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

Pablo Salvador†‡, Francesco Gringoli†, Vincenzo Mancuso†‡, Pablo Serrano†, Andrea Mannocci†‡, Albert Banchs†‡

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 paper we propose a simple scheme to improve the efficiency of WLANs when voice traffic is present. The mechanism is based on piggybacking voice frames over the acknowledgments, which reduces both frame overheads and time spent in contentions. We evaluate its performance in a large-scale testbed consisting of 33 commercial off-the-shelf devices. The experimental results show dramatic performance improvements in both voice-only and mixed voice-and-data scenarios.

I. INTRODUCTION

IEEE 802.11 [1] 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, the legacy DCF operation results extremely inefficient and 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 prioritization mechanisms [2]. As a trade-off, the performance of data frame has to be reduced to sustain a decent level of quality for voice traffic.

The impact of protocol overhead on VoIP has been extensively researched in the literature. The authors of [3] and [4] 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 [5]. 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., [6] and [7].

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 [6] show that appropriate MAC tuning can improve capacity by 20% to 40%. Experiments reported in [8] confirm that commercial devices need non-trivial prioritization mechanisms in order to guarantee the quality of voice. Experiments in [4] show how voice conversations dramatically impair the performance of UDP data traffic since they reduce the available bandwidth.

Motivated by the limited efficiency of the standard operation of 802.11 with voice traffic, we propose a simple mechanism to dramatically reduce the overhead of the MAC operation, which also results in a reduction of contention. Our proposal, named VoIPiggy, consists in piggybacking 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 system performance. We implement VoIPiggy on COTS devices, and validate its operation against the legacy 802.11 operation.

The main contributions of this paper can be summarized as follows: (i) We propose a mechanism, VoIPiggy, to improve the general performance of WLANs when voice traffic is present; (ii) We describe the implementation of VoIPiggy using COTS devices; and (iii) we present an extensive performance evaluation in a large testbed of 33 nodes. These experiments show that VoIPiggy practically doubles the capacity of a WLAN in terms of voice calls.

The rest of the paper is organized as follows. Section II presents the rationale behind our proposal. In Section III we introduce the design of VoIPiggy and compare it to the legacy MAC. Section IV describes the implementation details of VoIPiggy over a COTS-based platform. In Section V we present an extensive performance evaluation. Finally, Section VI summarizes our main results and exposes the directions of our future work.

II. MOTIVATION

The standard operation of 802.11 introduces a large overhead for the case of voice traffic, given its small frame size. To quantify it, let us consider the exchange of two voice frames between 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. 1. According to the figure, the total time $T_s$ required to perform this simplified two-frame exchange using DCF is given by:

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

where $T_{\text{plcp}}$ 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.

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

This extremely simple analysis serves to illustrate that 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 piggybacks the voice frames over the MAC acknowledgments.

### Algorithm 1 STA operation

```plaintext
1: while true do
2:   while (Pck.out == null & !Rx) do
3:     Listen
4:   end while
5:   if queue.out == empty then
6:     i = 1. $T = 0$
7:   if Pck.out == VoIP then
8:     while ($T \leq 25 ms$) do
9:       Listen
10:      if Rx(packet) == VoIP then
11:        Send VoIP+ACK after SIFS
12:      Remove packet from queue
13:     else
14:       Legacy Tx
15:     end if
16:   end while
17: else
18:   Legacy Tx
19:   end if
20: else
21:   Legacy Rx
22: end if
23: end while
```

### Algorithm 2 AP operation

```plaintext
1: while true do
2:   i = 1
3:   while i $\leq$ MaxAttempts do
4:     Wait backoff and listen
5:     Send packet
6:     Wait for ACK or timeout
7:   if ACKreceived then
8:     Remove packet from queue
9:     if length(ACK) > length(legacyACK) then
10:        Extract VoIP frame from the ACK
11:        Send VoIP to the upper layer
12:   end if
13:   Reset CW & Restart
14:   end if
15:   i += CW* = 2
16: end while
17: end while
```

This extremely simple analysis serves to illustrate that 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 piggybacks the voice frames over the MAC acknowledgments.

### III. THE VoIPiggy MECHANISM

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) A station (STA) sends an ACK frame immediately before 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 collisions.

These two observations motivate the design of our mechanism, whose operation is illustrated in the bottom part of Fig. 1. As the figure shows, a SIFS interval 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. As a result, the VoIPiggy mechanism saves airtime, and thus allocating a higher number of voice calls in the network.

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:

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

where $ACK'$ is the length of the modified acknowledgment header. As compared to the $T_s$ values provided in Table I, $T_v$ provides time savings between 55% and 75%.

As we have enlightened 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 busy.

The use of VoIPiggy requires introducing changes in both the AP and the STA. 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, which includes both the acknowledgment for the frame (that 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. We also prioritize voice traffic, thus we will set the minimum contention window of the AP to $CW_{\text{min}} = 2$, which is the minimum allowed by the standard.

At the STA side, 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.

The operation of VoIPiggy is summarized in Algorithms 1 and 2, while its implementation is detailed next.

IV. IMPLEMENTATION DETAILS

In this section we detail the implementation of the most relevant features of VoIPiggy. We describe the overall architecture, the development platform used for our implementation and the modifications to the Linux kernel and the device firmware to implement the VoIPiggy mechanism.

A. Linux 802.11 stack

The 802.11 network stack in Linux spans over three layers: (i) The mac80211 framework that takes care of the operations related to 802.11 traffic; (ii) The device driver, which is a wrapper between the Linux internal 802.11 packet buffers and the physical device; (iii) The device internal logic, namely firmware, which 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 HW interfaces.

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

B. The Broadcom chipset

We use the Broadcom chipset, which 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 to external conditions, such as the arrival of a new frame or the expiration of programmable timers. A representative set of blocks that compose the chipset are presented as follows:

- **Tx FIFO queues**: driven by the DMA controller and deliver outgoing packets composed by the host kernel.
- **Rx Engine (RXE)**: The RXE decodes the signal received from the air, checks the validity of the CRC coefficient and reports the received packet length.
- **Firmware modifications**: At the firmware level, 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 reported by the RXE.
sockets and a Compact Flash socket, to which we attached a queue immediately after its transmission under VoIPiggy operation. Then, our modification properly acknowledges a legacy ACK frame header. Then, our modification properly sets up the TXE when the PE flag is set, by using the values precomputed by the kernel, i.e., bytes to skip, packet length and timing to use to build the PLCP; (iii) Delay outgoing packets when PE flag is set, until a voice frame is received or a maximum threshold $T$ is reached. In this way it is more likely that a voice frame from the AP will arrive and therefore the HOL frame can be piggybacked.

V. EXPERIMENTAL EVALUATION

In this section we experimentally validate our VoIPiggy implementation in a real testbed deployment and compare the results with the legacy operation.

A. Testbed Description

Our testbed is composed of 33 Alix 2d2 devices from PC Engine, as depicted in Fig. 2. 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 with a 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 Section IV.

One of the devices acts as AP, while the rest are stations associated to the AP, distributed as Fig. 2a shows. All nodes are equipped with a 5-dBi omnidirectional antenna and use a transmission power of 27 dBm. Stations are spaced a few meters from each other (squares in Fig. 2a represent 60 cm x 60 cm floor units), and the resulting link quality is excellent for all nodes to communicate with each other.

The deployment is set up under a raised floor (Fig. 2b), which protects devices from physical harm and provides radio shielding to some extent [13]. Configuration and control of the experiments are centralized in a single terminal, not shown in the figure.

For traffic generation we use mgen, which supports the computation of relevant voice 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 daemon over the wired interfaces of the nodes, achieving synchronization with $\mu$s 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, following the behavior of the G.726 codec. 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. After a calibration process, the timeout threshold during which stations wait for a voice packet from the AP is set to $T = 25$ msec.

B. 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 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. 3 for the DL (top of the figure) and UL (bottom part of the figure) directions, for the standard DCF (denoted as Legacy) and for our VoIPiggy mechanisms, for two 802.11b modulation coding schemes, i.e., $R = 1$ 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 unacceptable are $n = 5$ for $R = 1$, and $n = 8$ for $R = 2$, for the case of VoIPiggy these values grow to $n = 8$ and $n = 13$, respectively. Results show that VoIPiggy almost doubles the capacity in a voice-only scenario.

C. Mixed voice-and-data scenario

Here we evaluate the maximum number of conversations in presence of a data flow and the data throughput performance.

Fig. 4 considers the legacy 802.11 MAC and the VoIPiggy, and depicts the data throughput achieved by the data flow for an increased number of voice conversations. For the

---

1PC Engines: http://www.pceengines.ch/


3Precision Time Protocol: http://ptp.sourceforge.net/
In the case of VoIPiggy, we conclude that: (i) The throughput of data decreases linearly with the number of voice flows; (ii) The maximum number of possible conversations with VoIPiggy is equal to 8 and 13, respectively with $R = 1$ Mbps and $R = 2$ Mbps; (iii) 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 leaves more resources for data flows.

**VI. Summary & Future Work**

In this paper we have designed, implemented and evaluated VoIPiggy, 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.

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, which provides a strong empirical support for the adoption of VoIPiggy.

As future work we envision the following tasks: (i) Assessing the performance when using real VoIP clients, i.e.: Skype, Ekiiga or Viber. We have already carried out some preliminary tests with Ekiga, obtaining promising results; (ii) Implementing additional MAC enhancement mechanisms, e.g., enabling piggybacking at the AP; (iii) Testing VoIPiggy with higher modulation coding schemes.

**ACKNOWLEDGMENTS**

The research leading to these results was funded by the EU’s Seventh Framework Programme (FP7-ICT-2009-5) under grant agreement n.257263 and by the the Spanish Ministry of Science and Innovation under grant TEC2010-10440-E. This work was supported in part by the MEDIANET Project (grant S2009/TIC-1468) from the General Directorate of Universities and Research of the Regional Government of Madrid.

**REFERENCES**


