4.4.

We observe a good match between analytical and experimental values. Furthermore, results confirm the good service provided to voice traffic: for the case of 802.11b it only reaches 10 ms for the lowest MCS, while for the case of 802.11g delay figures are well below this value. Note that, in all considered scenarios, the experiments are run with the maximum number of voice conversations supported by the WLAN, which are the most stringent conditions, and therefore they provide worst case results for the delay.

Voice and data scenarios. We next consider the case of the WLAN with voice and data stations. Like in the previous case, we compute the CDF of the total delay of voice traffic in the downlink direction. Results are given in Fig. 8. For each MCS considered in the figure, we select one specific point in the capacity region $\mathcal{C}$, which is is the pair (n_v, n_d) reported in the figure below the MCS (e.g., for the case of 11 Mbps, the scenario is $n_v = 6$, $n_d = 8$).

Our results show that, as expected, the activity of data stations worsens the performance of voice traffic, even when the number of conversations is smaller than n_v^*. The results also confirm that channel deferral because of data transmissions has a significant impact, while the impact of collisions (responsible for the rarely occurring long delays, which appear in the CDF's tails) is practically negligible. In any case, and despite of the increased activity in the channel, the CDFs show that voice traffic still receives a good service, since delay figures are in the range of a few ms. There results thus confirm that VoIPiggy is able to make room for more voice conversations in the WLAN while maintaining a good service level.

In order to gain further insight into the delay performance of a WLAN with voice and data stations,

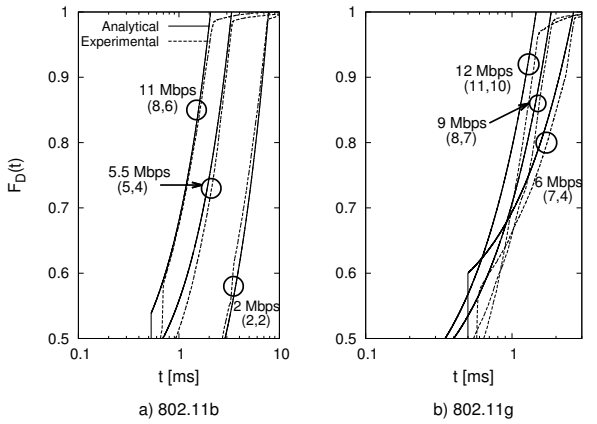


Fig. 8: CDF of the voice delay for voice and data scenarios.

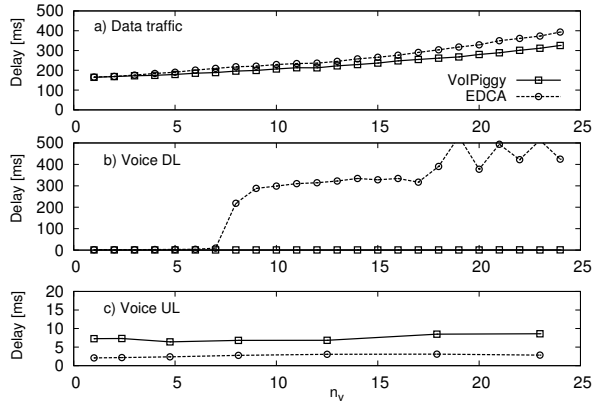


Fig. 9: Delay for voice and data traffic.

we consider a scenario with n_v voice conversations and one station sending data to the AP in saturation, and measure the average delay of the data and voice traffic (in the uplink and downlink). The results, given in Fig. 9, show that not only VoIPiggy improves the delay performance of voice traffic, but it also improves the delay performance of data traffic. Therefore, even though VoIPiggy gives higher priority to voice traffic, as a result of making a more efficient use of the WLAN resources, it also improves the performance of data traffic in terms of delay (as shown in this section) and throughput (as shown in the next one).

6.4 TCP traffic

In the previous scenarios we have assumed, for the case of the data stations, UDP uplink traffic to the AP. In this section we analyze the performance of VoIPiggy when TCP is used instead of UDP. To this aim, we set up a WLAN scenario in which n_v voice conversations are present and one data station is receiving TCP traffic from the AP.[12] For each scenario, we run the experiment for 60 s and compute the average

12. We also performed experiments with the data station sending TCP traffic, obtaining very similar results to the ones reported here.

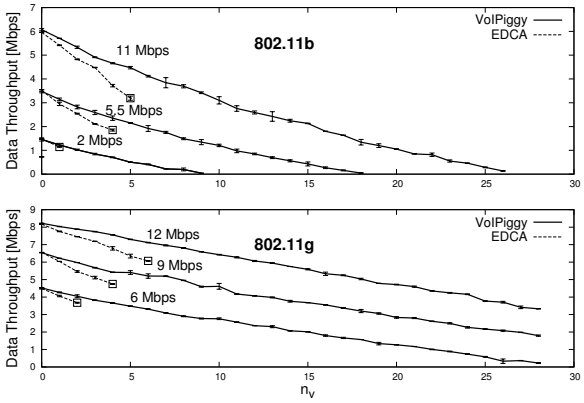


Fig. 10: TCP throughput with n_v voice conversations (the figure only shows those points for which voice traffic does not suffer losses above 1%).

TCP throughput obtained, repeating the experiment 5 times. We present the obtained results in Fig. 10, with VoIPiggy (solid line) and EDCA (dashed line), for an increasing number of conversations, until n_v reaches the maximum number of stations (29, which is reached with VoIPiggy in the 802.11g case) or voice losses are above 1% (this only happens with EDCA).

There are several conclusions that can be drawn from the figure. First, the use of VoIPiggy effectively protects the performance of voice traffic, as the maximum number of voice conversations that can be supported reaches the values obtained in voice-only scenarios (n_v^*). In contrast, EDCA only supports a much smaller number of conversations than in the previous case, due to the increased aggressiveness of the data traffic. Second, TCP traffic obtains a larger throughput with VoIPiggy than with EDCA for all values of n_v, which confirms the previous observation that VoIPiggy improves not only voice performance but also data performance. Third, since VoIPiggy gives higher priority to voice stations, if the number of voice stations reaches its maximum value, data stations starve. However, in case we wanted to ensure a minimum throughput for data traffic, our analysis of Section 4 could be used to apply admission control and thus limit the number of voice stations in the WLAN.

6.5 Validation of the algorithm to compute δ

Finally, we validate the ability of the algorithm presented in Section 3.4 to estimate the inter-arrival variability of voice frames, required to hold waiting frames long enough until a frame from the AP arrives while avoiding unnecessary delays. To this aim, we use two Alix devices to set up a WLAN scenario consisting in one AP and one station, and compare the performance in terms of percentage of piggybacked frames for (i) a fixed δ setting, and (ii) the use of our adaptive algorithm.

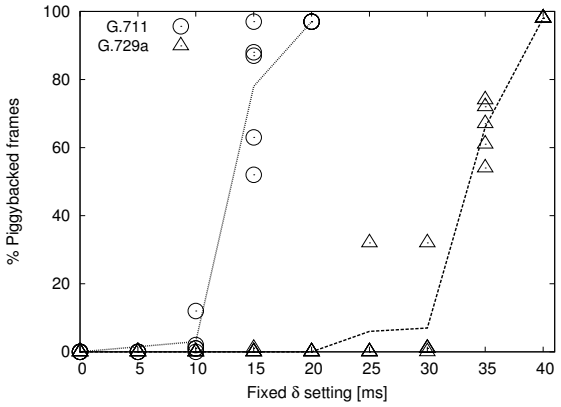


Fig. 11: Percentage of piggybacked frames for a fixed configuration of δ.

TABLE 3: Performance of the algorithm to compute δ

Voice codec	% piggybacked avg. $\pm$ σ	T_i (ms)	v_i (ms)
G.711	99.22 $\pm$ 0.90 %	19.8	0.58
G.729a	99.12 $\pm$ 0.73 %	39.4	1.10
Skype	99.61 $\pm$ 0.17 %	30.3	5.76

We perform our evaluation using `mgen` to emulate the behavior of the G.711 and G.729a codecs, which generates 160 B frames every 20 ms and 40 B frames every 40 ms, respectively. We first analyze the case of a static setting of δ, for different values of this parameter. More specifically, we sweep from 0 to 40 ms, and for each δ value we compute the percentage of piggybacked frames during a 60 s experiment, running 5 repetitions for each one. The results are depicted in Fig. 11 using one point per repetition (for ease of visualization we use lines to represent the average of the experiments). They confirm that, depending on the codec used, a different value of δ is required to maximize the number of piggybacked frames (i.e., $\delta \geq 20$ ms and $\delta \geq 40$ ms for G.711 and G.729a, respectively), and therefore a fixed setting of δ would require manual tuning.

We next analyze the performance of the algorithm to dynamically adapt δ, following a similar procedure as in the previous case (i.e., 5 experiments of 60 s each). The results, summarized in Table 3, show the average percentage of piggybacked frames of the 5 experiments and its standard deviation (σ), along with the average values of the T_i and v_i parameters of our algorithm. The numerical figures confirm the ability of the algorithm to adapt to the variability of each frame arrival process, as practically all frames are piggybacked without using overly large values for δ.

To confirm that our algorithm also works with real voice applications, we also evaluated its performance with Skype. The results, given also in Table 3, show that practically all frames are also piggybacked in this case, which confirms that VoIPiggy works with Skype. This also confirms that Skype does not use silence suppression, as otherwise many of the uplink frames would not be piggybacked. In addition, we also confirmed experimentally that Google Hangouts does not implement silence suppression either.

6.6 Reliability of VoIPiggy

The design of VoIPiggy aims to achieve reliable delivery of voice frames both downlink (provided by the standard operation) and uplink (provided by the design of VoIPiggy). To evaluate the reliability of VoIPiggy, we have run several experiments and measured the the number of frames that had to be retransmitted as well as the total number of lost frames. For all experiments, out off (approx.) 1500 frames, about 50 had to be retransmitted (less than 5%), and no one was lost. We conclude that VoIPiggy is able to reliably deliver frames in both directions.

7 Conclusions

In this paper, we have designed, implemented and evaluated VoIPiggy, a mechanism to improve the efficiency of MAC operation when voice traffic is present in 802.11 WLANs. In contrast to legacy operation, which incurs a very large overhead and wastes substantial time in in contention, 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. In this way, not only the capacity to support voice traffic in the WLAN is boosted, but also data traffic benefits from the increased efficiency in the delivery of VoIP.

We have described the small set of modifications required to implement VoIPiggy, which are supported by existing COTS devices, and provided an analytical model to predict its performance, in terms of capacity region and delay. To assess the performance improvements of VoIPiggy and validate its modeling, we have deployed a large-scale testbed consisting of 30 devices. Through extensive performance evaluation we have shown that our mechanism dramatically improves performance—which provides a strong empirical support for the adoption of VoIPiggy—as well as we demonstrated the good accuracy of our analytical model.

References

[1] P. Salvador, F. Gringoli, V. Mancuso, P. Serrano, A. Mannocci, and A. Banchs, "VoIPiggy: Implementation and evaluation of a mechanism to boost voice capacity in 802.11 WLANs," in *Proceedings of IEEE INFOCOM'12*, March 2012, pp. 2931–2935.

[2] *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*, IEEE Std. 802.11, 2012.

[3] *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications - Amendment 8: Medium Access Control (MAC) Quality of Service Enhancements*, IEEE Amendment 802.11e, 2005.

[4] A. Banchs, P. Serrano, and L. Vollero, "Reducing the Impact of Legacy Stations on Voice Traffic in 802.11e EDCA WLANs," *IEEE Communications Letters*, vol. 11, no. 4, April 2007.

[5] S. Garg and M. Kappes, "Can I add a VoIP call?" in *Proc. of IEEE ICC'03*, Anchorage, Alaska, USA, May 2003.

[6] ——, "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] 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'08*, Stockholm, Sweden, Jun 2008.

[8] G. Hanley, S. Murphy, and L. Murphy, "Adapting WLAN MAC Parameters to Enhance VoIP Call Capacity," in *Proceedings of MSWiM'05*, October 2005, pp. 250–254.

[9] 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*, December 2003.

[10] *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 - Amendment 5: Enhancements for Higher Throughput*, IEEE Std. 802.11n, 2009.

[11] M. Ozdemir, D. Gu, A. McDonald, and J. Zhang, "Enhancing MAC Performance with a Reverse Direction Protocol for High-Capacity Wireless LANs," in *Proceedings of Vehicular Technology Conference, 2006. VTC-2006 IEEE 64th*, September 2006.

[12] Hsiang-Ho Lin and Chih-Yu Wang and Hung-Yu Wei, "Improving Online Game Performance over IEEE 802.11n Networks," in *Proceedings of NetGames'10*, November 2010.

[13] M. Rashid, E. Hossain, and V. Bhargava, "HCCA Scheduler Design for Guaranteed QoS in IEEE 802.11e Based WLANs," in *Proceedings of IEEE WCNC'07*, 2007, pp. 1538–1543.

[14] H.-J. Lee, J.-H. Kim, and S. Cho, "A Novel Piggyback Selection Scheme in IEEE 802.11e HCCA," in *Proc. of IEEE ICC'07*, 2007.

[15] C. Casetti, C. F. Chiasserini, M. Fiore, and M. Garetto, "Notes on the Inefficiency of 802.11e HCCA," in *Proceedings of IEEE VTC'05*, 2005.

[16] I. Tinnirello, G. Bianchi, P. Gallo, D. Garlisi, F. Giuliano, and F. Gringoli, "Wireless MAC Processors: Programming MAC Protocols on Commodity Hardware," in *Proceedings of IEEE INFOCOM'12*, March 2012, pp. 1269–1277.

[17] Lee, Hyun-Jin and Kim, Jae-Hyun and Cho, Sunghyun, "A Novel Piggyback Selection Scheme in IEEE 802.11e HCCA," in *Proc. of IEEE ICC'07*. IEEE, 2007, pp. 4529–4534.

[18] Y. Xiao, "IEEE 802.11 Performance Enhancement via Concatenation and Piggyback Mechanisms," *IEEE Transactions on Wireless Communications*, vol. 4, no. 5, pp. 2182–2192, 2005.

[19] P. Verkaik, Y. Agarwal, R. Gupta, and A. C. Snoeren, "Softspeak: Making VoIP Play Well in Existing 802.11 Deployments," in *Proceedings of NSDI'09*, 2009, pp. 409–422.

[20] S. A. Baset and H. G. Schulzrinne, "An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol," in *Proceedings of IEEE INFOCOM'06*, April 2006.

[21] Y.-C. Chang, K.-T. Chen, C.-C. Wu, and C.-L. Lei, "Inferring Speech Activity from Encrypted Skype Traffic," in *Proceedings of IEEE GLOBECOM'08*, 2008.

[22] R. Ramjee, J. Kurose, D. Towsley, and H. Schulzrinne, "Adaptive Playout Mechanisms for Packetized Audio Applications in Wide-Area Networks," in *Proceedings of IEEE INFOCOM'94*, June 1994, pp. 680–688.

[23] G. Bianchi, "Performance Analysis of the IEEE 802.11 Distributed Coordination Function," *IEEE Journal on Selected Areas in Communications*, vol. 18, no. 3, pp. 535–547, March 2000.

[24] P. Serrano, A. Banchs, P. Patras, and A. Azcorra, "Optimal Configuration of 802.11e EDCA for Real-Time and Data Traffic," *IEEE Transactions on Vehicular Technology*, vol. 59, no. 5, pp. 2511–2528, 2010.

[25] A. Eckberg Jr., "The Single Server Queue with Periodic Arrival Process and Deterministic Service Times," *IEEE Transactions on Communications*, vol. 27, no. 3, pp. 556–562, Mar 1979.

[26] A. Kashyap, U. Paul, and S. R. Das, "Deconstructing Interference Relations in WiFi Networks," in *Proceedings of IEEE SECON'10*, June 2010.

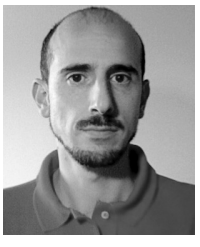
[27] B. Han, A. Schulman, F. Gringoli, N. Spring, B. Bhattacharjee, L. Nava, L. Ji, S. Lee, and R. Miller, "Maranello: Practical Partial Packet Recovery for 802.11," in *Proceedings of NSDI'10*, March 2010, pp. 205–218.

[28] 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'11*, Poland, June 2011.

Pablo Salvador received his B.Sc. in telecommunication engineering from Universidad Carlos III de Madrid (UC3M) in 2010, and his M.Sc. in telematics engineering from the same university in 2011. He was awarded as nationwide finalist of the Telematics Association for his B.Sc. project. Pablo is currently working on his Ph.D. at the Institute IMDEA Networks. His work focuses on performance evaluation of wireless networks.

Vincenzo Mancuso obtained his master degree in Electronics from University of Palermo, Italy, in 2001, and a PhD in Electronics, Computer Science and Telecommunications from the same University in 2005. He has been visiting scholar at the ECE Department of Rice University, Houston, Texas, and postdoc in the MAESTRO team at INRIA Sophia Antipolis, France. Since September 2010, he is with Institute IMDEA Networks.

Pablo Serrano received his degree in telecommunication engineering and his Ph.D. from Universidad Carlos III de Madrid (UC3M) in 2002 and 2006, respectively. He has been with the Telematics Department of UC3M since 2002, where he currently holds the position of associate professor. He serves on the Editorial Board of IEEE Communications Letters, and has served on the TPC of a number of international conferences.



Francesco Gringoli received his Laurea degree in Telecommunications Engineering from the University of Padua (Italy) in 1998 and his Ph.D. degree in Information Engineering from the University of Brescia (Italy) in 2002. Since 2005 he is Assistant Professor of Telecommunications at the Dept. of Information Engineering of the University of Brescia, Italy. He started the OpenFWWF Project in 2009.

Albert Banchs received his degree in telecommunications engineering from the Polytechnic University of Catalonia in 1997, and his Ph.D. degree from the same university in 2002. He has been with the University Carlos III of Madrid since 2003. Since 2009, he also has a double affiliation as Deputy Director of the institute IMDEA Networks. He has served on the TPC of a number of conferences and workshops,and has been TPC chair for European Wireless 2010, IEEE HotMESH 2010 and IEEE WoWMoM 2012. He is senior member of IEEE.